e-Informatica

software engineering journal

2021 volume 15 issue 1

e-Informatica

e-Informatica

software engineering journal

2021 volume 15 issue 1

e-Informatica

Wrocław University
of Science and Technology

Editor-in-Chief
Lech Madeyski (*Lech.Madeyski@pwr.edu.pl, http://madeyski.e-informatyka.pl*)
Editor-in-Chief Emeritus
Zbigniew Huzar (*Zbigniew.Huzar@pwr.edu.pl*)

Faculty of Information and Communication Technology, Department of Applied Informatics
Wrocław University of Science and Technology,
50-370 Wrocław, Wybrzeże Wyspiańskiego 27, Poland

e-Informatica Software Engineering Journal
www.e-informatyka.pl, DOI: 10.37190/e-inf

Editorial Office Manager: Wojciech Thomas

Typeset by Wojciech Myszka with the $\LaTeX\,2_\varepsilon$ Documentation Preparation System

OFICYNA WYDAWNICZA POLITECHNIKI WROCŁAWSKIEJ
Wybrzeże Wyspiańskiego 27, 50-370 Wrocław
www.oficyna.pwr.edu.pl;
e-mail: oficwyd@pwr.edu.pl; zamawianie.ksiazek@pwr.edu.pl

ISSN 1897-7979

Print and binding: beta-druk, www.betadruk.pl

Editorial Board

Contents

e-Informatica Software Engineering Journal, Volume 15, Issue 1, 2021, pages: 7–27, DOI 10.37190/e-Inf210101

Fixing Design Inconsistencies of Polymorphic Methods Using Swarm Intelligence

Renu George*, Philip Samuel*
Department of Computer Science, Cochin University of Science and Technology, India
renugeorge@ceconline.edu, philips@cusat.ac.in

Abstract

Background: Modern industry is heavily dependent on software. The complexity of designing and developing software is a serious engineering issue. With the growing size of software systems and increase in complexity, inconsistencies arise in software design and intelligent techniques are required to detect and fix inconsistencies.
Aim: Current industrial practice of manually detecting inconsistencies is time consuming, error prone and incomplete. Inconsistencies arising as a result of polymorphic object interactions are hard to trace. We propose an approach to detect and fix inconsistencies in polymorphic method invocations in sequence models.
Method: A novel intelligent approach based on self regulating particle swarm optimization to solve the inconsistency during software system design is presented. Inconsistency handling is modelled as an optimization problem that uses a maximizing fitness function. The proposed approach also identifies the changes required in the design diagrams to fix the inconsistencies.
Result: The method is evaluated on different software design models involving static and dynamic polymorphism and inconsistencies are detected and resolved.
Conclusion: Ensuring consistency of design is highly essential to develop quality software and solves a major design issue for practitioners. In addition, our approach helps to reduce the time and cost of developing software.

Keywords: UML models, software design inconsistency, polymorphism, particle swarm optimization

1. Introduction

Today′s biggest industry is software industry in terms of manpower, complex interactions and changing tasks with evolving designs. The way people coordinate activities and work has seen a major transformation since the use of software in industries. With the increasing relevance of software in industries, software development has become more complex. Software changes are frequent due to evolution, agility and adaptability. Customized software is used to increase productivity in industries and quality of the software is a prime concern. Design and development of quality software is a major challenge for software developers and many a times, the process is manual. Artificial intelligence (AI) techniques can replace many of these manual efforts to make the development of software easier and cost effective.

Artificial intelligence replicates human decision making techniques to make machines more intelligent. Software development involves several complex human decision makings that deal with the task of designing, implementing and deploying complex systems. Software engineering problems can be represented as optimization problems. Search based software systems use optimization techniques and computational search techniques to solve problems in software engineering [1]. Although search based systems address many problems in software requirements and design, design verification is not yet addressed

Submitted: 08 April 2020; Revised: 02 November 2020; Accepted: 14 December 2020; Available online: 02 February 2021

[2, 3]. Particle swarm optimization (PSO) is an optimization technique based on population with computational intelligence [4]. Self Regulating Particle Swarm Optimization (SRPSO) is an improved version of PSO that provides optimum solutions by incorporating the best strategies for human learning [5]. We present an intelligent approach based on SRPSO to solve the inconsistency in polymorphic methods during the software system design.

The modelling language widely used for requirements modelling and documentation of the system is Unified Modeling Language (UML). UML models handle the complexity of the system by expressing different views with different diagrams that consist of a number of interrelated model elements. The interrelated design diagrams contain redundant information in the overlapping model elements. Hence, the probability of occurrence of design inconsistencies is more. The diagrams of a single system representing the static and dynamic aspects should be consistent and not contradictory [6]. Explicit mechanisms are required to verify the consistency of redundant information present across the diagrams [7, 8]. Generally, models are constructed for a specific application and the models are eventually implemented, usually in an object oriented programming language. Validating the models for consistency in the design phase guarantee that the design inconsistencies are not carried over to the code generation phase of software development. Automated consistency checking during the design phase ensures software quality, reduced development and maintenance cost and less time to market. Inconsistent design results in incorrect code, design rework, failure to meet timelines, and increase in cost of production.

Polymorphism is one of the key concepts that determine the quality of object oriented software systems [9]. Polymorphism enables different behavior to be associated with a method name. New methods with different implementation can be created with the same interface and the amount of work required to handle and distinguish different objects is reduced [9]. The result of execution of a polymorphic method depends on the object that executes the method and produces different results when received by different objects. The advantages of designing multiple methods with the same name make polymorphism an efficient approach during software design. We define an inconsistency related to object interactions in polymorphic and non-polymorphic methods: method-invocation inconsistency. Inconsistency exists if the method invocations are bound to a wrong class in the sequence diagram, i.e., the method is not invoked on an object of the class in which the method is defined. Inconsistencies in polymorphic method invocations cannot be identified by validating the method names as all polymorphic methods have the same name. Hence, detection of method invocation inconsistency in polymorphic methods requires more effort than non-polymorphic methods. As the design complexity increases, manual verification of inconsistencies in polymorphic methods is not practical. Intelligent techniques that require expertise are required to detect and solve the inconsistencies.

Method-invocation inconsistency occurs when a polymorphic or non-polymorphic method is invoked on a wrong object in the sequence diagram. The existing approaches of detecting inconsistencies in method invocations specified in [10–15] do not mention inconsistencies in polymorphic methods. Although polymorphism has a number of advantages, serious flaws may occur due to inconsistencies. Programmers may find it a challenging task to understand all the interactions between sender and receiver objects [16]. Understanding polymorphic codes is hard and therefore, fault-prone. Usually inconsistencies related to polymorphic method invocations are difficult to identify during testing phase. Separate tests are required for each polymorphic method binding. Identifying and testing all possible bindings of certain polymorphic references is difficult thereby increasing the chances of errors [17]. Inconsistent polymorphic behaviours may cause huge financial problems when detected.

Software design is prone to errors and design imperfections have a significant effect on software quality. Software failure can be attributed to various factors starting from requirements gathering to testing and poor quality management [18]. Inconsistencies in the design lead to the generation of defective software. One of the major activities in ensuring quality involves detection and removal of defects in the design. As the errors are carried over from the software design phase to the development phase, the cost incurred in fixing the error also increases. Defect detection during the design phase significantly prevents propagation of errors to further stages of software development and reduces development cost [19, 20]. Hence, code generation is based on consistent designs. This facilitates generation of software with fewer faults and improves the quality of the software generated. Development of software with fewer faults reduces the maintenance cost of the software. Cost increases with the delay in detecting and correcting the error. The cost of detecting defects after release is 30 times more than the cost of detecting defects in the analysis and design phase [19]. Therefore, inconsistency detection in the software design phase is inevitable for the development of accurate and quality software. We propose an intelligent approach to detect and fix inconsistencies during the design phase of software development. Inconsistencies are detected and handled with a fitness function by generating fitness values for each polymorphic and non-polymorphic method in the class diagram and sequence diagram. Inconsistencies are handled by maximizing the fitness values of methods subject to the constraint that the methods are invoked on the right classes. The proposed automated intelligent approach for consistency checking during the design phase facilitates generation of software with fewer faults, improves software quality, and reduces development and maintenance cost.

The organization of the paper is as follows. The related works in the areas of consistency checking and the various applications of PSO and its variants is presented in Section 2. Section 3 deals with the inconsistencies in polymorphic methods. The architecture of the consistency checking system is described in Section 4 and the implementation of the proposed approach is presented in Section 5. Results and discussion are presented in Section 6, threats to validity is presented in Section 7 and Section 8 concludes the paper.

2. Related work

The section presents the consistency handling techniques available in the literature and the applications of PSO techniques to find optimal solutions in software development and industries.

Inconsistencies in the design may result in the failure of a software project. The problems of establishing and maintaining consistency is discussed in [21]. The authors state that it is impossible to avoid inconsistency and more flexibility can be obtained by using tools that manage and tolerate inconsistency. A tool that detects inconsistency and locates the choices for fixing inconsistency is proposed in [10]. Model profiling is used to determine the model elements affected while evaluating a rule; a set of choices for fixing the inconsistency is proposed and the designer decides the choice for fixing the inconsistency. The method proposed in [11] fixes inconsistencies in class, sequence and statechart diagrams by generating a set of concrete changes automatically. The work focuses on deciding a method to fix inconsistencies. An approach that performs real time detection and tracking of inconsistencies in class, sequence and state chart diagrams is presented in [12]. Consistency checks are initiated during a model change.

The algorithm proposed in [13] performs consistency check on class and sequence diagrams based on the syntax specified and generates a sequence of Relational Calculus of Object Systems (rCOS) class declarations. Inconsistencies in well-formed class and sequence diagrams are detected with an algorithm based on breadth first search technique. Transformation, refactoring, merging or repair of models result in changes in the model and during consistency checking it may lead to performance problems. An automated approach with tool support to re-validate parts of the design rule affected by model transformation or repair is proposed in [22]. Although the paper mentions inconsistency in sequence and class diagrams, the focus is on improving the performance of incremental consistency checking by identifying parts of the model affected by model changes. A prototype tool developed using a UML based approach to handle impact analysis is proposed in [14]. Consistency check is performed on the UML diagrams, difference between the two versions is identified and the model elements that are directly or indirectly affected by the changes are determined. The focus of the paper is on changes and its impact, i.e., which model elements are affected by the change. Instant detection of consistency between source code and design models is performed in [23] and a live report of the consistency status of the project is provided to the developers.

A classification of model repair techniques based on features is presented in [24]. The focus is on proposing taxonomy for model repair techniques and not on inconsistency detection and causes of inconsistency. The paper [15] proposes a method for automatic generation of executable and concrete repairs for models based on the inconsistency information, a set of generator functions and abstract repairs. An automated planning approach based on artificial intelligence is proposed in [25] to resolve inconsistencies. A regression planner is implemented in Prolog. The approach is restricted to detection of structural inconsistencies in class diagrams only.

A review of the consistency management approaches available in the literature is presented in [26]. The works described does not address inconsistencies related to polymorphic methods. Object Constraint Language (OCL) rules are specified for consistency checking of UML model in [27], the approach does not address polymorphic methods. Consistency rules to detect inconsistencies in method invocations between sequence and class diagrams are presented in [28], but no approaches are presented to detect and fix inconsistency. A method to detect inconsistencies between state diagrams and communication diagrams using the language Alloy is presented in [29].

Soft computing techniques find its application in providing solutions to problems in industries. PSO is used to minimize the cost of heating system [30], to assign applications to resources in the cloud [31], in job-shop scheduling [32], in networking [33], power systems [34, 35], signal processing [36], control system [37] and many more. PSO is also applied to find effective solutions to problems in software development. PSO is applied to UML class diagram and an algorithm for class responsibility assignment problem is presented in [38]. The PSO method reassigns the attributes and methods to the different classes indicated in the class diagram. The application of SRPSO and PSO in detecting and resolving inconsistencies in class attribute definitions is presented in [39, 40]. The fitness value determines the consistency of attributes and the PSO and SRPSO algorithm iterates to fix inconsistency by optimizing the fitness value of attributes. The papers deal with fixing inconsistencies in attribute definitions only. The performance of SRPSO algorithm is better than PSO in term of statistical evaluation parameters and convergence. An SRPSO based approach to fix state change inconsistencies in state diagrams and sequence diagrams is proposed in [41]. Inconsistencies are detected and fixed with a fitness function.

An optimization based approach using PSO and simulated annealing to find transformation fragments that best cover the source model is proposed in [42]. PSO is applied to achieve high structural code coverage in evolutionary structural testing by generating test cases automatically [43]. Parameter estimation using PSO to predict reliability of software reliability growth models (SRGM) is described in [44]. During testing, faults are detected and a mathematical model SRGM, models the properties of the process. A comparative study of metaheuristic optimization framework is proposed in [45] and the study states that a wider implementation of software engineering practices is required.

The application of PSO in diverse areas of engineering has yielded better results over existing methods, but works that describe the application of PSO in the design phase for software design consistency checking is rare. Although consistency checking of UML models is a widely discussed problem and different techniques to detect and fix inconsistencies are available in the literature, techniques that perform consistency checking of polymorphic methods are rarely reported. We present an intelligent approach that detects inconsistencies with a fitness function. Inconsistencies are fixed by remodelling the sequence diagram method invocations during iterations of the SRPSO algorithm. Our approach efficiently detects and fixes the inconsistencies.

3. Inconsistencies in polymorphic methods

Polymorphism is an important feature of object oriented programming that provides simplicity and flexibility to the design and code. It enables different behaviour to be associated with a method name. Polymorphism keeps the design simple, flexible and extensible [46]. New methods with different implementation can be created with the same interface and the amount of work required to handle and distinguish different objects is reduced. Each polymorphic method has a class specific way to respond to a message. Polymorphic methods execute different subroutines depending on the type of object they are applied to. Inconsistency occurs if the method is invoked on a wrong class. Two methods of implementing polymorphism are (a) static binding: methods have the same name, different signature and different implementation (b) dynamic dispatch: methods have the same name, same signature and different implementation [47]. Static binding occurs with method overloading at compile time and the method to be invoked is determined from the signature of the method call. Dynamic dispatch is related to inheritance hierarchy. Method overriding provides a superclass/subclass inheritance hierarchy allowing different subclass implementation of inherited methods [48, 49]. The overriding methods represent different functionalities and require different algorithms [50]. The exact method to which the method call is bound is known only at run time. Method overriding is implemented with dynamic dispatch [49].

Inconsistency in UML models occurs when two or more diagrams describe different aspects of the system and they are not jointly satisfiable. Any method invoked on an object in the sequence diagram should be defined in the class instantiated by the receiving object. The rule is part of the UML well-formedness principle. There is scope for many subtle errors with polymorphism since a method name occurs in more than one class. The exact operation to be performed is determined from the data types of the arguments in static polymorphism. The same signature is used by more than one class in dynamic polymorphism and determining whether the correct method is invoked in the sequence diagram is an issue. Understanding polymorphic codes is hard and therefore fault-prone [16]. Hence, inconsistency detection during the design phase has become inevitable for the development of accurate software [16]. We propose an intelligent approach using SRPSO algorithm to detect and fix method-invocation inconsistency in polymorphic methods. Method invocation inconsistency is identified from the signatures of the class diagram and sequence diagram methods in static polymorphism. The method signatures are the same for all polymorphic methods in dynamic polymorphism and hence more difficult. Inconsistency is detected from the guard condition for message invocation in the sequence diagram and precondition for the method in the class diagram.

The inconsistencies are illustrated with the UML models 3DObject and ThreeDObject represented in Figures 1 and 2, respectively. The class diagram and sequence diagram for the UML model 3DObject is represented in Figure 1 . The model provides an example of static polymorphism. The class diagram consists of 4 classes. A generalized class ThreeDShape is defined with an attribute Area of type float. The classes Sphere, Cuboid and Cylinder are specializations of the class ThreeDShape. The methods computeArea() and perimeter() defined in the classes Sphere, Cuboid and Cylinder are polymorphic since methods with the same name and different signature are defined. The method vertices() defined in the class Cuboid is non-polymorphic. The sequence diagram represents the method invocations to compute the area of the objects. The class Cuboid has a method computeArea(l, b, h) with signature computeArea(int, int, int). Similarly, the signatures of the method computeArea() defined in the classes Sphere and Cylinder are computeArea(int) and computeArea(float, fint), respectively. The signatures of the method computeArea() invoked on the objects of classes Cuboid, Sphere and Cylinder in the sequence

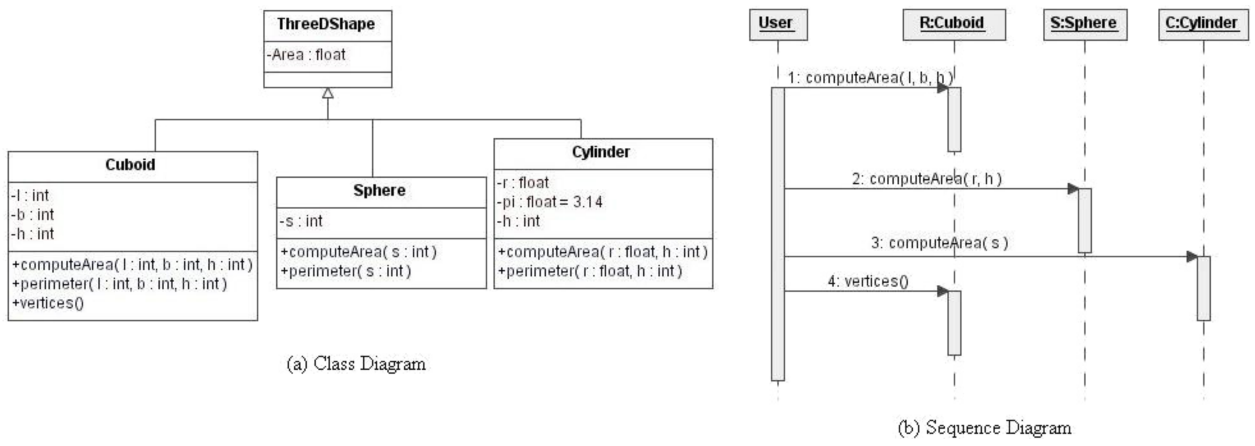


Figure 1. Class Diagram and Sequence Diagram for UML Model for 3DObject

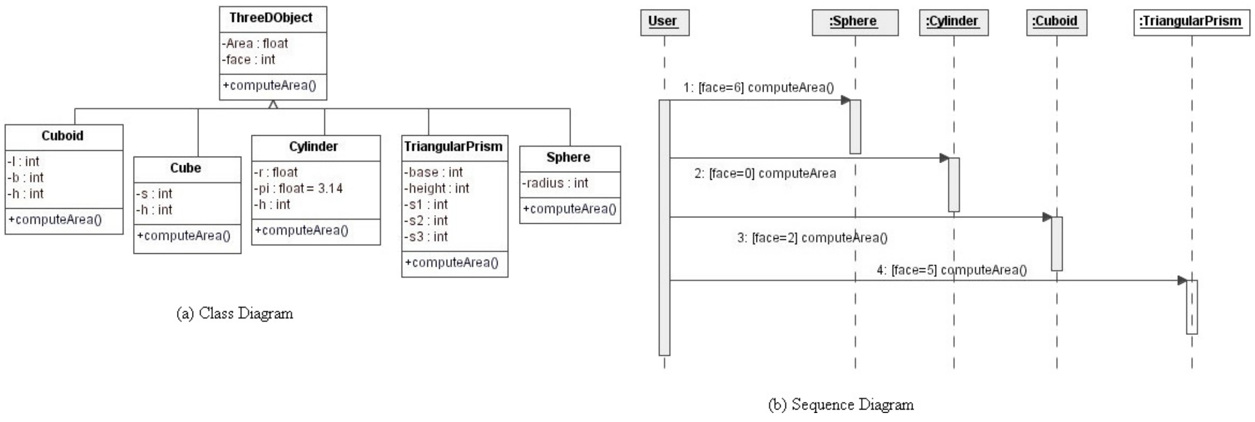


Figure 2. Class Diagram and Sequence Diagram for UML Model for 3DObject

diagram are computeArea(int, int, int), computeArea(float, int) and computeArea(int). The invocations of the polymorphic method computeArea(l, b, h) and the non-polymorphic method vertices() are consistent whereas the invocations of the polymorphic methods, computeArea(s) and computeArea(r, h) are inconsistent. The inconsistencies, if unnoticed will result in a wrong value for area. Inconsistencies are detected by computing the fitness values of methods. The fitness value computation to detect inconsistency is represented in Table 1.

A UML model ThreeDObject representing dynamic polymorphism is depicted in Figure 2. A method computeArea() and two attributes *Area* and *face* are defined in the class ThreeDObject. The method is overridden in the child classes since the method of computing area depends on the shape of the object. The attribute *face* represents the number of faces possessed by an object. Sphere has no face, Cylinder has two faces, Cuboid and Cube have 6 faces and TriangularPrism has 5 faces. Constraints are defined for the methods and expressed as preconditions in

Table 1. Fitness values of methods in UML Models 3DObject and ThreeDObject

Method name	CD Class	SD Class	Fitness value	UML Model
computeArea(r, h)	Cylinder	Sphere	0.9375	3DObject
computeArea(l, b, h)	Cuboid	Cuboid	1	3DObject
vertice()	Cuboid	Cuboid	1	3DObject
computeArea(s)	Sphere	Cylinder	1.11	3DObject
computeArea()	Cylinder	Cylinder	1	ThreeDObject

object constraint language. The preconditions for the method computeArea() in the classes Cuboid, Cube, Sphere, TriangularPrism and Cylinder state that the value of the attribute face should be equal to 6, 6, 0, 5 and 2, respectively. As the signatures of all the methods involved in dynamic polymorphism are the same, it is impossible to detect inconsistency by comparing the method signatures. The guard conditions and the preconditions are compared and method invocation inconsistency is detected with the method computeArea() invoked on the objects of classes Cuboid, Sphere and Cylinder. The method computeArea() invoked on the object of class Sphere should satisfy the guard condition face = 6 which is not true resulting in run time errors. The invocation of the method computeArea() on the object of class TriangularPrism is consistent as the value of the attribute face in the guard condition and precondition is equal to 5.

The inconsistent method invocations in Figures 1 and 2 result in wrong value for Area. If the models are used in the cost estimation of buildings, the estimated cost will be computed with wrong values of area. The cost estimation will produce a wrong value affecting the feasibility of the project. Since the design errors are propagated to the code generation phase, the software generated will have errors. Identifying the source of errors in the code and fixing the errors is more difficult, time consuming and costly than detecting the errors in the software design. Errors detected in the testing phase may delay the software project. The errors identified during the testing phase or after delivery of the software product increases the time to market as well as development and maintenance cost of the software.

4. Architecture of the consistency handling system

PSO is an intelligent algorithm that can be used in scientific and engineering area [51]. Consistency checking is formulated as an optimization problem with a maximizing fitness function that operates on the diagram specification. An optimization problem maximizes or minimizes a fitness function subject to the condition that the constraints are satisfied. In our approach the fitness function represents the consistency and completeness of method invocations. The aim of the SRPSO algorithm is to optimize the consistency of polymorphic method invocations in sequence diagram subject to the constraint that the methods are invoked on the right classes. The SRPSO algorithm is preferred because it does not require transformation of models and can be directly applied on UML model specification. The inconsistent particles are guided by the best particles to achieve consistency and hence search speed is high [52].

The architecture of the system to perform consistency checking using SRPSO is described in Figure 3. The algorithms are implemented in Java running on a windows platform. The consistency checking system comprises of UML tool to model the requirements, parser to generate diagram specification and consistency checker to

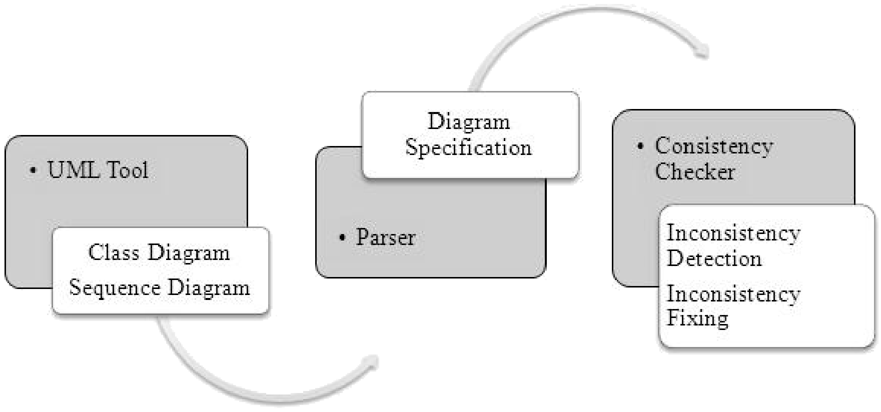


Figure 3. Architecture of Consistency Checking System

detect method-invocation inconsistency and fix the inconsistency using SRPSO algorithm.

4.1. UML tool

The requirements of the system to be designed are gathered and modelled into graphical representations using a UML tool. Several UML modelling tools like Magic Draw, Rational Software Architect, Agro UML, Papyrus, etc. are available for modelling software. Our method can be integrated with any tool that give XMI format. The models are saved in XMI format. The static and dynamic aspects are represented using class diagram and sequence diagram. Class diagrams represent the information regarding the classes required to implement a system and the type of relationship that exists between the classes. The attributes and operations describe the properties and behaviour of the objects of a class. Preconditions associated with method invocations are also represented. The preconditions of the overridden methods in the super class and subclass are different [50]. Sequence diagram represents the dynamic aspects by portraying the interactions in the form of messages/methods between objects and the ordering of the interactions to produce a desired outcome. Polymorphic behaviour can be represented using a sequence diagram by controlling the polymorphic invocations with guard conditions.

4.2. Parser

The parser parses the UML model and produces specifications of the diagrams. We have used the Document Object model (DOM) parser to parse the diagrams saved in XMI format. A class diagram specification comprises of the classes, the type of association between the classes, attributes and methods of each class and the preconditions for method invocations. The sequence diagram specification consists of the objects in the sequence diagram, messages, sender and receiver of each message, the guard conditions on the message invocations and the order of method invocations.

4.3. Consistency checker

The design inconsistencies in polymorphic method invocations are detected by consistency checker module. Although the focus is on detection of inconsistencies in polymorphic methods, the algorithm detects inconsistencies in polymorphic and non-polymorphic methods. The specifications of class and sequence diagrams are input to the consistency checker. Inconsistencies are detected by a fitness function. The inconsistency is resolved by reassigning methods with the SRPSO algorithm. Consistency checking is a two-step process: a) inconsistency detection and b) inconsistency fixing.

4.3.1. Inconsistency detection

The fitness function, f_s computes the fitness value of the methods to detect inconsistency. The fitness value is computed as a function of the class name, method signature and properties of the method. The sequence diagram method is defined in terms of its properties like name, id, parameters, sender, receiver, guard and a number that represents the message order. Each method has a specific value for a property (denoted as weight) and each position in the vector corresponds to one property of the method. The values for the properties are set as 5, 3, 5, 5, 5, 4 and 3, respectively. The fitness function f_s computes the fitness value of each sequence diagram method. A method invocation is classified as inconsistent if the fitness value is not equal to one. The fitness function is defined with equation 1 as

$$f_s = \frac{\left(\sum_{i=1}^{m} t_i * w_i) * (w_n + w_{cs} + \sum_{k=1}^{n} w_k * p_k\right)}{W * \left(\sum_{j=1}^{q} w_j * p_j\right)} \tag{1}$$

where t_i represents the property i of method specification, w_i represents the weight of the property t_i, w_n represents the weight value associated with the method name n, p_j and w_j represents the position and weight of parameter j of the method n in the class diagram, p_k and w_k denotes the position and weight of the parameter k of the method n in the sequence diagram, w_{cc} represents the class name of the method in the class diagram, w_{cs} represents the class name of the object on which the method is invoked in sequence diagram and W represents the weight assigned to a complete method specification. The value of W is set as 30. A complete method specification has values for all its properties. Unique values are assigned as the weights for method names, class names and data type of parameters. Distinct method names, data types and classes have distinct weight. All polymorphic methods have the same weight value for name and any numerical value can be selected to correspond to w_n. The UML model in Figure 1 has a polymorphic method with name computeArea. The value of w_n is set as 5. The value of w_n for the method names vertices() and perimeter() are set as 3 and 4, respectively. The classes are assigned weights in the range $[1 \ldots n]$ where n is the number of classes. Each class has a unique weight. A class name present in both the class diagram (w_{cc}) and sequence diagram (w_{cs}) has the same weight. The UML model in Figure 1 has four classes Cuboid, Sphere, Cylinder and ThreeDShape and the weight values for the classes Cuboid, Sphere, Cylinder and ThreeDShape are 1, 2, 3 and 4, respectively. The weight value of parameter is defined as the number of bytes required for the storing the data type of the parameter and the weights of char, int and float are defined as 1, 2 and 4, respectively. The weights assign unique numerical values to the method name, class name and data types of the parameters. The fitness value computation for the methods in the sequence diagram of Figures 1 and 2 is illustrated in Table 1.

Method invocation inconsistency is detected with the methods computeArea(r, h) and computeArea(s) since the fitness values of the methods are not equal to one. The methods computeArea(l, b, h) and vertices() are consistent since the fitness values are equal to one. Although inconsistency is detected from the fitness value in static polymorphism, fitness value alone does not reveal inconsistency in dynamic polymorphism. Irrespective of the object on which the method is invoked, the fitness value of the method computeArea() in the UML model ThreeDObject is one since the method is overridden in the child classes. Hence, validation of the guard condition and method precondition is necessary. The guard condition and precondition are represented as tuples consisting of attribute, operator-value pairs. Depending on the precondition, there can be more than one operator-value pair. The tuples are compared to identify inconsistency. The tuple corresponding to the guard condition for the method computeArea() in Figure 2 in the sequence diagram invoked on the object of class Cylinder is (*face*, (=, 0)). The tuple representation for the precondition of the method computeArea() in the class Cylinder is (*face*, (=, 2)). There is a mismatch in the value of the attribute *face* and method invocation inconsistency is detected.

4.3.2. Inconsistency fixing with SRPSO

The inconsistency is resolved by identifying the right classes and remodelling the sequence diagram by replacing the inconsistent method invocations with consistent method invocations using SRPSO. To identify the right class, we compute the cohesion of the attributes of the inconsistent method to all the classes in the class diagram. The inconsistent method is reassigned to the class with the highest cohesion value. The cohesion value between the method attributes and the class attributes is computed for each method-class pair. The method attributes MA(m) of method m are derived from the parameters of the method. The class attributes of class C, CA(C) are obtained from the class specification. The cohesion value of a method m to class C is computed using equation 2 as

$$\text{cohesion}(m, C) = \frac{n(\text{CA}(C) \cap \text{MA}(m))}{n(\text{MA}(m))} \tag{2}$$

where CA(C) represents the attributes defined in class C, MA(m) represents the attributes of method m and n represents the number of attributes. The SRPSO algorithm iterates until all the method definitions are complete and consistent. The sequence diagram is remodelled during iterations of the SRPSO algorithm. With static binding, the cohesion value determines the class to which an inconsistent method is to be reassigned whereas in dynamic binding, the cohesion value and guard condition together determine the class to which the method belongs.

5. Consistency handling with SRPSO algorithm

SRPSO is a bio-inspired metaheuristic technique that can provide better results than exact techniques even with increased size of search space. Metaheuristic techniques are more effective in finding software errors utilizing less number of resources when compared with exact techniques [53]. SRPSO is an intelligent, optimization procedure in which the solution space contains a swarm of particles and the optimum value is attained by an iterative process of updating generations. The particles occupy a position in the solution space. They have a velocity, a fitness value and the particles update their velocity and position based on the direction of a) the previous velocity, b) the personal best position and c) position of the global best [54]. The fitness function determines how close a particle is to the optimum solution by computing the fitness value. The velocity directs the movement of particles and during each iteration of the SRPSO algorithm the particles compute their new velocity. The position is updated using the new velocity and with each position update the particle moves to a better position. The process is iterated until an optimum solution is reached.

5.1. Fitness function

The fitness function is an integral part of the SRPSO algorithm and it determines how close a particle is to the optimum solution. We have defined a maximizing fitness function, f_s to detect and fix method invocation inconsistency. The fitness function is defined with equation 1. The consistency and completeness of a sequence diagram method is computed using the fitness function. The invocations of inconsistent methods are removed from the sequence diagram and the inconsistent methods are added to the set of inconsistent methods (IM).

5.2. Particle creation

The search space of the SRPSO algorithm is initialized with particles. The proposed approach focuses on inconsistency in polymorphic and non-polymorphic method invocations and hence, the methods invoked in the sequence diagram are treated as particles. A sequence diagram method is specified using a set of properties and is represented as a vector. The representation of the sequence diagram method (SeqM) is SeqM = [name id param sender receiver guard number]

The representation of SeqM consists of a method name, a unique xmi id, the parameters, sender class of the method, receiver class of the method, guard condition for method invocation and number representing the message order in the sequence diagram. Each method has a specific value for a property and each position in the vector corresponds to one property of the method. The values for the properties are fixed as 5, 3, 5, 5, 5, 4 and 3, respectively. Any numerical value can be used to represent a property. The restriction is that the value of W should be equal to the sum of the numerical values assigned to the properties. The inconsistent methods in the set IM are represented as particles.

5.3. Velocity and position update

The particles in the search space are characterized by a position and velocity. A particle is defined in terms of its properties and in our approach; the position of a particle represents the number of properties defined for the particle. The specification of the inconsistent particle initially has only one property, *name* and hence, the value of position is one. As the iteration progresses, depending on the value of velocity the particle specification will be updated with its properties like id, sender, receiver etc. The number of properties of the particle to be updated in one iteration is determined by the value of velocity. If the value of velocity is one, one property will be added to the particle specification and position will be incremented by one. Velocity of the best particle is computed with equation 3, velocity of the rest of the particles with equation 4, position is updated using the equation 5 and inertia weight with equation 6.

$$V_k(t+1) = \omega_k + V_k(t) \tag{3}$$

$$V_k(t+1) = \omega_k + V_k(t) + a_1 * r_1 * (\text{pBest}_k - X_k(t)) + a_2 * r_2 * p_{so} * (\text{gBest} - X_k(t)) \tag{4}$$

$$X_k(t+1) = X_k(t) + V_k(t+1) \tag{5}$$

$$\omega_k(t) = \begin{cases} \omega_k + \eta\Delta\omega & \text{for best particle} \\ \omega_k - \Delta\omega & \text{otherwise} \end{cases} \tag{6}$$

where $V_k(t)$ represents the velocity of particle k at time t, a_1 and a_2 are the acceleration coefficients, r_1 and r_2 are the random numbers, $X_k(t)$ represents the position of particle k at time t, pBest_k represents the personal best of particle k and gBest the global best of all the particles in the swarm, p_{so} is the perception for the social cognition, ω_k is the inertia weight of the k^{th} particle, $\Delta\omega = (\Delta\omega_I - \Delta\omega_F)/Itr$, $\Delta\omega_I = 1.05$ and $\Delta\omega_F = 0.5$, Itr is the number of iterations, and $\eta = 1$ is the constant to control the rate of acceleration.

5.4. Stopping criteria

The SRPSO algorithm resolves method invocation inconsistency. The algorithm iterates until method invocation inconsistency is resolved or the number of iterations reaches a maximum limit. We have defined a variable method consistency count (MCC) that keeps track of the number of methods with consistent and complete invocations. MCC is incremented if fitness value of a method is equal to one. If MCC is equal to the number of inconsistent methods in the set IM, method invocation inconsistency is resolved.

5.5. Algorithm

The consistency checking algorithm for polymorphic methods is outlined in algorithm 1.

Algorithm 1: Consistency Checking

```
Begin
Initialize SRPSO parameters, IM = $\phi$
for each method, m in sequence diagram do
identify sender class, SC($m$) and receiver class, RC($m$)
compute $f_s(m)$ with equation 1
Case I: $f_s(m) == 1$
    if guard conditions do not match
        IM = IM $\cup$ {$m$}
    endif
Case II: $f_s(m) \neq 1$
    IM = IM $\cup$ {$m$}
endfor
identify the receiving class
for each method, $m$ in set IM do
    identify method attributes, MA($m$)
    for each class, $C_i$ in class diagram do
        determine class attributes CA($C_i$)
    endfor
    compute cohesion($m$, $C_i$)
    RCnew = $C_j$ where $C_j$ = max(cohesion($m$, $C_i$), $i = 1$ to number of classes
    delete sequence diagram invocation for the method $m$
endfor
Initialize the search space with particles in the set IM
repeat
    for each particle $k$ in IM do
        compute fitness of particle $k$
        if ($f_s(X_k) > f_s(\text{pBest}_k)$)
            $\text{pBest}_k = X_k(t)$
        endif
        if ($f_s(X_k) > f_s(\text{gBest})$)
            gBest = $X_k(t)$
        endif
        Compute inertia weight using equation 6
        Update velocity of gBest particle using equation 3
        for each particle except gBest particle do
            Generate random number, $r$ between 0 and 1
            if ($r > 0.5$)
                $p_{so} = 1$ else $p_{so} = 0$
            endif
            Compute velocity using equation 4
        endfor
        if($V_k(t+1) > 1$
            $V_k(t+1) = 1$ else $V_k(t+1) = 0$
        endif
       Update position using equation 5
```

```
        **if** $(f_s(X_k(t+1)) == 1)$
            Increment MCC
        **endif**
    **endfor**
    Increment iteration count, *Itr*
**until** *Itr* = maxCount or MCC = number of inconsistent methods
**End**
```

The algorithm initializes the search space with particles and SRPSO parameters. The acceleration coefficients are set as 1.49445 [5], *Itr* is initialized as zero, W is set as 30 and maxCount is set as 35. The set IM is initialized to null. The algorithm computes the fitness values of methods. The guard conditions and preconditions of methods are also validated. The inconsistent method names are added to the set IM and the inconsistent method invocations are removed from the sequence diagram.

To fix the inconsistency, the inconsistent methods in the set IM are treated as new particles and the position of the particles are initialized. The cohesion of each method in the set IM to the different classes of the class diagram is computed to identify the new receiving class, RCnew. The class with the maximum cohesion value is identified as RCnew. The receiving class is identified from the precondition and cohesion value in dynamic polymorphism.

The newly created particles are inconsistent since its properties are not completely specified. Initially, all the inconsistent particles have only one property, its name. The fitness values of the particles in their current position are computed using the fitness function, f_s. If the current position is better than the personal best (pBest) position of the particle, the personal best position of the particle is updated. If the current position is better than the global best (gBest) position of all the particles in the swarm, the global best position is updated. New velocity and position of the particles are computed. Depending on the velocity value, properties such as id, sender, receiver etc. are added to the particle specification. The velocity component determines the number of properties to be updated in one iteration. If the fitness value is equal to one, the method consistency count is incremented. The velocity, position, fitness value, pBest and gBest values of all the particles in the set IM are updated during an iteration of the algorithm. The iteration count is also incremented. The SRPSO algorithm iterates until the method consistency count is equal to number of particles in the set IM or maximum number of iterations is reached. The updation of the properties of the inconsistent particles ensures that inconsistencies are resolved and the method specification is complete. The SRPSO algorithm efficiently detects and resolves inconsistency.

6. Results and discussions

The consistency checking algorithm is applied to the UML models to detect method invocation inconsistency. The UML model in Figure 1 contains the polymorphic method computeArea. The method-invocation inconsistency detection module detects two inconsistent methods: computeArea(r, h) and computeArea(s) by computing the fitness values of the methods. The inconsistent methods are added to the set IM and the sequence diagram invocations of the inconsistent methods are removed. The attributes required for the implementation of the method are derived from the parameters of the method. The cohesion of the method attributes to the different classes in the class diagram is computed. The cohesion values of the inconsistent methods to different classes are represented in Table 2.

Table 2. Cohesion Value for UML Model 3DObject

Method	Class Name		
	Cube	Cuboid	Cylinder
computeArea(r, h)	0.0	f 0.5	1.0
computeArea(s)	1.0	0.0	0.0

The class Cylinder has the highest cohesion value for the method computeArea(float, int) and the class Cube has the highest cohesion value for the method computeArea(int). The receiving class of the inconsistent method computeArea(r, h) is identified as class Cylinder and the new receiving class of the method computeArea(s) is identified as class Cube. The sequence diagram methods are specified with a set of properties. On detecting inconsistency, the properties related to the method invocation of the inconsistent methods are also deleted. The inconsistency is fixed during iterations of the SRPSO algorithm. During each iteration of the SRPSO algorithm, the specification of the sequence diagram method in the set IM is updated by adding the properties of the methods. The approach ensures that method invocation inconsistency is resolved and the method specification is complete. The algorithm terminates when MCC becomes equal to the number of inconsistent methods or when *Itr* reaches the maxCount.

A graph representing the fitness value of the inconsistent methods computeArea(r, h) and computeArea(s) during different iterations of the SRPSO algorithm with acceleration coefficient values equal to 1.49445 is represented in Figure 4. The method computeArea(s) has a fitness value 0.0925 during the first iteration of the algorithm. As the iteration count increases, the fitness value of the particle increases. In iteration 8, the fitness values of the two inconsistent particles become one and the UML model 3DObject is consistent in terms of polymorphic method invocation and specification. The fitness value of the inconsistent method in the UML model ThreeDObject is represented in Figure 5. The algorithm is implemented with acceleration coefficient values equal to 1.49445 and converges in 8 iterations.

The result of implementation of the algorithm is represented in Figure 6. The XMI parser identifies the methods present in each class of the class diagram. The method computeArea() is overridden in all child classes. The signatures of the class diagram method and sequence diagram methods are compared and no inconsistency is detected. A further validation of guard conditions and preconditions identifies three inconsistent methods due to wrong guard conditions. The SRPSO algorithm resolves the inconsistencies in 8 iterations and the sequence diagram specification has consistent method invocations with guard conditions matching the preconditions. The execution time of the algorithm is 875 ms.

The UML model Deposit and Payroll System used for evaluating the algorithm are represented in Figures 7 and 8, respectively. The UML model exhibits dynamic polymorphism, whereas the UML model Payroll system exhibits static polymorphism. The UML model Deposit has three inconsistent method invocations. The method invocations are prefixed with the guard condition. The UML model Payroll System has 9 method invocations out of which 5 invocations are inconsistent. The UML model Deposit in Figure 7 forms a part of the banking system to compute the interest of term deposits. The method Interest() is overridden in the derived classes. The interest rate depends on the period of the term deposit. The three method invocations are inconsistent. Inconsistent design results in wrong values for the interest calculated and maturity value. This creates a set of unsatisfied customers and affects the credibility of the banking system. Inconsistent design results in the creation of software with faults. This affects the software quality. The errors may be identified either during the testing phase or after delivery of the product, which increases the software de-

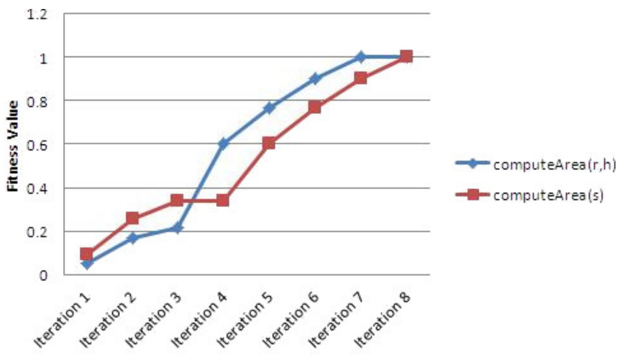


Figure 4. Fitness Values of Inconsistent Methods for UML model 3DObject

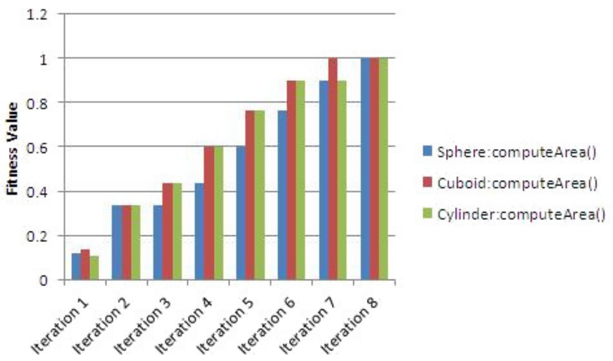
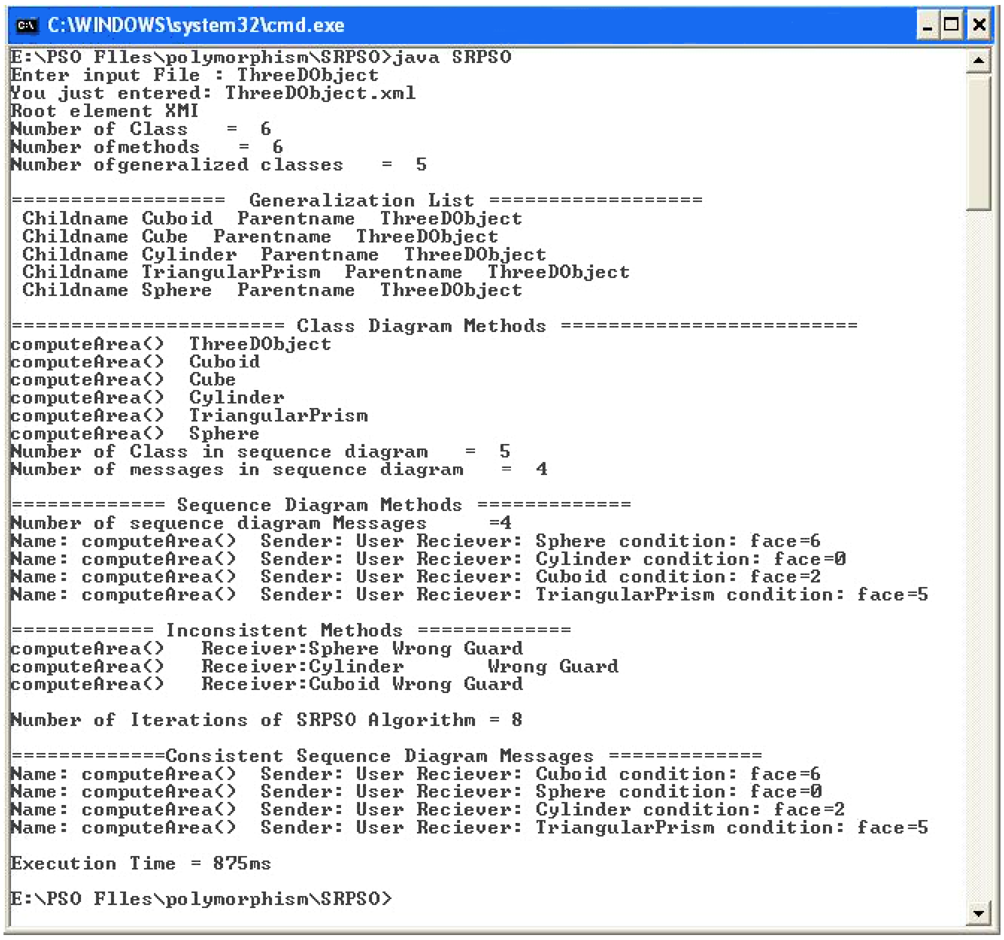


Figure 5. Fitness Values of Inconsistent Methods for UML model ThreeDObject



```
E:\PSO FIles\polymorphism\SRPSO>java SRPSO
Enter input File : ThreeDObject
You just entered: ThreeDObject.xml
Root element XMI
Number of Class    =  6
Number ofmethods    =  6
Number ofgeneralized classes    =  5

==================  Generalization List ==================
 Childname Cuboid  Parentname  ThreeDObject
 Childname Cube  Parentname  ThreeDObject
 Childname Cylinder  Parentname  ThreeDObject
 Childname TriangularPrism  Parentname  ThreeDObject
 Childname Sphere  Parentname  ThreeDObject

======================= Class Diagram Methods ========================
computeArea()  ThreeDObject
computeArea()  Cuboid
computeArea()  Cube
computeArea()  Cylinder
computeArea()  TriangularPrism
computeArea()  Sphere
Number of Class in sequence diagram   =  5
Number of messages in sequence diagram   =  4

============= Sequence Diagram Methods =============
Number of sequence diagram Messages     =4
Name: computeArea()  Sender: User Reciever: Sphere condition: face=6
Name: computeArea()  Sender: User Reciever: Cylinder condition: face=0
Name: computeArea()  Sender: User Reciever: Cuboid condition: face=2
Name: computeArea()  Sender: User Reciever: TriangularPrism condition: face=5

============ Inconsistent Methods =============
computeArea()   Receiver:Sphere Wrong Guard
computeArea()   Receiver:Cylinder        Wrong Guard
computeArea()   Receiver:Cuboid Wrong Guard

Number of Iterations of SRPSO Algorithm = 8

=============Consistent Sequence Diagram Messages =============
Name: computeArea()  Sender: User Reciever: Cuboid condition: face=6
Name: computeArea()  Sender: User Reciever: Sphere condition: face=0
Name: computeArea()  Sender: User Reciever: Cylinder condition: face=2
Name: computeArea()  Sender: User Reciever: TriangularPrism condition: face=5

Execution Time = 875ms

E:\PSO FIles\polymorphism\SRPSO>
```

Figure 6. Handling Inconsistencies for the UML model ThreeDObject

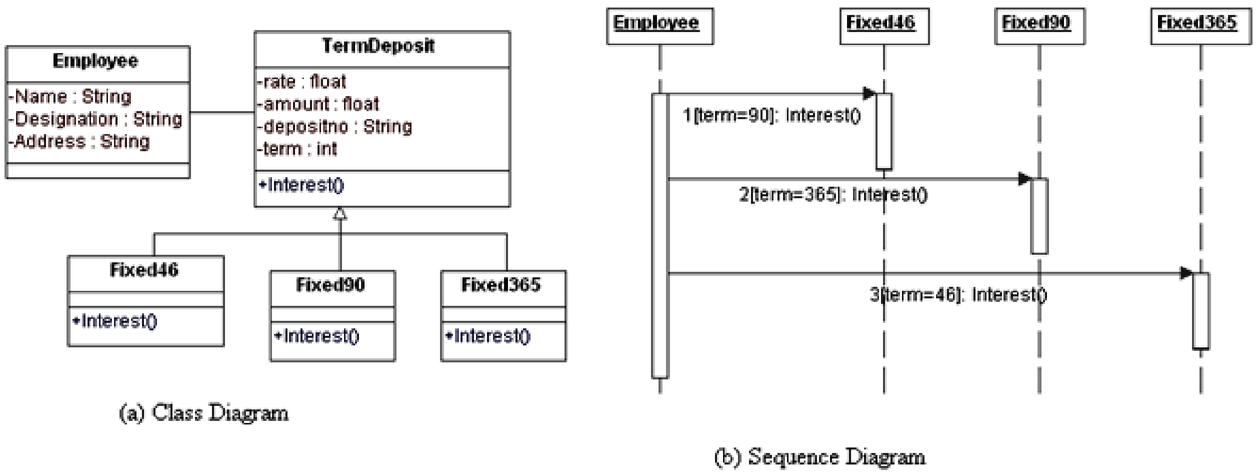
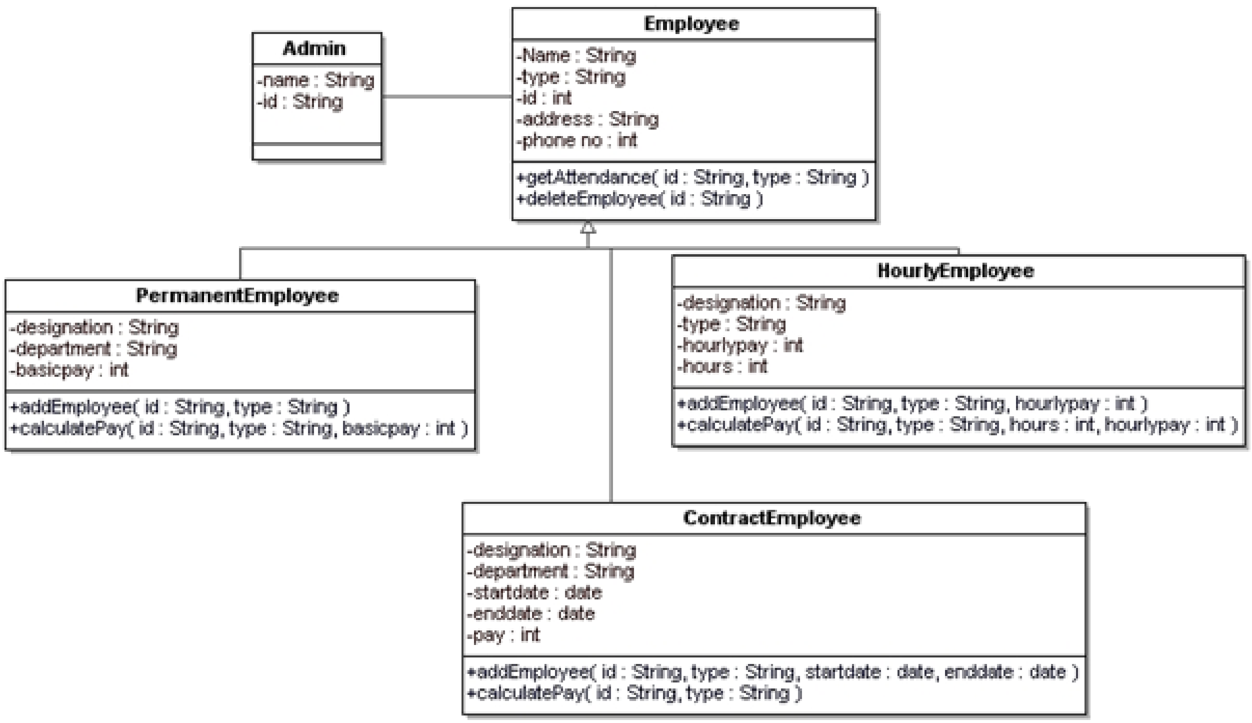


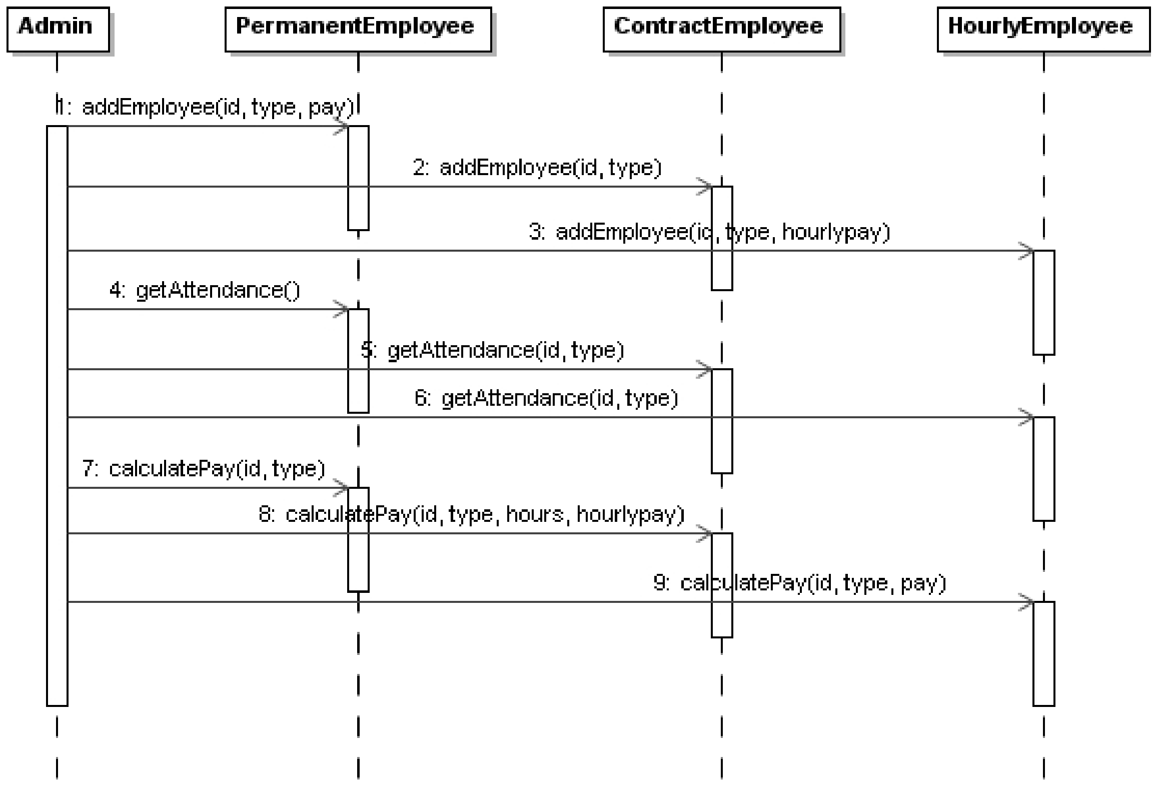
Figure 7. UML Model Deposit

vlopment cost, maintenance cost, and time to market the software.

The algorithm is evaluated based on two criteria: convergence and execution time. The convergence of the algorithm is evaluated based on the number of iterations required to resolve inconsistency. Inconsistency is resolved when the fitness values of all particles in the swarm are equal to one. We have modelled different case studies and the algorithm is experimented on


Admin
-name : String
-id : String
Employee
-Name : String
-type : String
-id : int
-address : String
-phone no : int
+getAttendance(id : String, type : String)
+deleteEmployee(id : String)
PermanentEmployee
-designation : String
-department : String
-basicpay : int
+addEmployee(id : String, type : String)
+calculatePay(id : String, type : String, basicpay : int)
HourlyEmployee
-designation : String
-type : String
-hourlypay : int
-hours : int
+addEmployee(id : String, type : String, hourlypay : int)
+calculatePay(id : String, type : String, hours : int, hourlypay : int)
ContractEmployee
-designation : String
-department : String
-startdate : date
-enddate : date
-pay : int
+addEmployee(id : String, type : String, startdate : date, enddate : date)
+calculatePay(id : String, type : String)

(a) Class Diagram


Admin
PermanentEmployee
ContractEmployee
HourlyEmployee
1: addEmployee(id, type, pay)
2: addEmployee(id, type)
3: addEmployee(id, type, hourlypay)
4: getAttendance()
5: getAttendance(id, type)
6: getAttendance(id, type)
7: calculatePay(id, type)
8: calculatePay(id, type, hours, hourlypay)
9: calculatePay(id, type, pay)

(b) Sequence Diagram

Figure 8. UML Model Payroll System

Table 3. Execution Time and Convergence

UML Model	Polymorphism Type	Methods	Number of Inconsistent Invocations	Iteration Count	Avg. Running Time(ms)
Deposit	Dynamic	3	3	8	764
3DObject	Static	4	2	8	954
ThreeDObject	Dynamic	4	3	8	984
Course Registration System	Dynamic	12	6	7	850
Payroll System	Static	9	5	7	998
Demonstrative Sample	Static	12	7	8	1052

different inconsistent models exhibiting static and dynamic polymorphism. Table 3 represents the execution time and convergence of the algorithm on different UML models. The table represents the UML models, the type and number of inconsistencies present in the models, the number of iterations required to converge, and the average running time of the algorithm. The execution time of the algorithm is computed on an Intel Core i7 CPU running at 2.80 GHz with 4 GB primary memory. The UML model Deposit requires an average running time of 764 ms to achieve consistency; the average running time of UML model 3DObject and ThreeDObject are 954 ms and 984 ms, respectively. The UML model 3DObject exhibits static polymorphism and has 4 method invocations out of which two invocations are inconsistent. Models Deposit and ThreeDObject exhibit dynamic polymorphism. The UML model Demonstrative Sample has polymorphic and non-polymorphic methods. Inconsistencies in non-polymorphic methods are detected from the fitness value computation. The results show that the average time taken by the algorithm to detect and fix inconsistencies in polymorphic methods is of the order of milliseconds and the algorithm converges in all cases.

Table 4 represents the statistical evaluation results of the algorithm. The values of mean, standard deviation and variance are computed for different values of acceleration coefficients. The algorithm is statistically evaluated on the UML model and better results are obtained with acceleration coefficient values equal to 1.49445. The evaluation results have shown that the algorithm detects and fixes all inconsistent method invocations. As a result, no false positives or false negatives are detected. Hence the precision and recall values are high and equal to the one in our approach.

Inconsistency handling has a prime role in the development of quality software. Polymorphism makes the design extensible. It simplifies the design and enables the addition of new functions without creating additional overheads. Inconsistencies arising due to method invocation inconsistency of polymorphic methods are hard to detect. We have presented an AI based approach that detects and fixes inconsistency in polymorphic and non-polymorphic methods. Our approach provides significant role in ensuring software design consistency. The proposed approach of inconsistency detection has a number of advantages. The method operates on a specification of the diagram and uses a direct approach of detecting and fixing inconsistencies without transforming the model to an intermediate representation. The approach detects and fixes method invocation inconsistency in polymorphic and non-polymorphic methods. The fitness function uses simple calculations. Addition of new rules requires only a redefinition of the fitness function. Inconsistencies are fixed by identifying the receiver class from the cohesion values and guard conditions and redefining the method invocations in the sequence diagram. The algorithm is fast and computationally inexpensive. As the inconsistencies are detected and fixed in the design phase, the errors are not propagated to the code generation phase. Hence, the development and maintenance costs are reduced and quality of the code can be improved.

Table 4. Statistical Evaluation of the Algorithm

UML Model	Parameter	$a1 = a2 = 1.49445$		$a1 = a2 = 1$	
		4 Runs	7 Runs	4 Runs	9 Runs
3DObject	Mean	0.4685	0.95	0.2768	1
	SD	0	–5.551E–17	0	0
	Variance	0.017292	0.0025	0.003624	0
ThreeDObject	Mean	0.544444	0.9333333	0.488889	1
	SD	–3.70E–17	–1.110E–16	0	0
	Variance	0.00617	0.0022222	0.000202	0
Deposit	Mean	0.65556	0.95556	0.4888809	1
	SD	0	3.701E–17	–1.850E–17	0
	Variance	0.006173	0.003951	0.00617284	0
Course Registration System	Mean	0.711111	1	0.416667	0.911111
	SD	0	0	9.2519E–18	–3.701E–17
	Variance	0.006173	0	0.001389	0.003951
Payroll System	Mean	0.7	1	0.413333	1
	SD	0	0	–1.110E–17	0
	Variance	0.006667	0	0.0016	0

7. Threats to validity

The section deals with threats to validity.

External Validity concerns with how the result of the experiments can be generalized to other environments. As part of the evaluation, we have evaluated the algorithms on UML models involving polymorphic method invocations. The proposed approach detects and fixes inconsistencies involving static and dynamic polymorphic method invocations. The algorithm can be generalized to detect inconsistencies in non-polymorphic method invocations and handle other inconsistencies involving sequence diagrams. The generalization can be performed by modifying the fitness function. This argument is substantiated by describing how another inconsistency related to the class and sequence diagram is handled. The consistency rule states that two objects in the sequence diagram interact only if there is an association in the class diagram between the interacting objects. The fitness function can be modified to include another term comprising of the sender and receiver classes in the class and sequence diagram. The proposed approach models inconsistency handling as an optimization problem and detecting inconsistencies with fitness function. The algorithm can be expanded to detect and fix intra-model inconsistencies among different diagrams. We have defined the fitness function in terms of the properties of the inconsistent model elements. Inconsistency detection among different diagrams requires definition of the fitness function in terms of the properties of the inconsistent model element and a particle representation has to be formulated for the inconsistent model element in terms of its properties.

Construct Validity refers to the extent to which the experiment setting reflects the theory. We are able to successfully implement the algorithm on a set of UML models involving static and dynamic polymorphic method invocations. The fitness functions are defined with the aim of detecting method invocation inconsistencies and inconsistencies are identified and resolved accurately. The UML models are a representative of the models on which a consistency check can be performed. The number of inconsistencies in the UML models varies from 3 to 7 and the number of method invocations varies from 3 to 12.

Internal Validity represents the extent to which the casual relationship established cannot be explained by other factors. The casual relationships between class diagram method signature and sequence diagram method signature are analyzed to detect inconsistency. Method invocation inconsistency arises due to the invocation of a method on an object of a class in which the method is not defined. Fitness function is defined in terms of the method signature and class names. Hence, the method signature is the major component in inconsistency detection and the casual relationship between method signatures is exploited to detect inconsistencies. In the case of dynamic polymorphism, since the method signatures of the polymorphic methods are the same, a further comparison of guard conditions and constraints is performed.

Conclusion Validity: We have performed a statistical evaluation of the algorithm and the results are summarized in Table 4. The models used for evaluation are a representative of the UML models used in the design of software systems. The statistical evaluation results show that the algorithm converges in less number of iterations with acceleration coefficient values equal to 1.49445. The convergence of the algorithm and execution time are also computed. The average execution time is of the order of milliseconds and the number of iterations required for the algorithm to converge is independent of the number of method invocations or the number of inconsistencies.

8. Conclusion

With the increasing relevance of software in industries and manufacturing, the complexity and size of the software and the complexity of the design has increased. Developing quality software is one of the major challenges faced by software developers. One of the definitions of quality software is fitness for purpose and quality software should be able to function as per the user′s requirements. One of the key aspects to ensuring software quality is good design. Inconsistent design leads to the generation of software with faults. A periodic review of the software design is one the factors that can enhance the software quality and reduce software failures thereby improving manufacturing and productivity. The review helps to detect inconsistencies and fix the inconsistencies. Polymorphism is an important feature that makes the software design compact and extensible. It is hard to trace the polymorphism as it is often detected at run time. We introduce an intelligent automated approach that uses the SRPSO algorithm to detect and fix inconsistency in polymorphic methods. The algorithm is evaluated on different case study involving static and dynamic polymorphism. The method detects and fixes inconsistencies in all cases. Analysis of the results shows that the inconsistency detection and fixing in our approach is quick, easy, and effective. The proposed approach has a number of advantages. The algorithm can be invoked after the application is modelled or during and after refinements to the models. The method operates directly on the diagram specification and does not require transformation to another representation. Addition of new rules requires only a redefinition of the fitness function. The fitness function uses simple calculations. The time required to detect and fix inconsistencies is of the order of milliseconds. The inconsistencies developed in the design are detected and corrected in the same phase. Maintenance cost of software is a huge burden for manufacturing industries. Automatic detection of inconsistencies in polymorphic methods during the design phase ensures quality of the code produced and reduces development and maintenance cost of the software.

References

[1] M. Harman, “The role of artificial intelligence in software engineering,” in *First International Workshop on Realizing AI Synergies in Software Engineering (RAISE)*. IEEE, 2012, pp. 1–6.

[2] M. Harman and B.F. Jones, “Search-based software engineering,” *Information and Software Technology*, Vol. 43, No. 14, 2001, pp. 833–839.

[3] O. Raiha, “A survey on search-based software design,” *Computer Science Review*, Vol. 4, No. 4, 2010, pp. 203–249.

[4] J. Kennedy and R. Eberhart, “Particle swarm optimization,” in *Proceedings of IEEE International Conference on Neural Networks*. IEEE, 1995, pp. 1942–1948.

[5] M.R. Tanweer, S. Suresh, and N. Sundararajan, “Self regulating particle swarm optimization al-

gorithm," *Information Sciences*, Vol. 294, 2015, pp. 182–202.

[6] P. Stevens and R.J. Pooley, *Using UML: Software engineering with objects and components.* Pearson Educationr, 2006.

[7] R. France and B. Rumpe, "Model-driven development of complex software: A research roadmap," in *Future of Software Engineering.* IEEE Computer Society, 2007, pp. 37–54.

[8] B. Selic, "The pragmatics of model-driven development," *IEEE Software*, Vol. 20, No. 5, 2003, pp. 19–25.

[9] C. Pons, L. Olsina, and M. Prieto, "A formal mechanism for assessing polymorphism in object-oriented systems," in *Proceedings of First Asia-Pacific Conference on Quality Software.* IEEE, 2000, pp. 53–62.

[10] A. Egyed, "Fixing inconsistencies in UML design models," in *29th International Conference on Software Engineering (ICSE'07).* IEEE, 2007, pp. 292–301.

[11] A. Egyed, E. Letier, and A. Finkelstein, "Generating and evaluating choices for fixing inconsistencies in UML design models," in *Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering.* IEEE Computer Society, 2008, pp. 99–108.

[12] A. Egyed, "Automatically detecting and tracking inconsistencies in software design models," *IEEE Transactions on Software Engineering*, Vol. 37, No. 2, 2011, pp. 188–204.

[13] Q. Long, Z. Liu, X. Li, and H. Jifeng, "Consistent code generation from UML models," in *Proceedings of the Australian Software Engineering Conference.* IEEE, 2005, pp. 23–30.

[14] L.C. Briand, Y. Labiche, and L. O'Sullivan, "Impact analysis and change management of UML models," in *Proceedings of the International Conference on Software Maintenance, ICSM 2003.* IEEE, 2003, pp. 256–265.

[15] R. Kretschmer, D.E. Khelladi, A. Demuth, R.E. Lopez-Herrejon, and A. Egyed, "From abstract to concrete repairs of model inconsistencies: An automated approach," in *24th Asia-Pacific Software Engineering Conference (APSEC).* IEEE, 2017, pp. 456–465.

[16] A. Rountev, A. Milanova, and B.G. Ryder, "Fragment class analysis for testing of polymorphism in Java software," *IEEE Transactions on Software Engineering*, Vol. 30, No. 6, 2004, pp. 372–387.

[17] D.K. Saini, "Testing polymorphism in object oriented systems for improving software quality," *ACM SIGSOFT Software Engineering Notes*, Vol. 34, No. 4, 2009, pp. 1–5.

[18] R. Kaur and J. Sengupta, "Software process models and analysis on failure of software development projects," *arXiv preprint arXiv:1306.1068*, 2013.

[19] K.A. Briski, P. Chitale, V. Hamilton, A. Pratt, B. Starr, J. Veroulis, and B. Villard, "Minimizing code defects to improve software quality and lower development costs," *Development Solutions White Paper, IBM*, 2008.

[20] P. Jalote, "An integrated approach to software engineering," *Springer Science and Business Media*, 2012.

[21] B. Nuseibeh, S. Easterbrook, and A. Russo, "Making inconsistency respectable in software development," *Journal of Systems and Software*, Vol. 58, No. 2, 2001, pp. 171–180.

[22] A. Reder and A. Egyed, "Incremental consistency checking for complex design rules and larger model changes," in *International Conference on Model Driven Engineering Languages and Systems.* Springer, Berlin, Heidelberg, 2012, pp. 202–218.

[23] M.R. Ehrenleitner, A. Demut, and A. Egyed, "Towards model-and-code consistency checking," in *38th Annual Computer Software and Applications Conference.* IEEE, 2014, pp. 85–90.

[24] N. Macedo, T. Jorge, and A. Cunha, "A feature-based classification of model repair approaches," *IEEE Transactions on Software Engineering*, Vol. 43, No. 7, 2017, pp. 615–640.

[25] J.P. Puissant, R.V.D. Straeten, and T. Mens, "A regression planner to resolve design model inconsistencies," in *European Conference on Modelling Foundations and Applications.* Springer, Berlin, Heidelberg, 2012, pp. 146–161.

[26] F.J. Lucas, F. Molina, and A. Toval, "A systematic review of UML model consistency management," *Information and Software technology*, Vol. 51, No. 12, 2009, pp. 1631–1645.

[27] D. Kalibatiene, O. Vasilecas, and R. Dubauskaite, "Rule based approach for ensuring consistency in different UML models," in *EuroSymposium on Systems Analysis and Design.* Springer, Berlin, Heidelberg, 2013, pp. 1–16.

[28] C.F. Borbaand and A.E.A. Da Silva, "Knowledge-based system for the maintenance registration and consistency among UML diagrams," in *Brazilian Symposium on Artificial Intelligence.* Springer, Berlin, Heidelberg, 2010, pp. 51–61.

[29] D. Torre, Y. Labiche, and M. Genero, "ML consistency rules: A systematic mapping study," in *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering.* Springer, Berlin, Heidelberg, 2014, pp. 1–10.

[30] R.J. Ma, N.Y. Yu, and J.Y. Hu, "Application of particle swarm optimization algorithm in the heating system planning problem," *The Scientific World Journal*, 2013, pp. 1–11.

[31] S. Pandey, L. Wu, S.M. Guru, and R. Buyya, "A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments," in *24th International Conference on Advanced Information Networking and Applications*. IEEE, 2010, pp. 400–407.

[32] D.Y. Sha and H.H. Lin, "A multi-objective PSO for job-shop scheduling problems," *Expert Systems with Applications*, Vol. 37, No. 2, 2010, pp. 1065–1070.

[33] M. Gong, Q. Cai, X. Chen, and L. Ma, "Complex network clustering by multiobjective discrete particle swarm optimization based on decomposition," *IEEE Transactions on Evolutionary Computation*, Vol. 18, No. 1, 2014, pp. 82–97.

[34] N.K. Sharma, D.S. Babu, and S.C. Choube, "Application of particle swarm optimization technique for reactive power optimization," in *International Conference on Advances in Engineering, Science and Management (ICAESM-2012)*. IEEE, 2012, pp. 88–93.

[35] P. Sivakumar, S.S. Grace, and R.A. Azeezur, "Investigations on the impacts of uncertain wind power dispersion on power system stability and enhancement through PSO technique," in *International Conference on Energy Efficient Technologies for Sustainability*. IEEE, 2013, pp. 1370–1375.

[36] F. Li, D. Li, C. Wang, and Z. Wang, "Network signal processing and intrusion detection by a hybrid model of LSSVM and PSO," in *15th IEEE International Conference on Communication Technology*. IEEE, 2013, pp. 11–14.

[37] Z. Jun and Z. Kanyu, "A particle swarm optimization approach for optimal design of PID controller for temperature control in HVAC," in *Third International Conference on Measuring Technology and Mechatronics Automation*. IEEE, 2011, pp. 230–233.

[38] D.K. Saini and Y. Sharma, "Soft computing particle swarm optimization based approach for class responsibility assignment problem," *International Journal of Computer Applications*, Vol. 40, No. 12, 2012, pp. 19–24.

[39] R. George and P. Samuel, "Fixing class design inconsistencies using self regulating particle swarm optimization," *Information and Software Technology*, Vol. 99, 2018, pp. 81–92.

[40] R. George and P. Samuel, "Particle swarm optimization method based consistency checking in UML class and activity diagrams," in *Innovations in Bio-Inspired Computing and Applications*. Springer, Cham, 2016, pp. 117–127.

[41] R. George and P. Samuel, "Fixing state change inconsistency with self regulating particle swarm optimization," *Soft Computing*, Vol. 24, No. 24, 2020, pp. 18 937–18 952.

[42] M. Kessentini, H. Sahraoui, and M. Boukadoua, "Search-based model transformation by example," *Software and Systems Modeling*, Vol. 11, No. 2, 2012, pp. 209–226.

[43] A. Windisch, S. Wappler, and J. Wegener, "Applying particle swarm optimization to software testing," in *Proceedings of the 9th annual conference on Genetic and evolutionary computation*. ACM, 2007, pp. 1121–1128.

[44] R. Malhotra and A. Negi, "Reliability modeling using particle swarm optimization," *International Journal of System Assurance Engineering and Management*, Vol. 4, No. 3, 2013, pp. 275–283.

[45] J.A. Parejo, A. Ruiz-Cortes, S. Lozano, and P. Fernandezi, "Metaheuristic optimization frameworks: A survey and benchmarking," *Soft Computing*, Vol. 16, No. 3, 2012, pp. 527–561.

[46] S. Milton and H. Schmidt, "Dynamic dispatch in object-oriented languages," The Australian National University, Canberra, Technical Report TR-CS-94-02, January 1994.

[47] E. Ernst and D.H. Lorenz, "Aspects and polymorphism in AspectJ," in *Proceedings of the 2nd International Conference on Aspect-Oriented Software Development*. ACM, 2003, pp. 150–157.

[48] D. Ancona, S. Drossopoulou, and E. Zucc, "Overloading and inheritance," in *FOOL' 01 – International. Workshop on Foundations of Object Oriented Languages*, 2001.

[49] D.P. Friedman, M. Wand, and C.T. Haynes, *Essentials of Programming Languages*, 2nd ed., Prentice-Hall of India, 2001.

[50] R.V. Binder, "Testing object oriented software: A survey," *Software Testing, Verification and Reliability*, Vol. 6, No. 3–4, 1996, pp. 125–252.

[51] D.P. Rini, S.M. Shamsuddin, and S.S. Yuhaniz, "Particle swarm optimization: technique, system and challenges," *International Journal of Computer Applications*, Vol. 14, No. 1, 2011, pp. 19–26.

[52] Q. Bai, "Analysis of particle swarm optimization algorithm," *Computer and Information Science*, Vol. 3, No. 1, 2010, pp. 180–184.

[53] M. Ferreira, F. Chicano, E. Alba, and J.A. Gomez-Pulido, "Detecting protocol errors

using particle swarm optimization with Java pathfinder," in *Proceedings of the High Performance Computing and Simulation Conference (HPCS 08)*, 2008, pp. 319–325.

[54] D. Floreano and C. Mattiuss, "Bio-inspired artificial intelligence: theories, methods, and technologies," MIT Press, Aug 2008.

e-Informatica Software Engineering Journal, Volume 15, Issue 1, 2021, pages: 29–45, DOI 10.37190/e-Inf210102

A Deep-Learning-Based Bug Priority Prediction Using RNN-LSTM Neural Networks

Hani Bani-Salameh[*], Mohammed Sallam[*], Bashar Al shboul[*]
[*]*Department of Software Engineering, The Hashemite University, Jordan*
hani@hu.edu.jo, mohammed.yasser@intix.net, bashar.alshboul@hu.edu.jo

Abstract

Context: Predicting the priority of bug reports is an important activity in software maintenance. Bug priority refers to the order in which a bug or defect should be resolved. A huge number of bug reports are submitted every day. Manual filtering of bug reports and assigning priority to each report is a heavy process, which requires time, resources, and expertise. In many cases mistakes happen when priority is assigned manually, which prevents the developers from finishing their tasks, fixing bugs, and improve the quality.
Objective: Bugs are widespread and there is a noticeable increase in the number of bug reports that are submitted by the users and teams' members with the presence of limited resources, which raises the fact that there is a need for a model that focuses on detecting the priority of bug reports, and allows developers to find the highest priority bug reports.
This paper presents a model that focuses on predicting and assigning a priority level (high or low) for each bug report.
Method: This model considers a set of factors (indicators) such as component name, summary, assignee, and reporter that possibly affect the priority level of a bug report. The factors are extracted as features from a dataset built using bug reports that are taken from closed-source projects stored in the JIRA bug tracking system, which are used then to train and test the framework. Also, this work presents a tool that helps developers to assign a priority level for the bug report automatically and based on the LSTM's model prediction.
Results: Our experiments consisted of applying a 5-layer deep learning RNN-LSTM neural network and comparing the results with Support Vector Machine (SVM) and K-nearest neighbors (KNN) to predict the priority of bug reports.
The performance of the proposed RNN-LSTM model has been analyzed over the JIRA dataset with more than 2000 bug reports. The proposed model has been found 90% accurate in comparison with KNN (74%) and SVM (87%). On average, RNN-LSTM improves the F-measure by 3% compared to SVM and 15.2% compared to KNN.
Conclusion: It concluded that LSTM predicts and assigns the priority of the bug more accurately and effectively than the other ML algorithms (KNN and SVM). LSTM significantly improves the average F-measure in comparison to the other classifiers. The study showed that LSTM reported the best performance results based on all performance measures (Accuracy = 0.908, AUC = 0.95, F-measure = 0.892).

Keywords: Assigning, Priority, Bug Tracking Systems, Bug Priority, Bug Severity, Closed-Source, Data Mining, Machine Learning (ML), Deep Learning, RNN-LSTM, SVM, KNN

Submitted: 03 July 2020; Revised: 07 November 2020; Accepted: 23 December 2020; Available online: 08 February 2021

1. Introduction

Software projects (both open and closed source) get an overwhelming number of bug reports, and the presence of bugs usually affects reliability, quality, and cost management of software. In practice, it is impossible to have a bug-free software unless the software is implemented carefully and developers can quantify software behaviors as being a bug or not [1].

Bugs are prevalent, and many software projects are delivered with bugs. To address these bugs and to improve the quality of the released products, bug tracking systems such as JIRA and Bugzilla allow users and team members to report bugs [2].

Bug tracking systems help to predict the progress of a milestone-based on bug reports raised. They allow users to add stories (functional requirements) and divide them into tasks, as well as preparing bug reports and test suites [2].

Developers and testers can create new bug reports, monitor the state of bug reports as well as any update on existing bug reports. Bug reports progress through a series of states, where the bug reports begin when the bug is found and ends when the bug reports are closed [1].

Bug reports may then be used to direct the software corrective maintenance behavior and contribute to creating more stable software systems. Prioritizing software bug reports can help to handle the bug triaging process, and allows developers to prioritize and fix important reports first [2]. Developers are often receiving numerous bugs reports and may fail to fix it due to different constraints including time. The process of prioritizing bug reports is manual and is time-consuming. Thus, there is a need to develop a bug's priority prediction model that helps to automate the priority's prediction process.

The model helps to (1) improve accuracy and effectiveness in predicting the priority of the bug reports, (2) improve efficiency by reducing the time spent during manual priority prediction, and (3) reduce the cost of assigning incorrect priority.

This article proposes a framework that used a dataset extracted from five closed-source projects containing more than 2000 bug reports (provided by JIRA). Also, it uses different algorithms, namely RNN-LSTM, SVM, and KKN, to predict the priority and compare the accuracy results.

The rest of this paper is organized as follows. The rest of Section 1 provides background about the bug reports lifecycle and machine learning (ML) and its relationship with software bug problems. Section 2 presents the related works. Section 3 presents the detailed description of the proposed approach. The results of the study are presented in Section 4. Section 5 discusses the priority prediction tool. The possible threats to the validity of our work were listed in Section 6. Finally, Section 7 presents the conclusion of the research work along with future work directions and enhancement.

1.1. Bug reports lifecycle

Bug reports go through a cycle during their lifetime. This article divides the life cycle of the bug reports into five states: *Open*, *InProgress*, *Resolved*, *Closed*, and *Reopened.* These phases are described hereunder (see Fig. 1).

When a tester posts a bug, a bug report is **opened** and logged in to the tracking system, the status is set to OPEN. After that, the leader approves that the bug exists and assigns the bug to the appropriate developer. Once the developer starts analysis and works on fixing the bug, the status set to **INPROGRESS**.

If a bug is posted twice, the status is set to **CLOSED** with a resolution *duplicate.* If the developer feels that the bug is not logical or incompatible with the specific user experience, the status is set to **CLOSED** with resolution *will not do.* If the bug is not reproducible (all attempts to reproduce this bug have failed, or insufficient information was available to reproduce the bug), then the status is set to **CLOSED** with resolution *cannot reproduce.*

Once the developer fixes the issue and verifies the changes, the status is set to **RESOLVED**. After fixing the bug, testing is pending and the tester either confirms the change or re-test the changes to make sure that the bug is no longer exists in the software. Next, the status is set to **CLOSED** with the decision done. If the bug still exists and not resolved, the status is set to **REOPEN**.

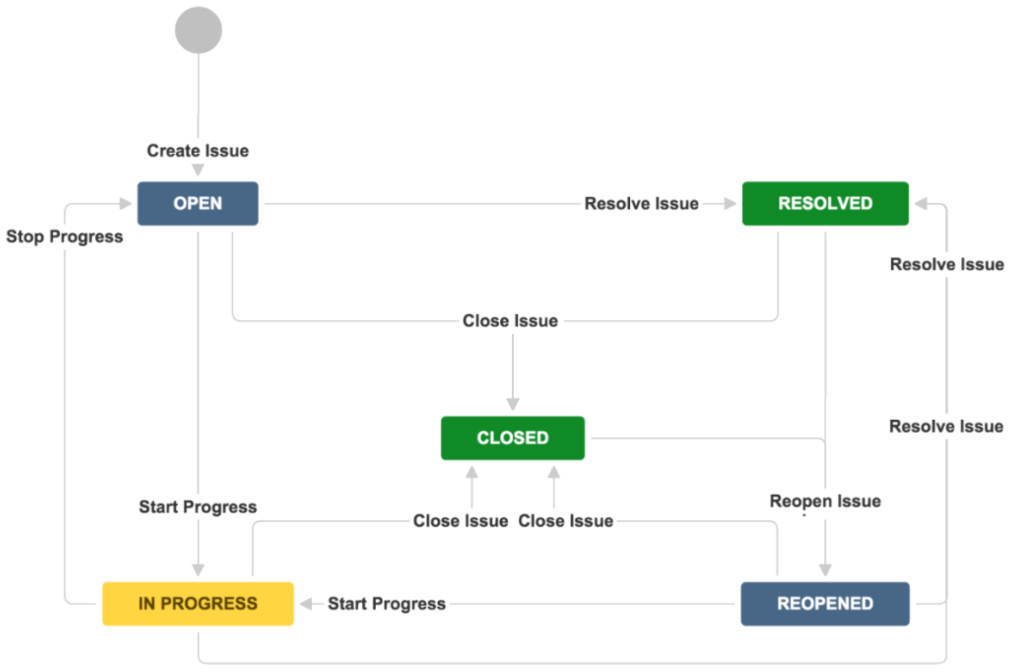


Figure 1: Lifecycle of bug reports

Finally, when the software delivery date (deadline) reaches and low priority bugs are not fixed, they must be moved to the next release by the product owner, and the status remains **OPENED**.

1.2. Bug reports contents

A bug report contains information on how the bug could be reproduced and the information that can help in its debugging and tracing. A bug report includes a set of factors like *summary*, *descriptions*, *report id*, *project name*, *priority*, *environment*, *attachment*, *assignee*, *reporter*, *created date*, *status*, *fix version*, and *component*. Table 1 shows the defined factors.

2. Related works

This article focuses on related works and studies that are mainly related to machine learning (ML)

Table 1: Summary of bug report fields

Field	Description
Summary	A brief one-line summary of the bug.
Descriptions	Details including test step, actual result, and expected result to reproduce this bug
Report ID	A unique identifier for this bug
Project Name	The parent project to which the bug belongs.
Priority	How quickly a bug should be fixed and deployed (e.g., Low, Medium, and High)
Environment	The environment in which the issue occurred (e.g., production, pre-production, staging, and development.)
Attachment	Documents, screenshots, and other elements that can help in identifying and fixing bugs.
Assignee	A person who created the bug (e.g., QA).
Reporter	A person who is responsible for fixing the bug (e.g., QA, scrum master, and owner).
Created Date	Date when a bug is submitted.
Status	The stage the bug is currently in during the lifecycle (workflow).
Fix Version	Project version(s) that contains the bug.
Component	Component(s) to which the bug relates (e.g., Android, IOS, and Backend (DB)).

and the techniques were applied to assigning bugs priority and prediction. This section introduces recent studies and literature that are related to bugs' priority.

2.1. Bug reports assignment

Anvik et al. [3] introduced a machine learning method that classifies appropriate developer names to resolve the report based on classifying and reporting bug using accuracy and recall. Applying their method on Firefox and Eclipse, they achieved +50% accuracy.

Wang et al. [4] address three limitations of the supervised bug fix approaches and propose an unsupervised method for assigning bugs for developers based on their involvement ("activeness score") in the project. The result of experiments showed that FixerCache gives better accuracy when compared with the supervised approaches, and it achieves prediction accuracy up to 96.32% and diversity up to 91.67% in Eclipse and Mozilla datasets.

Recently, Mani et al. [5] proposed an algorithm using a deep-bidirectional recurrent neural network (DBRNN-A) model. The model is for a specific software bug reports classifying an adequate developer depending on the title and characteristics of the bug reports using naive Bayes, cosine distance, SVM, and softmax. Experiments on bug reports from software projects (e.g., Google Chromium, Mozilla Core, and Mozilla Firefox). It showed a precision of 47% for Google Chromium, 43% precision for Mozilla Core, and 56% precision for Mozilla Firefox.

2.2. Bug priority prediction

Prioritizing bug reports is not an easy task. Only a small percentage of bug reports are extremely impactive reports (e.g., according to Ohira et al. [6] less than 1% of Ambari bug reports are absent in the dataset).

2.2.1. Traditional approaches to bug priority prediction

Tian et al. [2, 7] suggested an automated classification method to predict the priority of bug reports. They used a machine learning (ML) algorithm to prioritize bug reports and achieved an average F-measure of 209%. The dataset of the bug reports is usually unbalanced according to the low number of high impact bugs in the project.

Umer et al. [8] proposed an emotion-based automated priority prediction approach. Their approach combines the NLP techniques and ML algorithms. It allows team members to assign appropriate priority level bug reports in an automated manner. The results suggest that the proposed approach outperforms the state-of-the-are and it improves $F1$-score by more than 6%.

Mihaylov [9] conducted a study that aims to examine the behavior of NNs and predict the priority of bug reports. Their focus was to analyze the importance of adding numerical features to textual features by combining different kinds of NNs. The results suggest that adding numerical features to textual features improves the accuracy of priority classification. The results show that the priority classification improves the accuracy of about 85.5%.

Choudhary et al. [10] introduce an ANN technique used to develop prediction models for several Eclipse versions that set priority levels based on the textual, temporal, relevant report, author, severity, and the product.

Yu et al. [11] proposed an enhanced ANN--based system to predict the priorities of five different product bugs identified by an international health-care company. The threefold cross-validation tests suggest that the alternative approach is better in terms of precision, recall, and F-measure.

Jaweria Kanwal [12] proposed an ML-based recommender to automatically prioritize reported bugs. They used SVM to train a classification--based approach on Eclipse bug reports. The evaluation of the proposed approach used precision, retrieval, and F-measure to set the priority of the automatic defect.

Lin et al. [19] applies both of SVM and C4.5 classifiers on different fields (e.g., bug type, submitter, phase-ID, module-ID, and priority). They used a dataset with 2,576 bug reports. Their models achieve the accuracy of up to 77.64%.

Sharma et al. [13] proposed a priority prediction approach using SVM, NB, KNN, and neural networks. The proposed approach allows to predict the priority of the bug reports. The results showed that the accuracy of the used machine learning techniques in predicting the priority of bugs' reports within the project is found above 70% except NB technique.

Alenezi and Banitaan [14] proposed an approach to predict bugs' priority prediction. They used different ML techniques NB, Decision Tree, and Random Forests. The results show that the proposed approach is feasible in predicting the priority of bug reports. Also, the study shows that Random Forests and Decision Trees beat NB.

Others [15] They proposed an approach that constructs multiple decision trees based on existing datasets and features, which selects the best decision trees to measure the new bugs' priority and severity. They proposed the applicability of ran-

Table 2: Summary of machine learning based bug priority approaches available in literature

Paper(s)	Performance	Features Used	Classifier(s)
		Traditional Bug Priority Prediction	
[7]	improves the average of F-measure by a relative improvement of 58.61%	temporal, textual, author, related-report, severity, product	Drone, SVM, NBM,
[8]	improves F-score by more than 6%	summary	NLP + ML algorithm
[9]	accuracy = 85.5%	sentiment and textual analysis	MLP, CNN, LSTM
[10]	both algorithms are efficient	temporal, textual, severity, product, component	MLP, NB
[11]	suggested improvement in terms of precision, recall, and F-measure	milestone, category, module, main workflow, function, integration, frequency, severity, and tester	Rnhanced ANN, Bayes
[12]	SVM is better	categorical, summary, long description	SVM, NB
[13]	above 70% except NB		SVM, NB, KNN
[14]	F-measure values (Random Forest = 0.611, Decision Trees = 0.603, NB = 0.593)	component, operating system, severity	RF, DT, NB
[15]	accuracy = 75%	–	Decision Trees (DT), Random Forest(RF)
		Deep Learning in Bug Priority Prediction	
[5]	improvement = 12–15%, accuracy = 37–43%	title, description	softmax, SVM, MNB, cosine distance based machine
[16]	accuracy = 56–88%	title, component, priority, product	NB, TF-IDF with SVM, fastText, and DeepTriag
[17]	$F1$-measure improved by 14% and AUC by 7%	–	CNN, RNN-LSTM, and DP-ARNN
[18]	$F1$-measure improved by 7.9%	–	CNN, LSTM), Multinomial NB (MNB), RF
Our approach	accuracy = 90.8%	*component*, *summary*, *assignee*, and *reporter of bug reports.*	LSTM, SVM, KNN

dom forest (RF) for bug reports analysis. Results showed that RF yields 75% as an accuracy score.

2.2.2. Deep learning approaches to bug priority prediction

Mani et al. [5] propose a a bug report representation algorithm using deep-bidirectional RNN network model (DBRNN-A). They chose two features as input for the classification (title, description of the issues). They used bug reports from different projects such as Chromium, Mozilla Core, and Mozilla Firefox. The result shows that DBRNN-A achieves an improvement of 12–15% and performance between 37–43% when compared to other classifiers. Lyubinets et. al [16] present a model to label bugs reports using RNNs. The achieved accuracy were 56–88%.

Fan et al. [17] proposed a deep learning-based method called DP-ARNN, to help predict prospective code defects. They used the attention mechanism to capture important features that might help improve the defect prediction performance. They made use of seven open-source projects. Results indicated that DP-ARNN improves the state-of-the-art traditional methods of software defects prediction where $F1$-measure improved by 14% and AUC improved by 7%.

Ramay et al. [18] proposed a deep neural network-based approach for bug reports severity. They evaluated their model on the history-data of bug reports. The results showed that there is an improvement in the F-measure by 7.90%, which indicates that the approach outperforms the state-of-the-art approaches.

The above mentioned closely related works on bug priority are summarized in Table 2.

These studies have focused on using on both of deep learning and traditional classification algorithms such as C4.5, Bayesian, MLP, and Support Vector Machine (SVM). Our work presents an approach to predict and assign bugs' priority level using deep learning. The proposed approach use of LSTM and outperforms the other classifiers where accuracy improved by 90.8%.

3. Proposed approach

Given a dataset of bug reports from closed-source software projects, this study uses RNN-LSTM neural networks to detect and prioritizes bug reports. The process of assigning the priority level for bug reports consists of two phases (see Fig. 2). This section briefly explains the process phases.

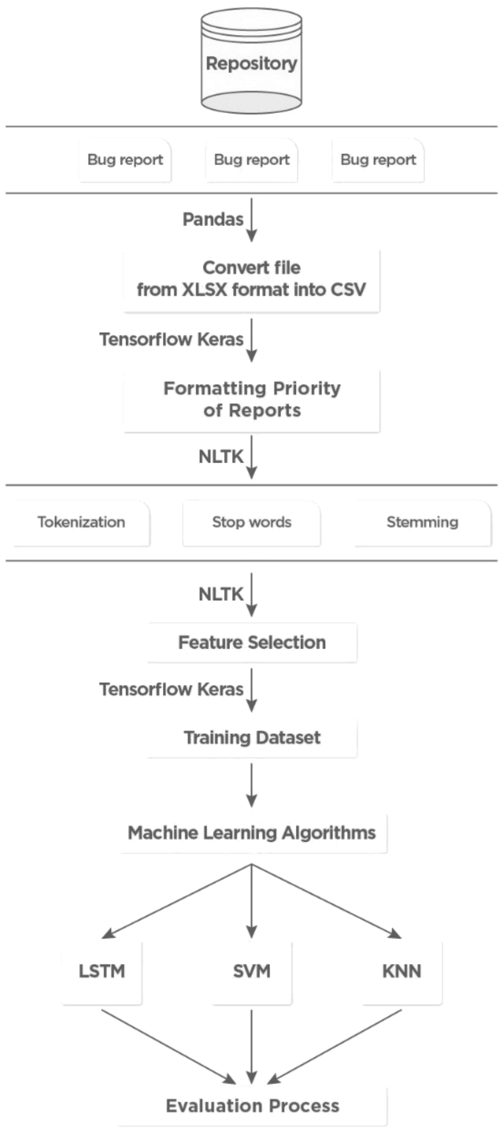


Figure 2: Proposed framework

Phase 1: involves data collection, formatting priority of the bug reports, and text preprocessing (tokenization, stop words, and stemming).

Phase 2: involves feature selection, dataset training, applying ML algorithms (LSTM, SVM, and KNN), and finally evaluation process.

3.1. Data collection

As mentioned earlier, this study used a dataset that was extracted from the JIRA bug tracking system using the INTIX DWC company dashboard [25]. The datasets consist of data from five closed-source projects and containing more than 2000 bug reports. Past studies [3, 4, 12, 26–28] used common datasets extracted from Bugzilla [29] system that are related to Eclipse and Mozilla.

JIRA dataset consists of 17 columns. The factors are summary, description, bug id, status, project name, project lead, priority, resolution, assignee, reporter, created date, resolved date, component, environment, sprint, attachment files, and comments.

In this work, we used a specific number of factors from the chosen dataset. The factors that are considered as the most appropriate to predict the priority level (high, medium, and low): *component*, *summary*, *assignee*, and *reporter of bug reports.*

Table 3 shows the bug reports in the closed--source projects included in the used dataset, which are *Martix*, *Hashfood*, *Tazaj*, *Workspaces*, and *Maharah*. The project with the highest number of bug reports is *Hashfood*. The bug reports are divided into three levels of priority (high, medium, and low). The number of bug reports with medium priority is higher in each dataset compared to the number of low and high priorities.

3.1.1. Labeling priority of reports

The priority of bug reports was labeled using the *to_categorical* function from the TensorFlow Keras library [30]. The used labels are 0 and 1, where 0 refers to the high priority and 1 refers to the low priority of bug reports.

3.1.2. Text preprocessing

Text preprocessing is applied using the Natural Language Toolkit library [31]. This is performed by practicing Python programming using PyCharm [32].

This section gives a brief definition of each activity.

- **Tokenization:** the process of splitting text into sentences, words, and clauses. It replaces all the punctuations with blank spaces, removes all the nonprintable escape characters, and converts all the words to lowercase [7].
- **Stop word removal:** prepositions, articles, conjunctions, verbs, pronouns, nouns, adjec-

Table 3: High, medium, low, and unselect priority levels in each project dataset

Projects name	High	Medium	Low	Unselect
Martix [20]	207	489	135	38
Hashfood [21]	453	659	95	0
Tazaj [22]	80	169	18	0
Workspaces [23]	143	186	9	0
Maharah [24]	55	61	9	0

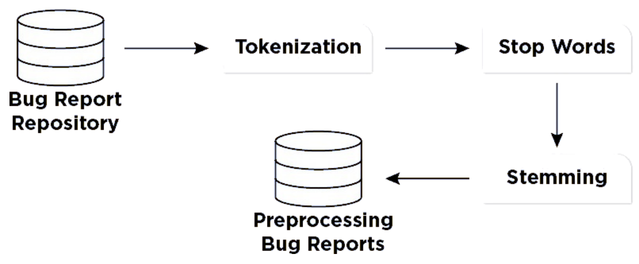


Figure 3: Text preprocessing activities

Table 4: Example illustrates the effect of preprocessing activities

Original Description	Crashed when clicking on order details “Consumer application”
Tokenization	Crash when I click on order details consumer application
Stop Words	Crashed click order details consumer application
Stemming	Crash click order detail consumer application

Table 5: The top-30 keywords based on their frequency (sorted using NLTK)

Rank	Keyword	Rank	Keyword	Rank	Keyword
1	IOS	11	search	21	back
2	Android	12	seller	22	login
3	Screen	13	click	23	logo
4	app	14	App	24	account
5	incorrect	15	Api	25	network
6	message	16	backend	26	payment
7	product	17	chat	27	google
8	user	18	button	28	service
9	order	19	mobile	29	server
10	error	20	design	30	web

Table 6: Keywords classified based on the priority level

Keywords	Count of frequency	Priority level	Keywords	Count of frequency	Priority level
crash	186	high	color	29	low
error	159	high	inconsistent	22	low
icon	110	low	layout	21	low
photo	108	low	avatar	20	low
tab	65	low	placeholder	20	low
image	101	low	doesn’t Work	16	high
menu	53	low	ux	16	low
design	52	low	toolbar	15	low
logo	47	low	textview	9	low
label	45	low	hot fix	9	high
title	34	low	failure	8	high

tives, and adverbs, which has no meaning in NL processing [7].

– **Stemming:** is the process for reducing words to their stem or root. All words with a common stem are replaced. For example, words like “take”, “takes”, “took”, and “taking” can be replaced with a single word as “take” [7].

Table 4 illustrates the preprocessing activities using Natural Language Toolkit (NLTK) [31].

3.1.3. Feature selection

Text preprocessing generates a large set of features that are still costly to be processed using the proposed machine learning algorithms. Thus, it is important to decide what features of the input are relevant.

Various techniques have been proposed to derive relevant features (terms/keywords) from the bug reports. This research used the NLTK library [7] as a features’ selection technique to reduce the number of input features and help improve the performance.

Features are derived from the bug reports to provide the most important words that impact the priority level, then the words are ranked highest to lowest based on the frequency of the word (see Table 5).

Also, manual analysis was applied and identified the strongest set of keywords that refer to the low and high priority levels of the bug reports (see Table 6).

3.2. Evaluation metrics

The performance and effectiveness of the classification algorithms were evaluated using well-known metrics such as *precision*, *accuracy*, *recall*, *F-measure*, and *improvement* [18].

Also, these metrics were used to evaluate the performance of the proposed approach to the priority bug reports. The metrics present the *precision*, *accuracy*, *recall*, and *F-measure* of the proposed approach in assigning priority of the bug reports. Following is a description of the used metrics.

Accuracy is the percentage of correctly predicted observation to the total, which is considered as an important performance measure when using asymmetric datasets that present when false positive and false negatives are the same value [33]. Accuracy can be measured using the following formula:

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{1}$$

Precision is the ratio of correctly predicted positive to the total predicted positive. The percentage of priority bug reports was predicted, and then considered precision for the high and low level of priority [33]. Precision can be measured using the following formula:

$$Precision = \frac{TP}{FP + TP} \tag{2}$$

Recall (Sensitivity) is the ratio of correctly predicted positive to the total observation in the same class. The percentage of all high-priority and low-priority bug reports that are correctly predicted [33]. Recall can be measured using the following formula:

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

F-measure means the average accuracy and recall taking into account false positives and false negatives. F-measure is more effective than accuracy, especially if the data distribution is unbalanced. If false positives and false negatives have the same results, this means that the accuracy is more effective [33]. F-measure can be measured using the following formula:

$$F\text{-measure} = \frac{2 * Precision * Recall}{Precision + Recall} \tag{4}$$

Improvement considers calculating the improvement between selected classification algorithms [34]. Improvement can be measured using the following formula:

$$Improvement =$$

$$\frac{(F\text{-measure}_{\text{LSTM}}) - (F\text{-measure}_{\text{KNN}})}{F-\text{measure}_{\text{KNN}}} \tag{5}$$

Also, to measure the quality of the classifiers, we calculated Mathews Coefficient Correlation (MCC).

$$MCC =$$

$$\frac{(TP*TN)-(FP*FN)}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}} \tag{6}$$

3.3. Machine learning algorithms

The proposed approach is compared with some existing ML algorithms. We used three different machine learning algorithms to predict the bug reports' priority. Theses algorithms are Long Short-Term Memory (LSTM), Support Vector Machine (SVM), and K-nearest neighbors (KNN).

3.3.1. Long short-term memory (LSTM)

RNN-LSTM is an example of supervised learning used in deep learning, which uses history measurements to generate bandwidth prediction and remembers the information for long periods. It can learn to transform input data into a preferred response and is widely used for prediction problems [35]. RNN-LSTM can remember past events that are seen and forget unimportant data. This happens through different activation function layers called gates. The Internal Cell State presents the relevant information that was selected to be saved. RNN-LSTM is a type of a Recurrent Neural Network (RNN) that uses past events to inform future ones [35–38].

In this research, we implemented a Python code for the LSTM neural network with five hidden layers feed-forward to predict the priority of the bug reports. The number of hidden layers has been selected to achieve the best performance. It includes a cell that saves relevant information which has an impact on the priority level. The model helps to assign an appropriate priority level of bug reports [38].

3.3.2. Support vector machine (SVM)

SVM is a supervised machine learning model that applies classification on two-group classification problems. It can be used for classifications, regression, and outliers' detection. The objective of applying SVMs is to classify the dataset space by finding the best line or hyperplane [39–41]. SVM is implemented using a Python code, mainly using Sklearn.svm library [42, 43].

3.3.3. K-nearest neighbors (KNN)

KNN is a supervised machine learning algorithm that can be used to solve both classification and regression problems. Using KNN, the input variables consist of the k closest training examples in the dataset. Predicting the output depends on whether k-NN is used for classification or regression problems [44, 45]. KNN is implemented using a Python code, mainly using Sklearn.neighbours library [42].

3.4. Building the LSTM neural network

As mentioned earlier, Python was used to implement the LSTM neural network with five hidden layers feed-forward to predict the priority of bug reports.

3.4.1. Input variables

The input variables were selected to predict the priority level by considering a set of factors (indicators). The factors are component name, summary, assignee, and reporter of the bug reports (see Table 7).

3.4.2. Output variables

In this study, *Priority* is the output variable used in the ML algorithms to be predicted (see Table 8).

3.5. Supervised training of LSTM

To train the LSTM neural network, we are using a python library called *TensorFlow* [46] that divides the datasets into training and test sets

Table 7: Factors considered from the bug reports

Field name	Field description	Field value
Component	Bug component(s) to which this bug relates.	Android, IOS, Backend (DB).
Summary	A brief one-line summary of the bug.	String(crash, failure, design, UX, or UI).
Assignee	A person who created a bug	Senior, Junior, and a fresh graduate.
Reporter	A person who is responsible to fix the bug.	QA, developer, scrum master, product owner.

Table 8: Output variable

Field name	Field description	Field value
Priority	How quickly the bug should be fixed and deployed?	Low/High

in the ratio 8:2. The datasets were converted from the .xlsx format into .cvs format using the panda's library [47] to make the module faster.

The priority data is converted into [0, 1] using the *to_categorical* function from the TensorFlow Keras library [46]. The assigned values to the priority of bug reports are (0 = high priority, 1 = low priority).

The architecture of LSTM units was trained using Adam algorithms and the Mean Square Error loss function. Adam algorithm has been used in our research instead of the traditional algorithms to update network weights based on training data [48]. The main benefits of *Adam* algorithms are computationally efficient and suitable for handling problems with large data.

The learning rate variable is set to 0.001 and it decays every 5 epochs and drops-out in each layer in 0.2 to remove any loss value in validation split. In this article, the model has been trained with 100 sequences per batch and the count of the batches is 64 with patience from 3 samples.

4. Experimental results and discussion

The performance of the proposed LSTM model was evaluated by the experiment on a dataset extracted from five different projects, which are *Matrix*, *Hashfood*, *Tazaj*, *Workspaces*, and *Maharah*, which have a different number of priority bug reports, as shown earlier in Table 4.

The proposed approach is compared with the existing ML algorithm. Three ML algorithms were utilized on the selected dataset. The evaluation was performed with *LSTM*, *KNN*, and *SVM* after defining that the test size is equal to 0.20 of our dataset, then the bug reports were selected randomly. This is done using the *train_test_split* Python library to split datasets into a random train and test subsets [49]. The model was trained and tested with more than 2000 bug reports.

This rest of this section presents the results of the experiment that was conducted to validate the proposed model and answers the research question.

4.1. Research question

This work investigates the following research question to evaluate the proposed framework:

"Does the proposed approach outperform the other machine learning algorithms in predicting and assigning bug priority? Does the proposed approach improve the accuracy of assigning priority levels of bug reports?"

The research question compares the selected deep neural network (LSTM) against other alternatives as shown in Sections 4.1–4.3. Also, it investigates the performance improvement of the proposed approach as shown in Sections 4.4 and 4.5.

4.2. LSTM neural network

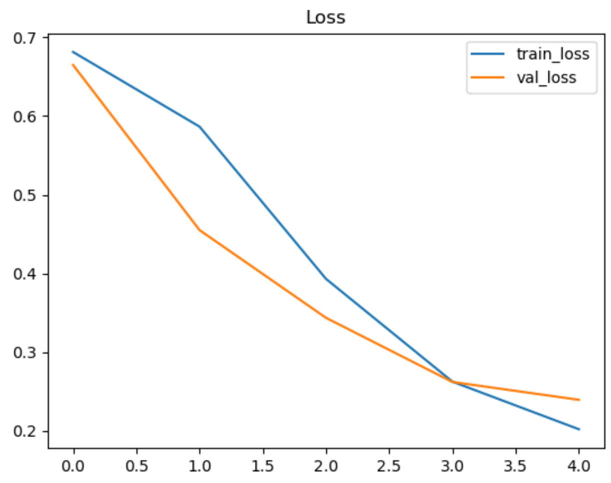
The experiments performed on LSTM Neural Network after training epoch. The LSTM recurrent neural network model was developed in Python using the Keras deep learning library [30].

Accuracy and loss in Keras model for validation data could be changed with different cases. When every epoch increases, the loss becomes lower and the accuracy becomes higher. With Keras validation loss and accuracy, the following cases may occur [50]:

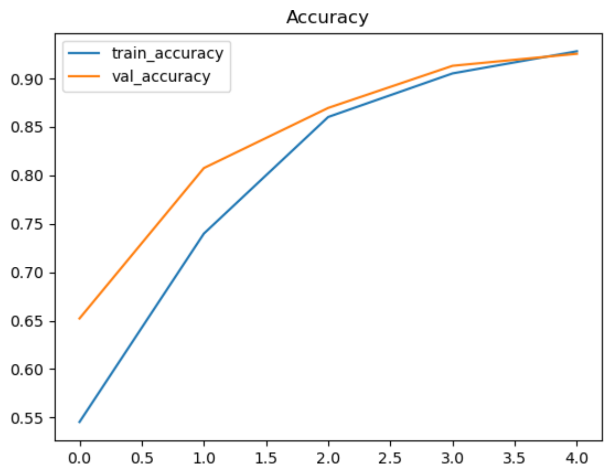
- **The model is not learning (cramming values):** when validation loss starts increasing, validation accuracy starts decreasing.
- **Overfitting:** both of validation loss and validation accuracy start increasing.
- **The model is learning probably:** validation loss starts decreasing, and validation accuracy starts increasing.

As shown in Figures 4(a) and 4(b), this research defined the loss and accuracy functions which are considered as a return to the difference between the training and testing data (predicted and actual outcome). Then we calculated the accuracy, precision, recall, and F-measure.

Figures 4(a) and 4(b) illustrate that the model has higher training accuracy and lower validation accuracy, thus it is learning probably. The training loss is decreasing, which means that the model is learning to recognize the training



(a) Model loss



(b) Model accuracy

Figure 4: Comparison between training and validation(loss and accuracy)

set. Also, the model is a good fit because training loss is slightly higher than validation loss.

4.3. Support vector machine (SVM)

This section presents the results when SVM was applied to datasets extracted from closed-source projects. Table 9 shows the performance results of the SVM model based on the level of priority. Based on the High priority, the metrics values are (F-measure = 0.87, *Recall* = 0.88, and *Precision* = 0.85). Based on the Low priority, the metrics values are (F-measure = 0.86, *Recall* = 0.85, and *Precision* = 0.88).

Table 9: Performance metrics results from applying SVM

Priority level	Precision	Recall	F-measure
High	0.85	0.88	0.87
Low	0.88	0.85	0.86

4.4. *K*-nearest neighbors (KNN)

Based on the performance results of KNN, the metrics values are *Accuracy* = 0.741, F-measure = 0.740, *Recall* = 0.742, and *Precision* = 0.740.

Table 10 shows the evaluation results of the three algorithms. It shows the performance of LSTM, SVM, and KNN.

Table 10: A comparison between performance results from applying LSTM, SVM, and KNN

	LSTM	SVM	KNN
Accuracy	0.898	0.865	0.741
F-measure	0.892	0.865	0.742
Recall	0.897	0.865	0.741
Precision	0.876	0.866	0.743
MCC	0.796	0.732	0.485

Figure 5 illustrates the performance differences between the three ML algorithms.

4.5. Comparison between LSTM, SVM and KNN results

The results of our experiments indicate that the proposed framework based on LSTM Neural Network correctly predicts the priority of the bug reports and the performance can be significantly increased compared with both SVM and KNN as shown in Figure 5.

Based on Table 11 and Figure 5, we make the following observations:

– The proposed approach obtains a slight improvement in performance. The LSTM improvement was calculated and compared with the other selected algorithms SVM and KNN.
– F-measure results show a 3% improvement for LSTM compared with SVM. Also, it shows a 15% improvement for LSTM compared with KNN.

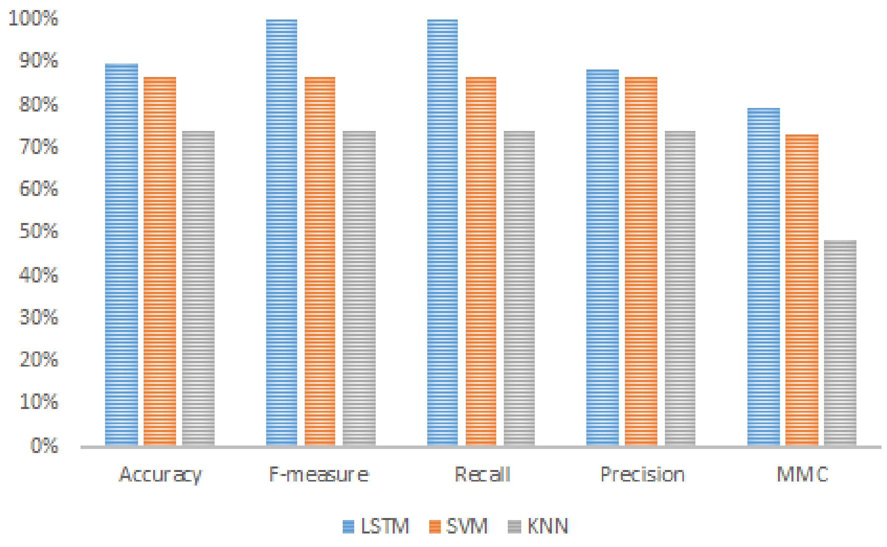


Figure 5: Comparison between LSTM, SVM, and KNN

Table 11: LSTM Improvement compared to SVM and KNN

	SVM	LSTM	Improvement		KNN	LSTM	Improvement
F-measure	0.865	0.892	3%	F-measure	0.742	0.892	15.2%
MCC	0.732	0.796	6.4%	MCC	0.485	0.796	31.1%

– MCC values improved by 6.4% compared to SVM and by 31.1% compared to KNN, which show that LSTM outperforms the other algorithms in detecting and assigning the bugs priority.

4.6. LSTM, SVM, KNN – Output quality comparison

To compare the selected algorithms, this study summarizes and compares the performance of each classifier by calculating the area under the ROC curve (AUC).

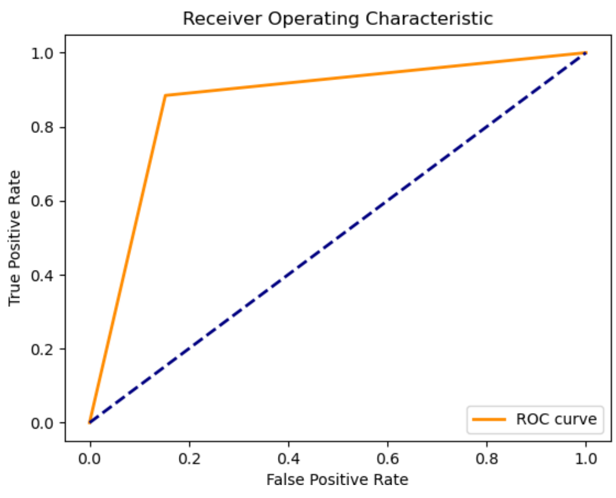


Figure 6: ROC curve for SVM

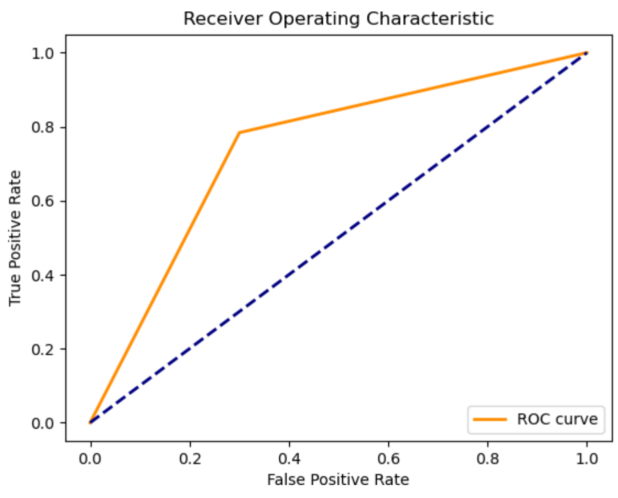


Figure 7: ROC curve for KNN

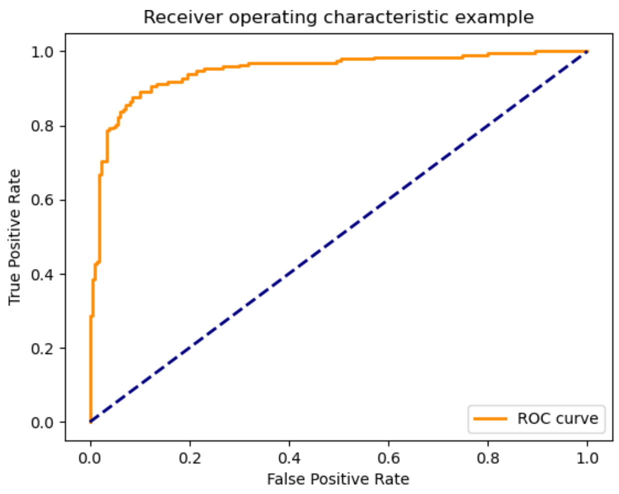
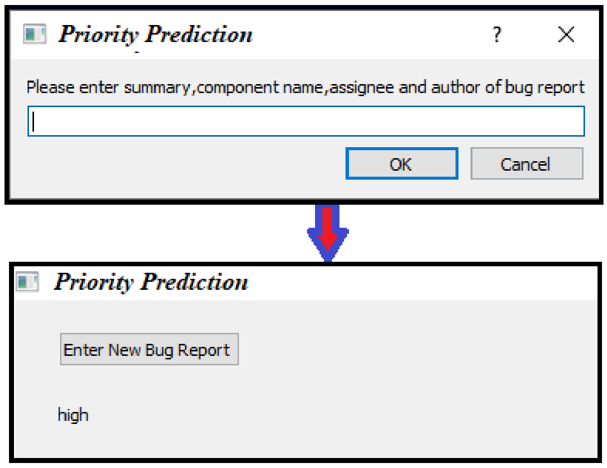


Figure 8: ROC curve for LSTM

Results show that LSTM is with better AUC, which is an effective measure of sensitivity and specificity (a measure of predictive accuracy). The AUC values for LSTM, SVM, and KNN are 0.95, 0.87, and 0.74, respectively (see Figures 6–8).

5. Bugs' priority prediction tool

Assigning priority to bugs' reports may play an important role in improving the bug triaging process which is an important process in software maintenance.



Please enter summary,component name,assignee and author of bug report

high

Figure 9: Predicting priority level for a bug report

This article introduces a tool that helps developers and team to predict and assign priority for bugs' reports. This tool was built using pyqt5.qt widgets [51]. It allows them to enter the input (RNN-LSTM input features) as a single comma separated statement (component name, summary, assignee, and author). Then, the neural network predicts the priority of the report. Figure 9 shows an example of the labeling panel in the proposed tool.

6. Threats to validity

Like any research, some factors may affect the performance of the proposed approach. The threats to the validity of our study are as follows.

The internal validity relates to the adoption of LSTM and not the other algorithms. We chose LSTM since others proved it effective for text classification [52, 53]. Also, the results are verified to avoid any errors.

External validity makes it difficult to generalize the results. As mentioned earlier the used dataset extracted from bug reports related to five closed-source projects. Using datasets from other projects, it is not sure to achieve the same performance results.

7. Conclusion

This research provides a framework for automatically assigning the appropriate priority level for bug reports to avoid time-consuming and limited resources during the software testing process.

The proposed framework involves the use of text pre-processing methods (tokenization, stop words, and stemming) and then extracting important keywords from the description of the bug reports. A dataset was extracted from JIRA using the INTIX DWC company dashboard [4], which consists of five closed-source projects and containing more than 2000 bug reports. The dataset was divided into training and test cases and applied dataset variations (20% test and 80% training).

The proposed model has been validated on a dataset extracted from five real projects. The performance of the model is compared with two well-known ML algorithms, SVM and KNN. The results show that LSTM predicts and assigns the priority of the bug more accurately and effectively. LSTM significantly improves the average

F-measure in comparison to the other classifiers. The study showed that LSTM reported the best performance results based on all performance measures ($Accuracy = 0.908$, $AUC = 0.95$, $F\text{-measure} = 0.892$). This answers our research question, which suggests that LSTM outperforms the alternatives and improves performance.

In the future, we will validate other deep learning approaches on open-source projects like Eclipse and Mozilla. This includes experiments to evaluate the performance of different classifiers such as *Naive Bayes*, *RBF Networks*, and *Functional Trees*. Also, a future work direction might involve integrating the proposed framework with JIRA software.

References

[1] H. Rocha, G. De Oliveira, H. Marques-Neto, and M.T. Valente, "NextBug: a Bugzilla extension for recommending similar bugs," *Journal of Software Engineering Research and Development*, Vol. 3, No. 1, 2015, p. 3.

[2] Y. Tian, D. Lo, X. Xia, and C. Sun, "Automated prediction of bug report priority using multi-factor analysis," *Empirical Software Engineering*, Vol. 20, No. 5, 2015, pp. 1354–1383.

[3] J. Anvik, L. Hiew, and G.C. Murphy, "Who should fix this bug?" in *Proceedings of the 28th international conference on Software engineering*, 2006, pp. 361–370.

[4] S. Wang, W. Zhang, and Q. Wang, "FixerCache: Unsupervised caching active developers for diverse bug triage," in *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 2014, pp. 1–10.

[5] S. Mani, A. Sankaran, and R. Aralikatte, "DeepTriage: Exploring the Effectiveness of Deep Learning for Bug Triaging," *arXiv:1801.01275 [cs]*, Jan. 2018. [Online]. http://arxiv.org/abs/1801.01275

[6] M. Ohira, Y. Kashiwa, Y. Yamatani, H. Yoshiyuki, Y. Maeda, N. Limsettho, K. Fujino, H. Hata, A. Ihara, and K. Matsumoto, "A dataset of high impact bugs: Manually-classified issue reports," in *IEEE/ACM 12th Working Conference on Mining Software Repositories*. IEEE, 2015, pp. 518–521.

[7] Y. Tian, D. Lo, and C. Sun, "DRONE: Predicting Priority of Reported Bugs by Multi-factor Analysis.(2013)," in *29th IEEE International Conference on Software Maintenance (ICSM)*, 2013, pp. 22–28.

[8] Q. Umer, H. Liu, and Y. Sultan, "Emotion based automated priority prediction for bug reports," *IEEE Access*, Vol. 6, 2018, pp. 35 743–35 752.

[9] M. Mihaylov and M. Roper, *Predicting the Resolution Time and Priority of Bug Reports: A Deep Learning Approach*, Ph.D. dissertation, Department of Computer and Information Sciences, University of Strathclyde, 2019. [Online]. https://local.cis.strath.ac.uk/wp/extras/msctheses/papers/strath_cis_publication_2727.pdf

[10] P.A. Choudhary, "Neural network based bug priority prediction model using text classification techniques," *International Journal of Advanced Research in Computer Science*, Vol. 8, No. 5, 2017.

[11] L. Yu, W.T. Tsai, W. Zhao, and F. Wu, "Predicting defect priority based on neural networks," in *International Conference on Advanced Data Mining and Applications*. Springer, 2010, pp. 356–367.

[12] J. Kanwal and O. Maqbool, "Bug prioritization to facilitate bug report triage," *Journal of Computer Science and Technology*, Vol. 27, No. 2, 2012, pp. 397–412.

[13] M. Sharma, P. Bedi, K.K. Chaturvedi, and V.B. Singh, "Predicting the priority of a reported bug using machine learning techniques and cross project validation," in *12th International Conference on Intelligent Systems Design and Applications (ISDA)*. IEEE, 2012, pp. 539–545.

[14] M. Alenezi and S. Banitaan, "Bug reports prioritization: Which features and classifier to use?" in *12th International Conference on Machine Learning and Applications*, Vol. 2. IEEE, 2013, pp. 112–116.

[15] H.M. Tran, S.T. Le, S. Van Nguyen, and P.T. Ho, "An analysis of software bug reports using machine learning techniques," *SN Computer Science*, Vol. 1, No. 1, 2020, p. 4.

[16] V. Lyubinets, T. Boiko, and D. Nicholas, "Automated labeling of bugs and tickets using attention-based mechanisms in recurrent neural networks," in *IEEE Second International Conference on Data Stream Mining and Processing (DSMP)*. IEEE, 2018, pp. 271–275.

[17] G. Fan, X. Diao, H. Yu, K. Yang, and L. Chen, "Software defect prediction via attention-based recurrent neural network," *Scientific Programming*, Vol. 2019, 2019.

[18] W.Y. Ramay, Q. Umer, X.C. Yin, C. Zhu, and I. Illahi, "Deep neural network-based severity prediction of bug reports," *IEEE Access*, Vol. 7, 2019, pp. 46 846–46 857.

[19] Z. Lin, F. Shu, Y. Yang, C. Hu, and Q. Wang, "An empirical study on bug assignment automation using Chinese bug data," in *3rd International Symposium on Empirical Software Engineering and Measurement.* IEEE, 2009, pp. 451–455.

[20] *Martix.* [Online]. https://www.martix.me/ (Last accessed May 17, 2020).

[21] *Hashfood.* [Online]. https://itunes.apple.com/us/app/hashfood/id1117103333?l=ar&ls=1&mt=8 (Last accessed May 17, 2020).

[22] *Tazej.* [Online]. https://itunes.apple.com/jo/app/%D8%B7%D8%A%D8%B2%D8%AC/id1150041871?mt=8 (Last accessed May 17, 2020).

[23] *Workspaces.* [Online]. https://itunes.apple.com/us/app/theworkspacesid1246555146?l=ar&ls=1&mt=8 (Last accessed May 17, 2020).

[24] *Maharah.* [Online]. https://play.google.com/store/apps/details?id=com.mharah.app&hl=ar (Last accessed May 17, 2020).

[25] *INTIX DWC Company.* [Online]. http://intix.net/ (Last accessed May 17, 2020).

[26] S.N. Ahsan, J. Ferzund, and F. Wotawa, "Program file bug fix effort estimation using machine learning methods for open source software Projects," *Institute for Software Technologist Technical*, 2009.

[27] L. Marks, Y. Zou, and A.E. Hassan, "Studying the fix-time for bugs in large open source projects," in *Proceedings of the 7th International Conference on Predictive Models in Software Engineering*, 2011, pp. 1–8.

[28] P. Kaur and C. Singh, "A systematic approach for bug severity classification using machine learning's text mining techniques," *Journal of Computer Science and Information Technology*, Vol. 5, No. 7, 2016.

[29] *Bugzilla.* [Online]. https://www.bugzilla.org/ (Last accessed May 17, 2020).

[30] M. Günel, *Keras: Deep Learning library for Theano and TensorFlow.* [Online]. https://web.cs.hacettepe.edu.tr/~aykut/classes/spring2016/bil722/tutorials/keras.pdf (Last accessed May 17, 2020).

[31] S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit*, 1st ed. Beijing; Cambridge Mass.: O'Reilly Media, Jul. 2009.

[32] *Pycharm, The Python IDE for Professionals.* [Online]. https://www.jetbrains.com/pycharm/ (Last accessed May 17, 2020).

[33] Z. Imran, *Predicting bug severity in open-source software systems using scalable machine learning techniques*, mathesis, Youngstown State University, 2016.

[34] I. Mani and I. Zhang, "kNN approach to unbalanced data distributions: A case study involving information extraction," in *Proceedings of Workshop on Learning from Imbalanced Datasets*, Vol. 126, 2003.

[35] S. Hochreiter and J. Schmidhuber, "Long short--term memory," *Neural Computation*, Vol. 9, No. 8, 1997, pp. 1735–1780.

[36] M.N. Karim and S.L. Rivera, "Comparison of feed-forward and recurrent neural networks for bioprocess state estimation," *Computers and Chemical Engineering*, Vol. 16, 1992, pp. S369–S377.

[37] R. Santos, M. Rupp, S. Bonzi, and A.M. Fileti, "Comparison between multilayer feedforward neural networks and a radial basis function network to detect and locate leaks in pipelines transporting gas," *Chemical Engineering Transactions*, Vol. 32, 2013, pp. 1375–1380.

[38] L. Mei, R. Hu, H. Cao, Y. Liu, Z. Han, F. Li, and J. Li, "Realtime mobile bandwidth prediction using lstm neural network," in *International Conference on Passive and Active Network Measurement.* Springer, 2019, pp. 34–47.

[39] S. Ray, "Suppor vector machine algorithm in machine learning," Sep. 2017. [Online]. https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/

[40] W. Harrad, "A top machine learning algorithm explained: Support vector machines (svms)," Feb. 2020. [Online]. https://www.vebuso.com/2020/02/a-top-machine-learning-algorithm-explained-support-vector-machines-svms/

[41] M. Waseem, "Support vector machine in python," Nov. 2019. [Online]. https://www.edureka.co/blog/support-vector-machine-in-python/

[42] *sklearn.metrics.accuracy_score.* [Online]. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html (Last accessed April 30, 2020).

[43] U. Malik, *Implementing SVM and Kernel SVM with Python's Scikit-Learn.* [Online]. https://stackabuse.com/implementing-svm-and-kernel-svm-with-pythons-scikit-learn/ (Last accessed April 30, 2020).

[44] O. Harrison, *Machine Learning Basics with the K-Nearest Neighbors Algorithm*, 2018. [Online]. https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761 (Last accessed April 30, 2020).

[45] *KNN Algorithm - Finding Nearest Neighbors.* [Online]. https://www.tutorialspoint.com/machine_learning_with_python/machine_learning_with_python_knn_algorithm_finding_nearest_neighbors.htm (Last accessed April 30, 2020).

[46] Google Brain Team, *tensorflow. Develop and train ML models*, 2015. [Online]. https://www.tensorflow.org/ (Last accessed December 15, 2019).

[47] T. Mester, *Pandas Basics (Reading Data Files, DataFrames, Data Selection)*, 2019. [Online]. https://data36.com/pandas-tutorial-1-basics-reading-data-files-dataframes-data-selection/ (Last accessed May 17, 2020).

[48] D.P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[49] *train_test_split.* [Online]. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html (Last accessed May 17, 2020).

[50] J. Brownlee, *How to Diagnose Overfitting and Underfitting of LSTM Models*, 2017. [Online]. https://machinelearningmastery.com/diagnose-overfitting-underfitting-lstm-models/ (Last accessed May 17, 2020).

[51] *PyQt5 Reference Guide.* [Online]. https://www.riverbankcomputing.com/static/Docs/PyQt5/ (Last accessed September 05, 2020).

[52] G. Yang, S. Baek, J.W. Lee, and B. Lee, "Analyzing emotion words to predict severity of software bugs: A case study of open source projects," in *Proceedings of the Symposium on Applied Computing*, 2017, pp. 1280–1287.

[53] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent trends in deep learning based natural language processing," *IEEE Computational Intelligence Magazine*, Vol. 13, No. 3, 2018, pp. 55–75.

e-Informatica Software Engineering Journal, Volume 15, Issue 1, 2021, pages: 47–64, DOI 10.37190/e-Inf210103

Use of Agile Practices in Start-up Companies

Eriks Klotins[*], Michael Unterkalmsteiner[*], Panagiota Chatzipetrou[**], Tony Gorschek[***], Rafael Prikladnicki[****], Nirnaya Tripathi[*****], Leandro Bento Pompermaier[******]

[*] *Software Engineering Research Lab, Blekinge Institute of Technology, Sweden*
[**] *Department of Informatics, CERIS, Örebro University School of Business, Sweden*
[***] *Software Engineering Research Lab, Blekinge Institute of Technology, Sweden*
[****] *Software Engineering Research Lab, Pontifical Catholic University of Rio Grande do Sul, Brasil*
[*****] *Software Engineering Research Lab, University of Oulu, Finland*
[******] *Software Engineering Research Lab, Pontifical Catholic University of Rio Grande do Sul, Brasil*

eriks.klotins@bth.se, michael.unterkalmsteiner@bth.se, panagiota.chatzipetrou@oru.se, tony.gorschek@bth.se, rafael.prikladnicki@pucrs.br, Nirnaya.Tripathi@oulu.fi, leandro.pompermaier@pucrs.br

Abstract

Context Software start-ups have shown their ability to develop and launch innovative software products and services. Small, motivated teams and uncertain project scope makes start-ups good candidates for adopting Agile practices.
Objective We explore how start-ups use Agile practices and what effects can be associated with the use of those practices.
Method We use a case survey to analyze 84 start-up cases and 56 Agile practices. We apply statistical methods to test for statistically significant associations between the use of Agile practices, team, and product factors.
Results Our results suggest that development of the backlog, use of version control, code refactoring, and development of user stories are the most frequently reported practices. We identify 22 associations between the use of Agile practices, team, and product factors. The use of Agile practices is associated with effects on source code and overall product quality. A teams' positive or negative attitude towards best engineering practices is a significant indicator for either adoption or rejection of certain Agile practices. To explore the relationships in our findings, we set forth a number of propositions that can be investigated in future research.
Conclusions We conclude that start-ups use Agile practices, however without following any specific methodology. We identify the opportunity for more fine-grained studies into the adoption and effects of individual Agile practices. Start-up practitioners could benefit from Agile practices in terms of better overall quality, tighter control over team performance, and resource utilization.

Keywords: Agile practices, start-up companies

1. Introduction

Software start-ups are important suppliers of innovation, new products, and services. However, engineering of software in start-ups is a complicated endeavor as the start-up context poses challenges to software engineers [1]. As a result of these challenges, most start-ups do not survive the first few years of operation and cease to exist before delivering any value [2, 3].

Uncertainty, changing goals, limited human resources, extreme time, and resource constraints are reported as characteristics of start-ups [1, 4].

To survive in such a context, start-ups use ad hoc engineering practices and attempt to tailor agile methods to their needs. However,

Submitted: 31 August 2020; Revised: 04 December 2020; Accepted: 12 January 2021; Available online: 29 March 2021

scaled-down agile methods could be irrelevant and ignore start-up specific challenges [5, 6].

Giardino et al. [7] suggest that start-ups adopt practices as a response to some problematic situations and do not consider adopting full agile methodologies, e.g., scrum or XP, at least in early stages.

Pantiuchina et al. [8] make a similar observation and argue that start-ups focus more on speed-related practices, e.g., iterations and frequent releases, than quality-related practices, e.g., unit testing and refactoring.

In this study, we explore the use of Agile practices in start-ups. We focus on identifying the associations between certain Agile practices, product, and team factors. We aim to understand what positive, and potentially adverse effects can be associated with the use of specific practices. We use our results to formulate propositions for further exploration.

We use a case survey to collect data from 84 start-up cases [9]. We use statistical methods to analyze 11,088 data points and identify associations between the use of Agile practices and respondents' estimates on various team and product factors.

We identify 20 statistically significant associations pointing towards potential causes and effects of using Agile practices. We identify that the use of automated tests and continuous integration is associated with positive attitudes towards following best practice. However, the use of planning and control practices are more associated with negative attitudes towards following the best practices.

The rest of this paper is structured as follows. In Section 2, we discuss related work. Section 3 covers the research methodology, data collection, and our approach to data analysis. Section 4 presents the results. We answer our research questions and discuss the implications for research and practice in Section 5. Section 6 concludes the paper.

2. Background and related work

2.1. Software start-ups

Software start-ups are small companies created for developing and bringing an innovative software intensive product or service to the market, and to benefit from economies of scale.

Start-up companies rely on external funding to support their endeavors. In 2015 alone, start--up companies have received investments of 429 billion USD in the US and Europe alone [10, 11]. With an optimistic start-up failure rate of 75% that constitutes of 322 billion USD of capital potentially wasted on building unsuccessful products.

Earlier studies show that product engineering challenges and inadequacies in applied engineering practices could be linked to start-up failures [1, 12]. To what extent software engineering practices are responsible or linked to the success rate is very hard to judge. However, if improved software engineering practices could increase the likelihood of success by only a few percent, it would yield a significant impact on capital return.

Some authors, e.g., Sutton [13] and Giardino [3], point out the unique challenges in start-ups, such as high risk, uncertainty, lack of resources, rapid evolution, immature teams, and time pressure among other factors. At the same time, start-ups are flexible to adopt new engineering practices, and reactive to keep up with emerging technologies and markets [7]. However, our earlier study [12] analyzing the amount of empirical evidence supporting the uniqueness of start-ups found that most start-up characteristics are based on anecdotal evidence. Thus, there could be a negligible difference between start-ups and other organizations launching new software-intensive products on the market in terms of software engineering.

2.2. Agile practices

Agile software engineering practices originate from the Agile manifesto, proposing a shift from heavyweight, plan-driven engineering towards more lightweight, customer-oriented, and flexible methodologies [14]. Agile methodologies, such as Scrum and XP, prescribe specific sets of Agile practices [15, 16]. However, in practice, by-the-book methodologies are often tailored with additional practices to address specific concerns [17, 18]. Thus, we focus our study on what practices start-ups use, without considering any specific agile methodology.

Small organizations have successfully adopted Agile practices for projects where requirements are uncertain and expected to change [19, 20]. In theory, Agile practices could be perfect for software start-ups [6]. However, successful adoption of Agile practices requires highly skilled teams and support throughout the organization [19, 21].

Earlier work on software engineering practices in start-ups suggests that start-ups initially rely on an ad hoc approach to engineering and adopt agile principles incrementally when the need for more systematic practice arises. The shift is often motivated by excessive technical debt, hindering quality, and lack of control over the engineering process [7].

The motivations for adopting agile practices in start-ups include accelerated product delivery, ability to manage changing priorities, and increased team productivity. Practices concerning team collaboration such as open work areas, use of task boards, and a prioritized backlog are reported as the most widely used [22]. Souza et al. [23] reports that start-ups primarily adopt practices that provide immediate benefits and help to accelerate time-to-market.

We explore the associations between 56 Agile practices, product, and team factors. We use a list and descriptions of Agile practices compiled by Agile Alliance, a non-profit community promoting agile principles [24]. To our best knowledge, their website contains the most comprehensive list of Agile practices to date.

In this study, we consider the following practices whose definitions can be found at the Agile Alliance's website [24]: Card, Conversation, Confirmation (3C's), Acceptance tests, Acceptance Test-Driven Development (ATDD), Automated build, Backlog, Backlog grooming, Behavior Driven Development, Burndown chart, Collective ownership, Continuous deployment, Continuous integration, Class Responsibility Collaborator (CRC) Card cards, Daily meeting, Definition of Done, Definition of Ready, Exploratory testing, Facilitation, Frequent releases, Given--When-Then, Heartbeat retrospective, Incremental development, INVEST, Iterations, Iterative development, Kanban board, Lead time, Mock objects, Niko-Niko, Pair Programming, Personas, Planning poker, Point estimates, Project charters, Quick design session, Refactoring, Relative estimation, Role-Feature-Reason, Rules of simplicity, Scrum of Scrums, Sign up for tasks, Simple design, Story mapping, Story splitting, Sustainable Pace, Task board, Team, Team room, Test-driven development, Three Questions, Timebox, Ubiquitous language, Unit tests, Usability testing, User stories, Velocity, and Version control.

In this paper, we follow Agile Alliance naming of the practices. Some of the terms describing practices can also refer to artifacts, e.g., acceptance tests. When we use such a term, we mean the practice of creating and utilizing acceptance tests.

2.3. Effects of using Agile practices

The use of Agile practices is associated with increased product quality and fewer defects compared to plan-driven approaches [25, 26]. We analyze the associations between the use of Agile practices, product documentation, software architecture, quality of the source code, tests, and the overall product quality. In this paper, we adopt the product view on software quality, recognizing the relationship between internal product characteristics and quality of use [27].

Product documentation comprises of written requirements, architecture documentation, and test cases. Deficiencies in such artifacts are associated with hindered knowledge distribution in the team and with adverse effects on further development and maintenance of the product [28]. Note that we analyze if documentation artifacts are understandable and useful without implying any specific format.

Even though the Agile manifesto emphasizes working software over comprehensive documentation, some documentation is essential [14]. For example, user stories are one of the key agile tools to document requirements [29]. System metaphor is useful to communicate the logical structure of the software to all stakeholders [30]. The use of automated testing in continuous integration and deployment pipelines require formally defined tests [31].

Software architecture denotes how different components, modules, and technologies are combined to compose the product. Symptoms such as outdated components, a need for workarounds and patches point towards deficiencies in the software architecture and the lack of attention to refactoring [32, 33].

Source code quality is determined using coding standards and refactoring practices [34, 35]. Degrading architecture and poorly organized source code is associated with increased software complexity, difficult maintenance, and product quality issues down the road [28].

We analyze the quality (or lack thereof) of automated test scripts, removing the need to perform manual regression testing on every release of the product. The effort of manual regression testing grows exponentially with the number of features, slowing down release cycles and making defect detection a time-consuming and tedious task [28].

We also examine the associations between product quality and the use of Agile practices. With product quality, we understand nonfunctional aspects of the product, such as performance, scalability, maintainability, security, robustness, and the ability to capture any defects before the product is released to customers [28].

Good communication, teamwork, adequate skills, and a positive attitude towards the following best practices are recognized as essential team factors for project success [19]. Agile software engineering practices aim to facilitate communication, empower individuals, and improve teamwork [36]. We analyze the associations between team characteristics and the use of specific Agile practices.

Attitudes determine the level of apathy or interest in adopting and following the best engineering practices. Skills characterize to what extent individual members of a start-up team possess relevant engineering skills and knowledge. Communication captures to what extent the team can communicate and coordinate the engineering work. Giardino et al. [7] identify the team as the catalyst for product development in start-ups. Sufficient skills, positive attitudes, and efficient communication are essential for rapid product development in both agile and start-up contexts [7, 19].

Pragmatism characterizes to what extent a team can handle trade-offs between investing in perfected engineering solutions and time-to-market. Agile practices advocate for frequent releases and good-enough solutions [15]. Such practices help to validate the product features early and gather additional feedback from customers [12]. On the other hand, quick product releases need to be accompanied by frequent refactoring and unit tests to manage technical debt and keep regression defects under control [19]. Start-ups often overlook such corrective practices [7, 12].

Sufficient time and resources for product engineering are essential for project success [19]. We analyze what Agile practices can be associated with better resource estimation and planning in start-ups. Several authors, e.g., Giardino et al. [3] and Sutton [13] identify resource shortage as one of the critical challenges in start-ups. However, we, in our earlier study identify the lack of adequate resources, planning and control practices in early start-ups [9].

We look into respondent estimates on the engineering process in their organizations. Process characterizes to what extent product engineering is hindered by unanticipated changes in organizational priorities, goals, and unsystematic changes in the product itself. Lack of organizational support for agile product engineering contributes to project failures [19]. On the other hand, Agile practices offer some room for adjusting to unclear and changing objectives [20].

Agile methods on a high level attempt to address and promise improvements in all these concerns [36]. However, analyzing the effects of applying the whole methodology on a large number of factors does not help to pinpoint specific practices for specific challenges. We aim to establish a fine-grained view on the use and effects of individual practices.

3. Research methodology

3.1. Research aim

We aim to explore how software start-ups use Agile practices and what positive and negative effects can be associated with specific practices.

3.2. Research questions

To guide our study, we define the following research questions (RQ):
RQ1: How are Agile practices used in start-ups?
Rationale: With this question, we identify what Agile practices and in what combinations start--ups use.
RQ2: What are the associations between specific Agile practices and product factors?
Rationale: With this question, we explore the associations between specific Agile practices, quality of documentation, architecture, source code, testing, and overall product quality.
RQ3: What are the associations between specific Agile practices and team factors?
Rationale: With this question, we explore the associations between specific Agile practices, attitudes towards following best engineering practices, pragmatism, communication, skills, resources, engineer process, and teams' productivity.

3.3. Data collection

We used a case survey method to collect primary data from start-up companies [9, 37].

The case survey method is based on a questionnaire and is a compromise between a traditional case study and a regular survey [38]. We have designed the questionnaire to collect practitioners' experiences in specific start-up cases.

During the questionnaire design phase, we conducted multiple internal and external reviews to ensure that all questions are relevant, clear and that we receive meaningful answers. First, the questions were reviewed in multiple rounds by the first three authors of this paper to refine the scope of the survey and question formulations. Then, with the help of other researchers from the Software Start-up Research Network[1], we conducted a workshop to gain external input on the questionnaire. A total of 10 researchers participated and provided their input.

Finally, we piloted the questionnaire with four practitioners from different start-ups. During the pilot, respondents filled in the questionnaire while discussing questions, their answers, and any issues with with the first author of this paper.

As a result of these reviews, we improved the question formulations and removed some irrelevant questions. The finalized questionnaire contains 85 questions in 10 sections. The questionnaire captures 285 variables from each start-up case.

We use a list of 56 Agile practices to capture the respondent's answers on what practices they use in their companies, as described in Section 2.2. The answers are captured in a binary use or not use format. In addition to specific practices, we offer an "I do not know" and "other" option to accommodate for the lack of respondents knowledge and to discover other, unlisted practices. We rely on the respondents best judgment to gauge whether the extent of using a practice in their start-ups qualifies as an application of the practice or not.

We use 45 other questions to capture respondents evaluations of product and team-related statements, such as:
– Initial product/service architecture has become outdated;
– Communication and collaboration within the development team regarding the product/service architecture is insufficient;
– Incremental changes to the product/service are unsystematic and degrades the architecture;
– Quick delivery of functionality is considered more important than good code.

The questions capture the respondents' agreement with a statement characterizing a factor on a Likert scale: not at all (1), a little (2), somewhat (3), very much (4). The values indicate the degree of agreement with a statement. Statements are formulated consistently in a way that

[1]The Software Start-up Research Network, https://softwarestartups.org/

lower values indicate less, and higher values indicate more agreement with the statement.

The questionnaire is designed to be filled in by one person and we analyze one response per start-up. To control this, we collect the contact information of the respondent and the title of their company. In addition to questions about software engineering, the questionnaire contains questions inquiring about the respondents background, and engineering context in the start-up. The full questionnaire is available as supplemental material on-line[2].

The data collection occurred between December 1, 2016, and June 15, 2017. The survey was promoted through personal contacts, by attending industry events, and with posts on social media websites. The survey was promoted as "help us to understand what engineering practices work and does not work in start-ups" and targeted for practitioners with an understanding about the engineering parts of their start-up.

We invited other researchers from the Software Start-up Research Network to collaborate on the data collection. This collaboration helped to spread the survey across many geographical locations in Europe, North and South America, and Asia.

3.4. Data analysis methods

To analyze the survey responses, we used several techniques. We started by screening the data and filtering out duplicate cases, responses with few questions answered, or otherwise unusable responses. In the screening, we attempt to be as inclusive as possible and do not remove any cases based on the provided responses.

Overall, we analyzed responses from 84 start--up cases, 132 data points per each case, and 11,088 data points. We use the Chi-Squared test of association to test if the associations between the examined variables are not due to chance. To prevent Type I errors, we used exact tests, specifically, the Monte-Carlo test of statistical significance based on 10,000 sampled tables and assuming ($p < 0.05$) [39].

To examine the strength of associations, we use Cramer's V test. We interpret the test results as suggested by Cohen [40], see Table 1. To explore the specifics of the association, such as which cases are responsible for this association, we performed post hoc testing using adjusted residuals. We consider an adjusted residual significant if the absolute value is above 1.96 ($\mathit{Adjusted\ residual} > 1.96$), as suggested by Agresti [41].

Table 1. Interpretation of Cramer's V test

Cramer's V value	Interpretation
≥ 0.1	Weak association
≥ 0.3	Moderate association
≥ 0.5	Strong association

The adjusted residuals drive our analysis on how different groups of start-ups estimate aspects of technical debt. However, due to the exploratory nature of our study, we do not state any hypotheses upfront and drive our analysis with research questions.

3.5. Validity threats

In this section, we follow the guidelines by Runeson et al. [42] and discuss four types of validity threats and applied countermeasures in the context of our study.

3.5.1. Construct validity

Construct validity concerns whether operational measures represent the studied subject [42]. A potential threat is that the statements we use to capture respondent estimates are not capturing the indented team and product factors.

To address this threat, we organized a series of workshops with other researchers and potential respondents to ensure that the questions are clear to the point and to capture the studied phenomenon.

We triangulate each factor by capturing it by 3–4 different questions in the questionnaire. To avoid biases stemming from respondents precon-

[2]Full questionnaire: http://eriksklotins.lv/files/GCP_questionnaire.pdf

ceived opinions about the effects of agile practices, we separate questions about the use of practices and questions inquiring about team and product factors.

To accommodate for the fact that a respondent may not know the answers to some of the questions, we provide an explicit "I do not know" answer option to all Likert scale questions.

Regarding the use of agile practices, we ask a binary question capturing the use/not use of a practice. Such an approach does not capture the extent, nor other specifics of the application of a practice. This creates a room for a wide interpretation of what entails using the practice. For example, cases having one automated test and cases with an extensive suite of automated tests would be treated the same.

In this study, we aim for breadth, in terms of studied cases and practices, to understand what agile practices are relevant to start-ups and the team and product factors associated with specific practices. Details of the optimal use of the practices is an avenue for further work.

3.5.2. Internal validity

This type of validity threat addresses causal relationships in the study design [42].

In our study, we do not seek to establish causal relationships, thus this type of validity threat is not relevant.

3.5.3. External validity

This type of validity threat concerns to what extent the results could be valid to start-ups outside the study [42]. The study setting for participants was close to real life as possible. That is, the questionnaire was filled in without researcher intervention and in the participant's environment.

The sampling of participants is a concern to external validity. We use convenience sampling to recruit respondents and with the help of other researchers, distributed the survey across several different start-up communities. Demographic information from respondent answers shows that our sample is skewed towards active companies, respondents with little experience in start-ups, young companies, and small development teams of 1–8 engineers. In these aspects, our sample fits the general characteristics of start-ups, see, for example, Giardino et al. [1, 3] and Klotins et al. [5]. However, there is a survivor bias, that is, failed start-ups are underrepresented. Thus, our results reflect state-of-practice in active start-ups.

Another threat to external validity stems from case selection. We marketed the questionnaire to start-ups building software-intensive products. However, due to the broad definition of software start-ups (see Giardino et al. [3]), it is difficult to differentiate between start-ups and small-medium enterprises. We opted to be as inclusive as possible and to discuss relevant demographic information along with our findings.

3.5.4. Conclusion validity

This type of validity threat concerns the possibility of incorrect interpretations arising from flaws in, for example, instrumentation, respondent and researcher personal biases, and external influences [42].

To make sure that respondents interpret the questions in the intended way, we conducted several pilots, workshops and improved the questionnaire afterwards. To minimize the risk of systematic errors, the calculations and the first and the third author performed statistical analysis independently, and the findings were discussed among the authors.

It could be that some respondents may lack the knowledge to fully answer our questions. We mitigate this threat by providing the "I do not know" option to all our questions. We further analyze the respondent demographics and background (see Section 4) to gauge the credibility of their responses. However, we cannot exclude that in some cases the responses are incomplete. As a result, we cannot reliably make conclusions from the absence of information in the responses.

To test if the order of appearance of Agile practices affects practitioner responses, we run a Spearman's rank-order correlation test [43]. We examine a potential relationship between the order of appearance and the frequency chosen by respondents. The results showed that there is no statistically significant correlation ($p > 0.05$).

4. Results

The majority of the surveyed start-ups (63 out of 84, 75%) are active and have been operating for 1–5 years (58 out of 84, 69%). Start-ups are geographically distributed among Europe (34 out of 84, 40%), South America (41 out of 84, 49%), Asia (7 out of 84, 8%), and North America (2 out of 84, 2%).

Our sample is about equally distributed in terms of the product development phase. We follow the start-up life-cycle model proposed by Crowne [44] and distinguish between inception, stabilization, growth, and maturity phases. In our sample, 16 start-ups have been working on a product but have not yet released it to the market, 24 teams have released the first version and actively develop it further with customer input, 26 start-ups have a stable product and they focus on gaining customer base, and another 16 start-ups have mature products, and they focus on developing variations of their products.

The questionnaire was filled in mostly by start-up founders (64 out of 84, 76%) and engineers employed by start-ups (15 out of 84, 18%). About half of the respondents have specified that their area of expertise is software engineering (49 out of 84, 58%). Others have specified marketing, their respective domains, and business development as their areas of expertise.

The respondents' length of software engineering experience ranges from 6 months to more than 10 years. A large portion of respondents (44 out of 84, 52%) had less than 6 months of experience in working with start-ups at the time when they joined their current start-up.

We provide a complete list of studied cases and their demographical information as supplemental material on-line[3].

The responses on what development type best characterizes the company suggest that most companies, 51 out of 84, 60%, follow agile and

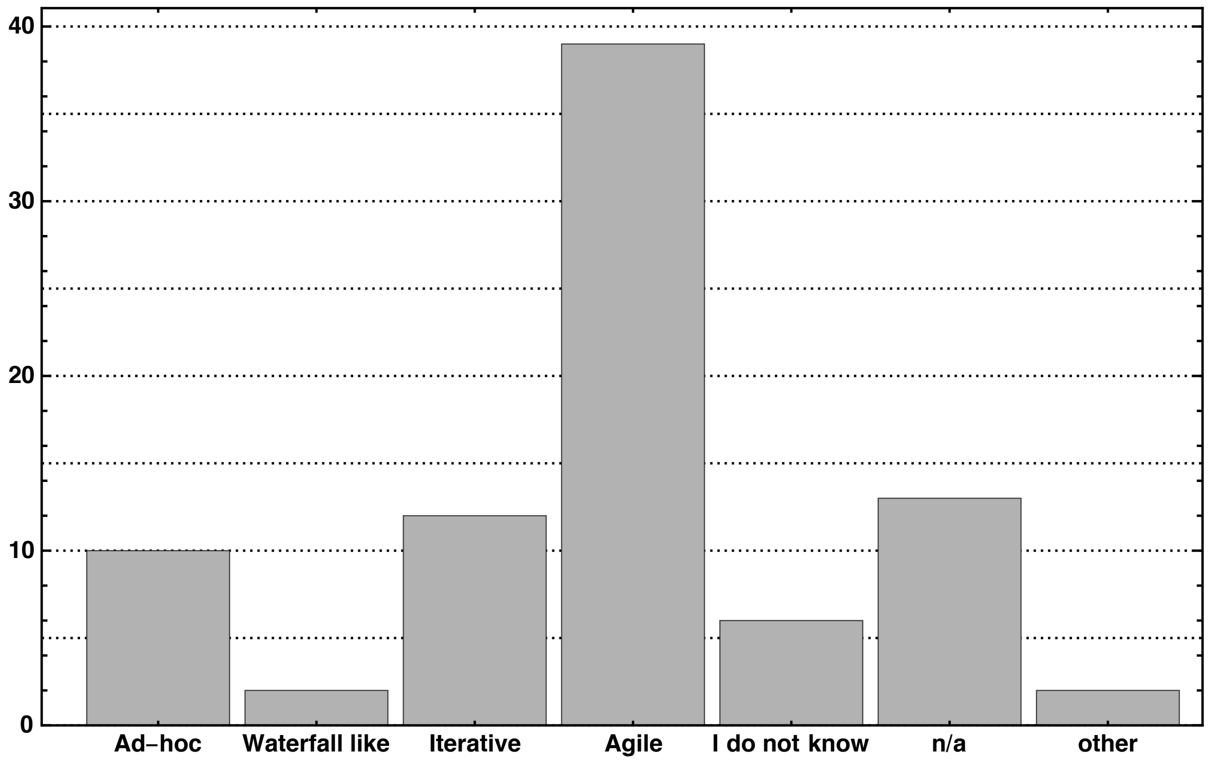


Figure 1. Use of development approaches in the studied cases

[3]The studied cases: http://eriksklotins.lv/files/GCP_demographics.pdf

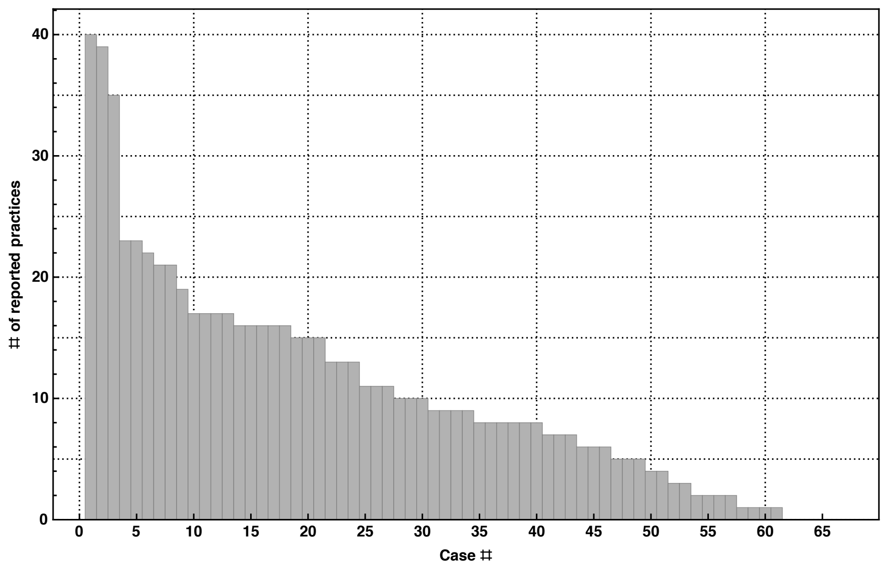


Figure 2. The number of reported agile practices in the studied start-up companies. y-axis show the number of reported practices, x-axis show the studied cases. The cases are sorted by the number of reported practices

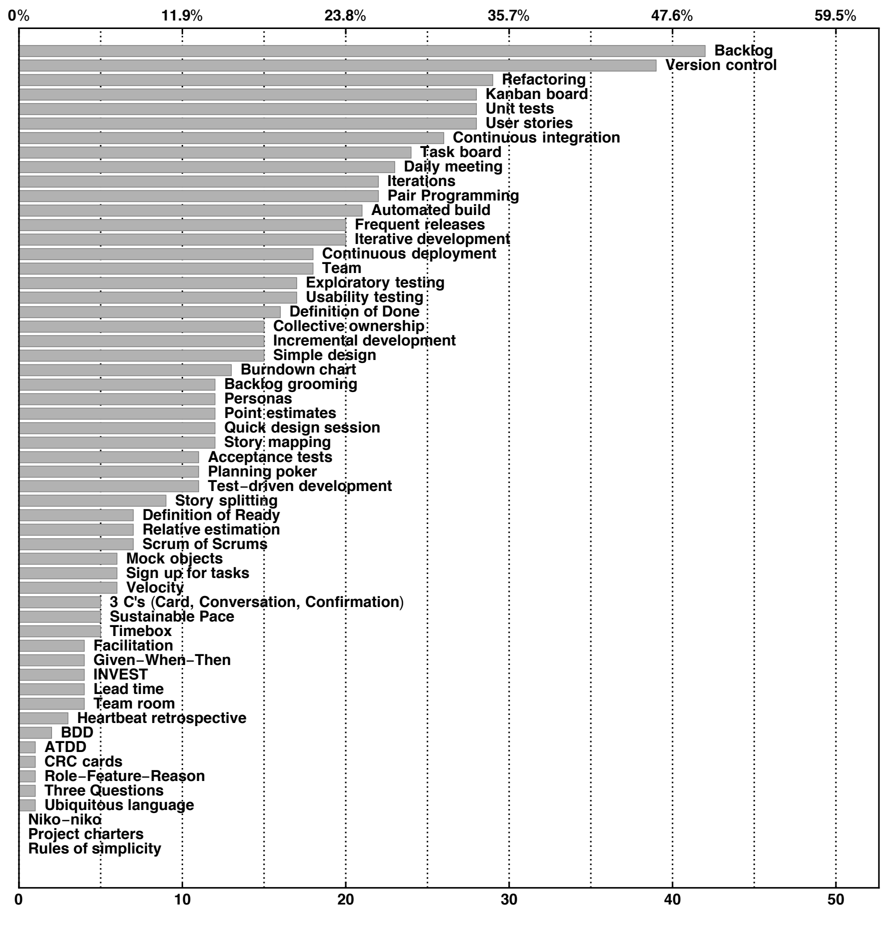


Figure 3. Frequency of Agile practices

iterative processes. A few 2 out of 84 follow a waterfall-like process, 10 companies report using an ad hoc approach, see Figure 1.

We presented respondents with a list of 56 Agile practices and asked to tick off the practices that they use in their companies. Most start-ups use between 0 and 20 Agile practices. However, the majority of companies report using only a few practices, see Figure 2. There is also a small cluster of companies reporting the use of more than 35 individual practices. Only 7 companies explicitly reported not using any agile practices, 16 respondents have not provided their answers.

The most frequently used Agile practices are backlog and version control reported by 42 and 39 companies, respectively (50% and 46% out of 84 cases). The use of other practices varies, see Figure 3. Respondents do not report the use of practices such as the Niko-Niko calendar (visualizing the team's mood changes), project charters (a poster with a high level summary of the project), and rules of simplicity (a set of criteria to evaluate source code quality).

4.1. Overview of the findings

In Table 2, we summarize the associations between the use of certain practices and product factors. In Table 3, we summarize the associations between the use of certain practices and team factors. We show only practices with statistically significant associations ($p < 0.05$). The numbers in the table show Cramer's V values denoting the strength of the associations, see Table 1 for interpretation of the values.

4.2. Interpretation of associations

An association shows that a specific practice and certain estimates of a factor are reported together. We use the Pearsons Chi-squared test ($p < 0.05$) to determine if the association is statistically significant. However, from associations alone, we cannot explain the phenomenon with confidence and guide guide practitioners in adopting Agile practices in start-ups. To explain the associations, we formulate 5 archetypes (A) of propositions characterizing the potential explanations of our findings:

It could be that a statistically significant association is a false positive. That is, the association between a practice and a factor is due to an error or some confounding factor.

A_0: There is a spurious association between P and F.

An association could point towards a causal relationship between the use of a practice (P) and

Table 2. Results of Cramer's V test on the association between product factors and use of Agile practices with $p < 0.05$. Up (↑) and down (↓) arrows denote whether the association is positive, i.e., use of the practice is associated with more positive responses, or negative, i.e., use of the practice is associated with more negative estimates from respondents

Practice	Documentation	Architecture	Source code	Testing	Overall quality
Card, Conversation, Confirmation	–	–	–	0.422↑	–
Unit tests	–	–	–	0.391↑	–
Automated build	–	–	0.374↑	–	–
Facilitation	–	–	0.330↓	–	–
Given-When-Then	–	–	0.330↓	–	–
INVEST	–	–	0.330↓	–	–
Iterations	–	0.359↑	–	–	–
Continuous integration	–	–	–	–	0.368↑
Collective ownership	–	–	–	–	0.372↓

Table 3. Results of Cramer's V test of association ($p < 0.05$) between the use of Agile practices and team factors. Up (↑) and down (↓) arrows denote whether the association is positive, i.e., use of practice is associated with more positive responses, or negative, i.e., use of practice is associated with more negative estimates from respondents

Practice	Attitudes	Pragmatism	Communication	Skills	Resources	Process
Backlog	–	–	–	–		0.401↓
Unit tests	0.379↑	–	–	–	–	–
Continuous integration	0.360↑	–	–	–	–	–
Automated build	–	–	–	–	–	0.346↓
Definition of Done	0.411↓	–	–	–	–	–
Simple design	–	–	–	–	0.365↓	–
Burndown chart	0.383↓	–	–	–	0.384↑	–
Story mapping	–	0.356↑	–	–	–	–
Relative estimation	0.399↓	–	–	–	0.399↑	–
Velocity	0.435↓	–	–	–	–	–
Team room	–	–	–	–	0.343↓	–

a factor (F). We are measuring factors through respondent evaluation, thus we cannot distinguish between actual and perceived improvements.
A_1: Use of P improves perception of F.

Some of the associations appear to be negative, i.e., the use of a practice is reported together with unfavorable estimates. It could be that the practice has adverse effects, or the use of the practice helped to expose the problematic factor:
A_2: Use of P hinders F.
A_3: Use of P exposes issues with F.

It could be that a practice is introduced as a consequence of a situation. That is, we could be observing a reverse causal relationship.
A_4: F is the cause or enabler for using P.

4.3. Specific findings

In this section, we link together our specific findings with relevant propositions, see Figure 4. In the figure we show a list of agile practices with statistically significant associations to factors. The factors are grouped into four blocks A_1–A_4 representing our propositions. The arrows denote potential explanations between factors and practices.

A product backlog is an authoritative list of new features, changes, bug fixes, and other activities that a team may deliver to achieve a specific outcome [24].

Our results show a moderately strong (Cramer's $V = 0.401$) association between the use of a backlog and worse perception of the engineering process. In particular, frequent changes in requirements, unclear objectives, and unsystematic changes hindering the engineering process are reported together with the use of the backlog.

Unit testing is a practice to develop short scripts to automate the examination of low-level behavior of the software [24].

Our findings show a moderately strong association (Cramer's $V = 0.379$) between the use of unit tests and teams' attitudes. In particular, a positive attitudes towards following the best design, coding, and testing practices are reported together with using unit testing.

Our findings also show a moderately strong association (Cramer's $V = 0.391$) between the use of unit testing and less reliance on manual testing of the product.

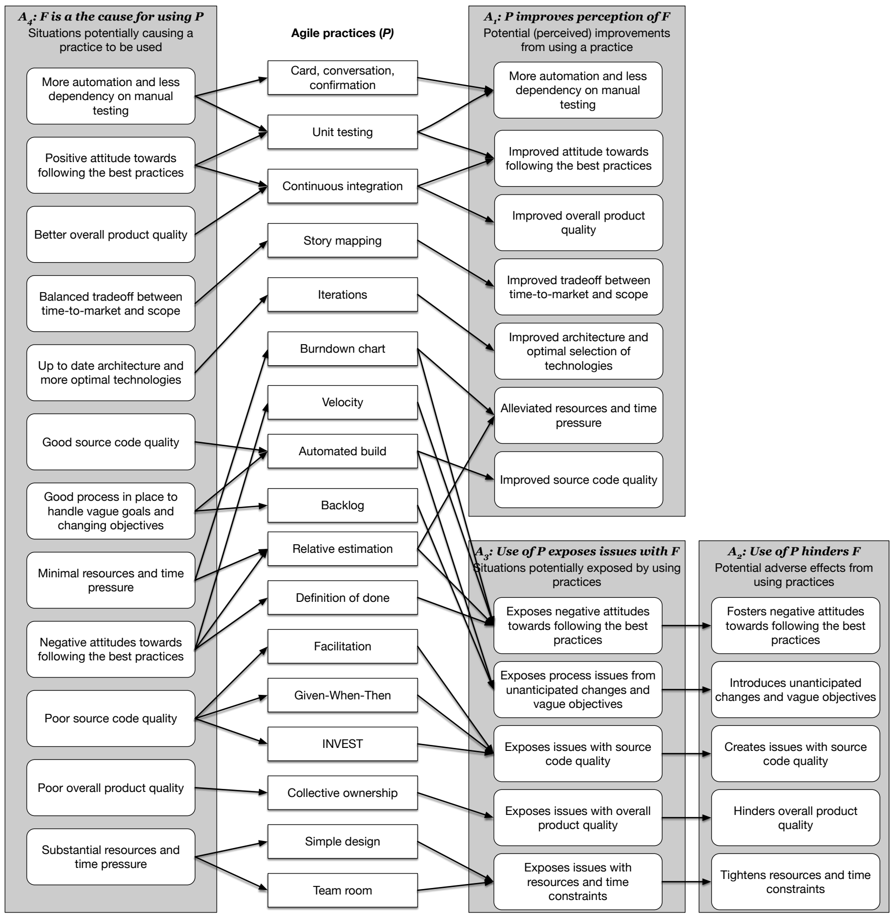

A_4: F is a the cause for using P
Situations potentially causing a practice to be used
More automation and less dependency on manual testing
Positive attitude towards following the best practices
Better overall product quality
Balanced tradeoff between time-to-market and scope
Up to date architecture and more optimal technologies
Good source code quality
Good process in place to handle vague goals and changing objectives
Minimal resources and time pressure
Negative attitudes towards following the best practices
Poor source code quality
Poor overall product quality
Substantial resources and time pressure
Agile practices (P)
Card, conversation, confirmation
Unit testing
Continuous integration
Story mapping
Iterations
Burndown chart
Velocity
Automated build
Backlog
Relative estimation
Definition of done
Facilitation
Given-When-Then
INVEST
Collective ownership
Simple design
Team room
A_1: P improves perception of F
Potential (perceived) improvements from using a practice
More automation and less dependency on manual testing
Improved attitude towards following the best practices
Improved overall product quality
Improved tradeoff between time-to-market and scope
Improved architecture and optimal selection of technologies
Alleviated resources and time pressure
Improved source code quality
A_3: Use of P exposes issues with F
Situations potentially exposed by using practices
Exposes negative attitudes towards following the best practices
Exposes process issues from unanticipated changes and vague objectives
Exposes issues with source code quality
Exposes issues with overall product quality
Exposes issues with resources and time constraints
A_2: Use of P hinders F
Potential adverse effects from using practices
Fosters negative attitudes towards following the best practices
Introduces unanticipated changes and vague objectives
Creates issues with source code quality
Hinders overall product quality
Tightens resources and time constraints

Figure 4. Overview of the findings and the propositions.
We show agile practices and different explanations for the associations (A_1–A_4)

Continuous integration aims to minimize the duration and effort of each integration episode and maintain readiness to deliver a complete product at any moment [24].

Our findings show a moderately strong association (Cramer's $V = 0.360$) between the use of continuous integration and more positive attitudes towards using sound design, coding, and testing practices.

Our findings also show a moderately strong association (Cramer's $V = 0.368$) between the use of continuous integration and more positive estimates of product internal and external quality, and less slipped defects.

Automated build is a practice to automate the steps of compiling, linking, and packaging the software for deployment [24].

Our findings show a moderately strong (Cramer's $V = 0.346$) association between the use of automated build and worse estimates in the engineering process.

Our findings also show a moderately strong (Cramer's $V = 0.374$) association between the use of automated builds and more positive estimates on the source code quality.

Definition of done is a list of criteria which a task must meet before it is considered done [24].

Our findings show a moderately strong (Cramer's $V = 0.411$) association between the use of a definition of done and worse attitudes towards following best engineering practices.

Simple design is a practice to favor simple, modular, and reusable software designs that are created as needed [24].

Our findings show a moderately strong association (Cramer's $V = 0.365$) between simple design practices and more pressing time and resource concerns.

Burndown chart is a graph visualizing the remaining work (x-axis) over time (y-axis) [24].

Our findings show a moderately strong association (Cramer's $V = 0.383$) between the use of the burndown chart and worse estimates on teams' attitudes towards following the best engineering practices.

Our findings also show a moderately strong association (Cramer's $V = 0.384$) between the use of the burndown chart and less time and resource pressure.

Story mapping is a practice to organize user stories in a two-dimensional map according to their priority and level of sophistication. Such a map is used to identify requirements for a bare-bones but usable first release, and subsequent levels of increased functionality [24].

Our findings show a moderately strong association (Cramer's $V = 0.356$) between the use of story mapping and a more pragmatic approach to handing the trade-off between time-to-market and following best engineering practices.

Relative estimation comprises of estimating task effort in relation to other similar tasks, and not absolute units [24].

Our findings show a moderately strong association (Cramer's $V = 0.399$) between the use of relative estimation and worse attitudes towards following the best testing, architecture, and coding practices.

Our results also show a moderately strong association (Cramer's $V = 0.399$) between the use of relative estimates and less time and resource pressure.

Velocity is a metric to calculate how long it will take to complete the project based on past performance [24].

Our findings show a moderately strong association (Cramer's $V = 0.435$) between the use of velocity and worse attitudes towards following the best engineering practices.

Team room is a dedicated, secluded, and equipped space for an agile team to collaborate on the project [24].

Our findings show a moderately strong association (Cramer's $V = 0.343$) between theuse of a team room and more pressing time and resource constraints.

Facilitation is a practice to have a dedicated person in the meeting, ensuring effective communication, and maintaining focus on the objectives [24].

Given-When-Then is a template for formulating user stories comprising of some contextual information, triggers or actions, and a set of observable consequences [24].

INVEST is a checklist to evaluate the quality of a user story [24].

Our findings show a moderately strong association (Cramer's $V = 0.330$) between the use of any of the three practices (Facilitation, Given-When-Then, and INVEST) and worse estimates on the product source code quality.

Iterations are time-boxed intervals in an agile project in which the work is organized. The project consists of multiple iterations, tasks, and objectives for the next iteration and is revised just before it starts [24].

Our findings show a moderately strong association (Cramer's $V = 0.359$) between the use of iterations and more positive estimates on the quality of the product architecture. Specifically, respondents report fewer workarounds, more optimal selection of technologies, and fewer issues with outdated designs.

Collective ownership is a practice to empower any developer to modify any part of the project source code [24].

Our findings show a moderately strong association (Cramer's $V = 0.372$) between collective ownership and worse estimates on the product's internal and external quality.

Card, Conversation, Confirmation is a pattern capturing the life cycle of a user story. The life cycle starts with a tangible "card", "conversations" regarding the user story occurs throughout the project; finally, a "confirmation" is received of a successful implementation of the user story [24].

Our findings show a moderately strong association (Cramer's $V = 0.422$) between the use of the life-cycle pattern and less dependence on manual testing of the product.

5. Discussion

5.1. Answers to our research questions

RQ1: How are Agile practices used in start--ups. Our results show that start-ups use Agile practices, even though they do not follow any specific agile methodology. Such results confirm earlier findings, e.g., Giardino et al. [7], and Yau and Murphy [6], stated that engineering practices and processes in start-ups gradually evolve from rudimentary and ad hoc to more systematic.

The most frequently used practices are a backlog, version control, refactoring, user stories, unit tests, and kanban board. We could not identify any clear tendencies comparing the frequencies of practices between different cohorts, e.g., team size, product stage, and team skill levels.

Our results show limited adoption of version control, a backlog, and refactoring. The use of these practices is reported only by 30–50% of cases. Disuse of such practices is reported both by respondents with and without software engineering background. This is surprising, as, e.g., version control is a widely adopted software development practice, can be applied with minimal overhead, and provide substantial benefits [45, 46]. Thus, not benefiting from version control to manage the source code is difficult to excuse with lack of resources, time shifting priorities, or other pressing concerns [3].

The use of Agile practices does not imply that an organization follows agile principles as proposed by the Agile manifesto [14]. Many of the Agile practices, for example, version control, unit testing, and refactoring, among others, could be equally well applied to other types of development methodologies. That said, a majority of start-ups characterize their development methodology as agile. Exploring the maturity of agile processes in start-ups remains a direction for further exploration [9, 47].

RQ2: What are the associations between specific Agile practices and product factors. We identify associations between the use of Agile practices and product architecture, source code quality, test automation, and the overall level of quality. We could not identify any associations regarding the quality and understandability of product documentation.

Practices related to automation, e.g., unit tests, automated build, and continuous integration, are associated with positive estimates of product factors. Practices related to requirements quality, e.g., Given-When-Then, and INVEST, show negative associations. It could be that start-ups introduce such practices as a response to the adverse effects of poor requirements. However, the causal effects of using Agile practices need to be explored further to draw any definitive conclusions.

The use of collective ownership is associated with negative estimates of overall product quality. We propose two interpretations: a) collective ownership exposes the actual state of product internal quality, b) collective ownership has adverse effects.

If two or more developers collaborate on the same part of the product, they may have a more objective view of its flaws. A single developer working on and "owning" a part of a product may be biased in estimating its quality [48].

Alternatively, inviting other developers to work on the part of a product could introduce defects. Other developers, who are not the original authors, may lack the essential contextual information to evaluate and change the component without introducing defects. Practices such as unit testing, continuous integration, and pair programming may help to prevent defects and distribute knowledge in the team. Collective ownership could be an example of a practice that must be supported by other practices to avoid adverse effects.

RQ3: What are the associations between specific Agile practices and team factors. Most associations pertain to teams' attitudes towards the following the best engineering practices. Both positive and negative attitudes towards the best engineering practices are precedents for using several practices. Automation practices, such as unit tests and continuous integration, are associated with positive attitudes. However, control and planning practices, such as the definition of done, burndown chart, relative estimation, and velocity, are associated with negative attitudes towards the following the best engineering practices. We explain such results with the need for tighter control over the team's performance when they do not see the benefits of following best practices.

We observe several associations between the use of Agile practices and respondents' estimates towards time and resource pressures. The use of burndown charts and relative estimates are associated with less pressure. We interpret such findings that the use of resource planning and control practices helps to plan any amount of resources better and alleviate the pressure.

We have not identified any associations about communication in the team. Other authors, e.g., Yau et al. [6] and Sutton [13], have identified that in small start-up teams, communication is not an issue. Small collocated teams do not need additional support for coordination. Such finding leads us to argue that the primary reasons for introducing Agile practices in start-ups are tighter control over a team's performance and resource utilization.

5.2. Implications to research

In this study, we have set forth a number propositions for further investigation. Looking at the propositions summarized in Figure 4, we identify several cross-cutting concerns to address with further studies in the area.

Our results suggest that software start-ups adopt Agile practices one by one without following any particular agile methodology, e.g., scrum or XP. Such finding is supported by earlier work, for example, Giardino et al. [7] and Gralha et al. [49], reporting that new practices are introduced gradually and aimed at addressing specific concerns. However, existing research on adopting agile software engineering considers mostly the adoption of whole methodologies, e.g., scrum or XP, and not individual practices [36]. We identify an opportunity for more fine-grained research on how to adopt Agile practices in small organizations to address their specific concerns.

Our results suggest a limited adoption of ubiquitous engineering practices such as the use of version control, backlog, and refactoring, reported only by 30–50% of cases, see Figure 3. While this result could be explained by the limitations of the survey data collection method, it also suggests a potential disuse of essential software engineering practices in start-ups. This result invites further research to understand how much neglect of the best engineering practices is motivated by the engineering context and how much by the lack of engineering acumen in start-ups.

Related work identifies the need to be more flexible and to alleviate the need for rigorous upfront planning as the primary goal for adopting agile. Other objectives include the aim to improve product quality, shorten feedback loops with customers, and to improve teams' morale [36]. Such objectives are superficial and do not support the adoption of specific practices or addressing specific start-up specific challenges [3]. We identify an opportunity to explore the precedents of introducing specific Agile practices, and longitudinal studies examining the effects of specific practices.

5.3. Implications for practitioners

Examining our findings, we identify several relevant patterns for practitioners.

Teams' attitudes towards following the best engineering practices appear as a strong denominator of adopting a range of Agile practices. Positive attitudes towards good practices drive the adoption of automated testing and continuous integration. Such practices have further positive effects on software quality.

Many respondents, both with and without a software engineering background, failed to report using, for example, version control, refactoring, and unit tests. These are standard software engineering practices, applicable with minimal overhead, and could provide substantial benefits in any software development context.

Negative attitudes towards best practices are associated with the use of practices for progress control, such as the definition of a done, burndown chart, and effort estimation. Our explanation for such a finding is that teams implement such practices to have tighter control over the development process.

Our results suggest that the primary benefits of adopting Agile practices are tighter control over the team's performance and product quality. The use of progress control practices alleviates resource pressures.

6. Conclusions

In this study, we investigate associations between the use of Agile practices and perceived impact on various product and team factors in software start-ups. Based on our findings, we set forth a number of propositions that narrow down the space of investigation for future studies on Agile practices and start-ups.

We conclude that start-ups adopt Agile practices, however do not follow any specific methodology. The use of Agile practices is associated with improved product quality, more positive attitudes towards following the best engineering practices, and tighter control over resource utilization. However, the exploration of the causal effects remains a direction of further work.

We have formulated several implications for researchers and practitioners. We identify an opportunity for more fine-grained studies (on a practice level) into the adoption and effects of Agile practices. We conclude that Agile practices show a potential to be used in start-ups setting, however, adopting individual practices without considering supporting practices could lead to adverse effects.

7. Acknowledgements

The authors of this paper would like to thank all practitioners who found time and motivation to participate in this study. Reaching this diverse population of start-ups would not be possible without help and support from Software Start-up Research Network[4] community, and specifically Nana Assyne, Anh Nguyen Duc, Ronald Jabangwe, Jorge Melegati, Bajwa Sohaib Shahid, Xiaofeng Wang, Rafael Matone Chanin, and Pekka Abrahamsson.

This work is partially funded by the Foundation for Research Support of the State of Rio Grande do Sul (17/2551-0001/205-4) and the Brazilian National Council for Scientific and Technological Development.

References

[1] C. Giardino, S.S. Bajwa, and X. Wang, "Key challenges in early-stage software startups," in *Agile Processes, in Software Engineering, and Extreme Programming*, Vol. 212, 2015, pp. 52–63.

[2] S. Blank, "Why the lean start up changes everything," *Harvard Business Review*, Vol. 91, No. 5, 2013, p. 64.

[3] C. Giardino, M. Unterkalmsteiner, N. Paternoster, T. Gorschek, and P. Abrahamsson, "What do we know about software development in startups?" *IEEE Software*, Vol. 31, No. 5, 2014, pp. 28–32.

[4] N. Paternoster, C. Giardino, M. Unterkalmsteiner, T. Gorschek, and P. Abrahamsson, "Software development in startup companies: A systematic mapping study," *Information and Software Technology*, Vol. 56, No. 10, 2014, pp. 1200–1218.

[5] E. Klotins, M. Unterkalmsteiner, and T. Gorschek, "Software engineering in start-up companies: An analysis of 88 experience reports," *Empirical Software Engineering*, Vol. 24, No. 1, 2019, pp. 68–102.

[6] A. Yau and C. Murphy, "Is a rigorous agile methodology the best development strategy for small scale tech startups?" University of Pennsylvania Department of Computer and Information Science, Tech. Rep., 2013.

[7] G. Carmine, N. Paternoster, M. Unterkalmsteiner, T. Gorschek, and P. Abrahamsson, "Software development in startup companies: The greenfield startup model," *IEEE Transactions on Software Engineering*, Vol. 42, No. 6, 2016, p. 233.

[8] J. Pantiuchina, M. Mondini, D. Khanna, X. Wang, and P. Abrahamsson, "Are software startups applying agile practices? the state of the practice from a large survey," in *International Conference on Agile Software Development.* Springer, Cham, 2017, pp. 167–183.

[9] E. Klotins, M. Unterkalmsteiner, P. Chatzipetrou, T. Gorschek, R. Prikladniki, N. Tripathi, and L. Pompermaier, "A progression model of software engineering goals, challenges, and practices in start-ups," *IEEE Transactions on Software Engineering*, 2019.

[4]The Software Start-up Research Network, https://softwarestartups.org/

[10] PitchBook Data, Inc., "European middle market report 2h 2015," Tech. Rep., 2015.

[11] PitchBook Data, Inc., "U.S. middle market report Q4 2015," Tech. Rep., 2015.

[12] E. Klotins, "Software start-ups through an empirical lens: are start-ups snowflakes?" in *1st International Workshop on Software-Intensive Business: Start-Ups, Ecosystems and Platforms, SiBW 2018, Espoo, Finland, 3 December 2018*. CEUR-WS, 2018.

[13] S.M. Sutton, E.C. Cubed, and M. Andretti, "The role of process in a software start-up," *IEEE Software*, Vol. 17, No. 4, 2000, pp. 33–39.

[14] K. Beck, M. Beedle, A. Van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries et al., "Manifesto for agile software development," 2001.

[15] L. Rising and N.S. Janoff, "The Scrum software development process for small teams," *IEEE Software*, No. August, 2000, pp. 26–32.

[16] V.E. Jyothi and K.N. Rao, "Effective implementation of agile practices-incoordination with lean kanban," *International Journal on Computer Science and Engineering*, Vol. 4, No. 1, 2012, p. 87.

[17] P. Diebold and M. Dahlem, "Agile practices in practice: a mapping study," in *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*. ACM, 2014, p. 30.

[18] S. Jalali and C. Wohlin, "Global software engineering and agile practices: a systematic review," *Journal of software: Evolution and Process*, Vol. 24, No. 6, 2012, pp. 643–659.

[19] T. Chow and D.B. Cao, "A survey study of critical success factors in agile software projects," *Journal of Systems and Software*, Vol. 81, No. 6, 2008, pp. 961–971.

[20] S. Misra, V. Kumar, U. Kumar, K. Fantazy, and M. Akhter, "Agile software development practices: evolution, principles, and criticisms," *International Journal of Quality and Reliability Management*, Vol. 29, No. 9, 2012, pp. 972–980.

[21] A. Solinski and K. Petersen, "Prioritizing agile benefits and limitations in relation to practice usage," *Software Quality Journal*, Vol. 24, No. 2, 2016, pp. 447–482.

[22] E. Mkpojiogu, N. Hashim, A. Al-Sakkaf, and A. Hussain, "Software startups: Motivations for agile adoption," *International Journal of Innovative Technology and Exploring Engineering*, Vol. 8, No. 8S, 2019, pp. 454–459.

[23] R. Souza, L. Rocha, F. Silva, and I. Machado, "Investigating agile practices in software startups," in *Proceedings of the XXXIII Brazilian Symposium on Software Engineering*, 2019, pp. 317–321.

[24] Agile Alliance, "Agile glossary," https://www.agilealliance.org/agile101/agile-glossary/, 2018, [Online; accessed 20-April-2018].

[25] L. Layman, L. Williams, and L. Cunningham, "Exploring extreme programming in context: an industrial case study," in *Agile Development Conference*. IEEE, 2004, pp. 32–41.

[26] S. Ilieva, P. Ivanov, and E. Stefanova, "Analyses of an agile methodology implementation," in *Proceedings. 30th Euromicro Conference*. IEEE, 2004, pp. 326–333.

[27] B. Kitchenham and S.L. Pfleeger, "Software quality: the elusive target [special issues section]," *IEEE Software*, Vol. 13, No. 1, 1996, pp. 12–21.

[28] E. Tom, A. Aurum, and R. Vidgen, "An exploration of technical debt," *Journal of Systems and Software*, Vol. 86, No. 6, 2013, pp. 1498–1516.

[29] G. Lucassen, F. Dalpiaz, J.M.E. van der Werf, and S. Brinkkemper, "Forging high-quality user stories: towards a discipline for agile requirements," in *IEEE 23rd international requirements engineering conference (RE)*. IEEE, 2015, pp. 126–135.

[30] R. Khaled, P. Barr, J. Noble, and R. Biddle, "System metaphor in extreme programming: A semiotic approach," in *7th International Workshop on Organizational Semiotics*. Citeseer, 2004.

[31] E.F. Collins and V.F. de Lucena, "Software test automation practices in agile development environment: An industry experience report," in *7th International Workshop on Automation of Software Test (AST)*. IEEE, 2012, pp. 57–63.

[32] R. Moser, P. Abrahamsson, W. Pedrycz, A. Sillitti, and G. Succi, "A case study on the impact of refactoring on quality and productivity in an agile team," in *IFIP Central and East European Conference on Software Engineering Techniques*. Springer, 2007, pp. 252–266.

[33] B. Selic, "Agile documentation, anyone?" *IEEE Software*, Vol. 26, No. 6, 2009.

[34] F. Palomba, G. Bavota, M. Di Penta, R. Oliveto, and A. De Lucia, "Do they really smell bad? A study on developers' perception of bad code smells," in *IEEE International Conference on Software Maintenance and Evolution*. IEEE, 2014, pp. 101–110.

[35] M. Mantyla, J. Vanhanen, and C. Lassenius, "A taxonomy and an initial empirical study of bad smells in code," in *International Conference on Software Maintenance*, 2003, pp. 381–384.

[36] T. Dybå and T. Dingsøyr, "Empirical studies of agile software development: A systematic review," *Information and Software Technology*, Vol. 50, No. 9-10, 2008, pp. 833–859.

[37] E. Klotins, "Using the case survey method to explore engineering practices in software start-ups," in *Proceedings of the 1st International Workshop on Software Engineering for Startups*. IEEE Press, 2017, pp. 24–26.

[38] R. Larsson, "Case survey methodology: Quantitative analysis of patterns across case studies," *Academy of Management Journal*, Vol. 36, No. 6, 1993, pp. 1515–1546.

[39] A.C. Hope, "A simplified Monte Carlo significance test procedure," *Journal of the Royal Statistical Society. Series B (Methodological)*, 1968, pp. 582–598.

[40] J. Cohen, *Statistical power analysis for the behaviour sciences*. Lawrence Erlbaum Associates, 1988.

[41] A. Agresti, *An introduction to categorical data analysis*. Wiley New York, 1996, Vol. 135.

[42] P. Runeson, M. Höst, A. Rainer, and B. Regnell, *Case study research in software engineering*. John Wiley and Sons, Inc., 2012.

[43] W.W. Daniel, "Spearman rank correlation coefficient," *Applied Nonparametric Statistics*, 1990, pp. 358–365.

[44] M. Crowne, "Why software product startups fail and what to do about it," in *Engineering Management Conference*. Cambridge, UK: IEEE, 2002, pp. 338–343.

[45] B. De Alwis and J. Sillito, "Why are software projects moving from centralized to decentralized version control systems?" in *ICSE Workshop on Cooperative and Human Aspects on Software Engineering*. IEEE, 2009, pp. 36–39.

[46] E. Daka and G. Fraser, "A survey on unit testing practices and problems," in *IEEE 25th International Symposium on Software Reliability Engineering*. IEEE, 2014, pp. 201–211.

[47] L. Gren, R. Torkar, and R. Feldt, "The prospects of a quantitative measurement of agility: A validation study on an agile maturity model," *Journal of Systems and Software*, Vol. 107, 2015, pp. 38–49.

[48] M. Ozer and D. Vogel, "Contextualized relationship between knowledge sharing and performance in software development," *Journal of Management Information Systems*, Vol. 32, No. 2, 2015, pp. 134–161.

[49] C. Gralha, D. Damian, A.I.T. Wasserman, M. Goulão, and J. Araújo, "The evolution of requirements practices in software startups," in *Proceedings of the 40th International Conference on Software Engineering*. ACM, 2018, pp. 823–833.

e-Informatica Software Engineering Journal, Volume 15, Issue 1, 2021, pages: 65–84, DOI 10.37190/e-Inf210104

A Framework for the Regression Testing of Model-to-Model Transformations

Issam Al-Azzoni[*], Saqib Iqbal[*]
[*]*Department of Software Engineering and Computer Science, Al Ain University, Al Ain, UAE*
issam.alazzoni@aau.ac.ae, saqib.iqbal@aau.ac.ae

Abstract

Background: Model transformations play a key role in Model-Driven Engineering (MDE). Testing model transformation is an important activity to ensure the quality and correctness of the generated models. However, during the evolution and maintenance of these model transformation programs, frequently testing them by running a large number of test cases can be costly. Regression test selection is a form of testing, which selects tests from an existing test suite to test a modified program.
Aim: The aim of the paper is to present a test selection approach for the regression testing of model transformations. The selected test case suite should be smaller in size than the full test suite, thereby reducing the testing overhead, while at the same time the fault detection capability of the full test suite should not be compromised.
Method: approach is based on the use of a traceability mapping of test cases with their corresponding rules to select the affected test items. The approach is complemented with a tool that automates the proposed process.
Results: Our experiments show that the proposed approach succeeds in reducing the size of the selected test case suite, and hence its execution time, while not compromising the fault detection capability of the full test suite.
Conclusion: The experimental results confirm that our regression test selection approach is cost-effective compared to a retest strategy.

Keywords: Model Transformation, Regression Testing, MDE

1. Introduction

Model-Driven Engineering (MDE) refers to representing, designing, and developing a system in the form of models [1]. A model is a core artifact of MDE, which refers to an abstract representation of data and behavior of a system. Model transformations are currently used in a variety of industrial projects [2] and ensuring their correctness is important [3, 4]. Model transformation programs are frequently changed during the evolution and maintenance phases of their life cycle. Several techniques for testing model transformations have been proposed in the literature [5, 6]. However, these techniques generally require executing a large number of test cases to ensure the desired coverage criteria. This can be time-consuming and may require days or even weeks to complete.

During the regression testing of model transformations, a test suite is generally available for reuse [7]. However, a retest-all approach in which all tests are rerun may consume excessive time and resources. In contrast, regression test selection techniques aim to reduce the time required to retest a modified program by selecting a smaller subset of the existing test suite [7, 8].

There have been many regression test selection techniques proposed for conventional software written in regular programming lan-

Submitted: 15 August 2020; Revised: 31 January 2020; Accepted: 09 February 2021; Available online: 29 March 2021

guages [8]. However, only a few exist for testing model transformation programs. For example, the work of Alkhazi et al. [3] proposes an approach for test case selection for model transformations based on a multiobjective search. Another work by Shelburg et al. [9] examined the issue of determining which test cases become invalid when changes occur in a model transformation program. The work of Troya et al. [4] presents an approach for fault localization for model transformations.

In this paper, we present a framework for the regression testing of model transformations, which is based on the use of a metamodel that links test cases to their corresponding test items and test artifacts. To the best of our knowledge, this is the first paper specifically proposing a framework for the regression testing of model transformation programs based on traceability models. We present a tool that can automatically create the traceability model, given the source meta-model and a set of input test models. The tool can also be queried to obtain the set of selected test cases for a changed rule.

Regression testing approaches can be classified into three main categories [8]: test suite minimization, test case selection, and test case prioritization. Test suite minimization approaches aim to reduce the size of a test suite by permanently eliminating redundant test cases from the test suite. Test case selection approaches aim to select a subset of test cases that will be used to test the changed parts of the software. Finally, test case prioritization approaches attempt to order test cases in a way that maximizes desirable properties, such as early fault detection. In the context of model transformation testing, our proposed approach can be classified into the test case selection category.

The main contributions of this paper are summarized as follows:

1. We present a test case metamodel that can be exploited in the regression testing of model transformations.
2. We present a tool that can automatically build the required traceability models and select test cases based on the names of changed rules.
3. We demonstrate the effectiveness of the proposed framework using several experiments which involve introducing several mutations to the model transformation program and being able to kill all the mutants using only a subset of the test case set that is chosen based on the framework. The experiments also demonstrate the time saving benefit of the proposed approach.

The organization of the rest of this paper is as follows: First, we provide the necessary background and a motivating example in Section 2. Our proposed approach for regression testing is presented in Section 3. Section 4 discusses and evaluates our experiments. The related literature is discussed in Section 5. The conclusion and future work are discussed in Section 6.

2. Study background and motivating example

This section provides a description of the core concepts and terms used in this research. In addition, we present a model transformation example which motivates the regression test selection framework presented in the paper.

2.1. Models and model transformation

Models play a central role in MDE. A model represents a simplified or abstract representation of a part of a world (system) [10]. Models of a system help to analyze certain properties of the system without the need to consider its full details. Models help designers and architects to deal with the complexity present in systems. The model needs to conform to a metamodel. This means that the model needs to satisfy the rules defined in the metamodel and it must respect its semantics. A meta-modeling language is used to specify metamodels. For example, Ecore is a meta-modeling language used to specify metamodels in the Eclipse Modeling Framework (EMF) [11].

Models can be transformed into other models allowing for several types of analysis at different levels of abstraction. Model transformation refers

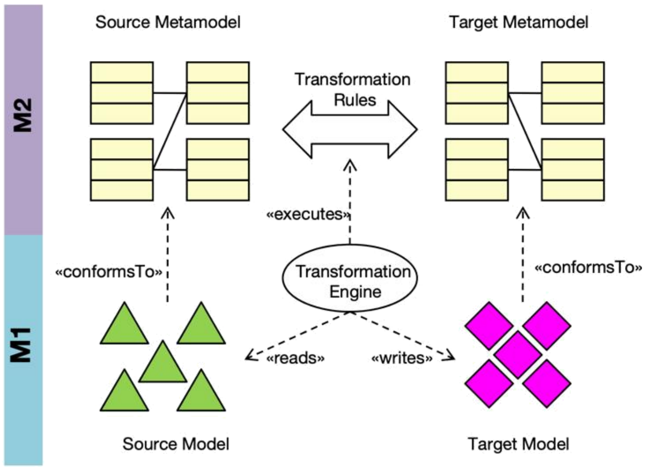
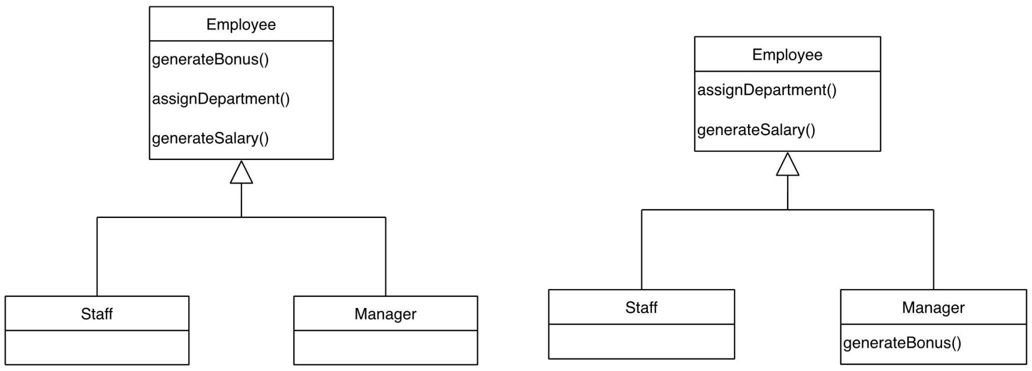


Figure 1: Model transformations pattern [12]



(a) Class diagram before executing the push down method transformation

(b) Class diagram after executing the push down method transformation

Figure 2: Class diagram before and after executing the push down method transformation

to automatically creating a target model from an existing source model by following transformation rules. A source model that conforms to a source metamodel is transformed into a target model that conforms to a target metamodel. A model-to-model transformation language is used to specify the transformation rules. The Epsilon Transformation Language (ETL), one of the Epsilon platform [13, 14] set of languages and tools, is a model-to-model transformation language that can be used to transform models specified in metamodels conforming to Ecore. ETL builds on the Epsilon Object Language (EOL), which is the main language in Epsilon for providing common model management facilities upon which several Epsilon-based languages are based. The ATLAS Transformation Language (ATL) is another model transformation language developed on top of the Eclipse platform [15].

Figure 1, taken from [12], depicts the general pattern of model transformations. *M1* in Figure 1 represents the models, which are instances of the metamodels represented by *M2*. Transformation rules are specified to transform the model elements of the source model to the model elements of the target model. The transformations are performed by a transformation engine, which reads a source model conforming to a source metamodel and produces a target model that conforms to a target metamodel.

Figure 2 shows an example of a model transformation where a push-down method transformation has been applied to a class diagram. The transformation pushed one of the methods from the super class to a child class while implementing better cohesiveness. Figure 2 (a) shows the source model and Figure 2 (b) shows the target model. The automatic model transformation saves time and energy compared to the manual conversion and brings in the consistency of the information provided the model transformation is correctly defined.

In this paper, we used several languages and tools from Epsilon. Epsilon [13, 14] is a family of languages and tools for automating model management tasks, such as model transformation and model testing. Epsilon is built on top of the Eclipse platform and supports EMF-based model management tasks. One of the languages of Epsilon is the Epsilon Unit Testing Framework (EUnit)'s language. EUnit [16, 17] is a unit testing framework, specifically designed to test model transformations. It is based on EOL and Ant's workflow task description. EUnit provides a language for comparing models. Test cases can be defined, reused, and automatically run on different sets of models. The results of the test cases can be viewed, and the differences between the expected and the actual models are graphically visualized. EUnit uses EMF Compare as the comparison engine [18]. For example, if the value of an attribute of an element in the source model is not equal to the value of the same element in the target model, then the two models are considered not equivalent. In this case, EUnit reports the test case as a failed one.

2.2. Motivation example

In this example, we consider a model transformation example that uses ETL to transform a model that conforms to an Object-Oriented (OO) metamodel to a model that conforms to a Database (DB) metamodel. The code of the example was obtained from [19] and is presented in the Appendix. Model transformation from OO to DB models is a classic example which has been used by several authors in the literature to evaluate new techniques and approaches [20–22].

Figure 3 shows the OO metamodel. We only include the metamodel elements that are involved in the ETL transformation. The names of abstract classes are shown in italics. There are four concrete classes in the OO metamodel: *Class*, *Attribute*, *Reference*, and *Package*. In an OO model, a package can be composed of other packages and classes. A class can have features: attributes and references. A feature can have a type: a *DataType* in the case of an attribute and a *Class* in the case of a reference. A class may extend another class. In this case, the class will inherit all features of the parent class.

Figure 4 shows the DB metamodel. We only include the metamodel elements that are involved in the ETL transformation. The names of ab-

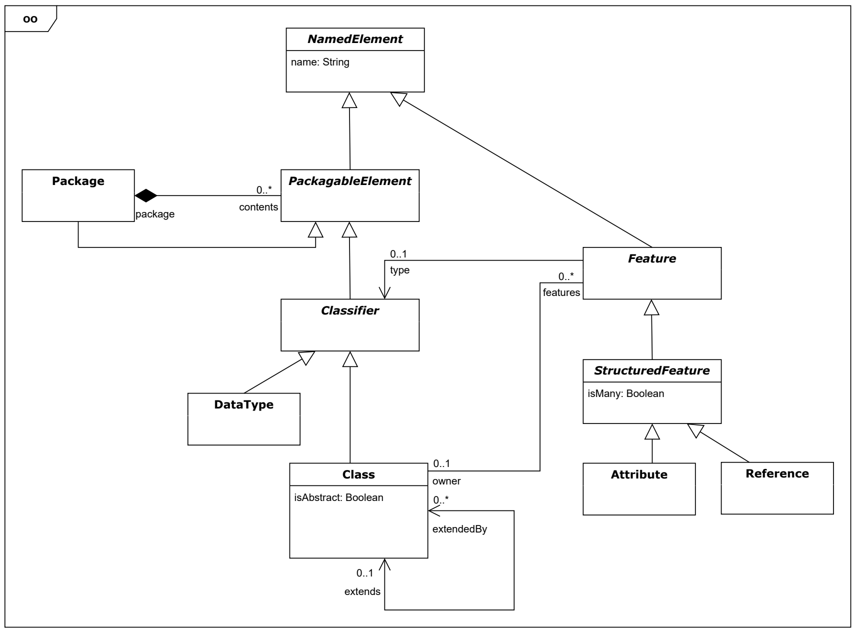


Figure 3: The OO metamodel

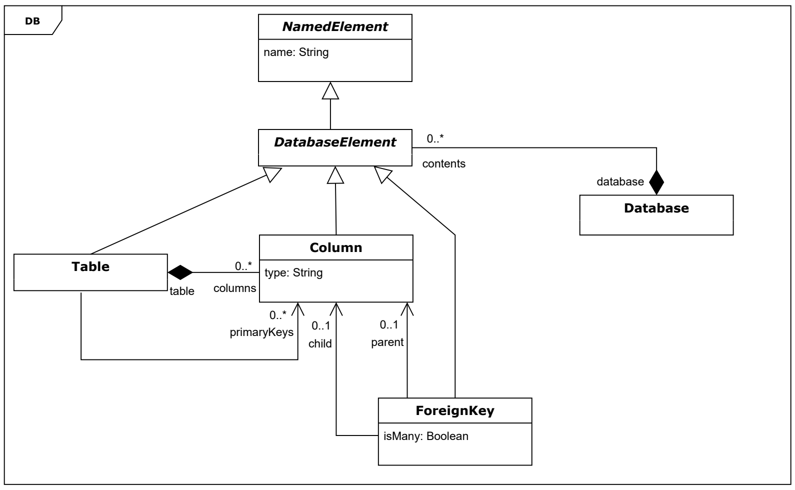


Figure 4: The DB metamodel

stract classes are shown in italics. There are four concrete classes in the DB metamodel: *Table*, *Column*, *ForeignKey*, and *Database*. In a DB model, a database is composed of database elements: tables, columns, and foreign keys. A table is composed of zero or more columns. For a table, a set of columns represent the primary keys for the table. A foreign key is associated with two columns: a child column and a parent column that could be in different tables. The table has a name attribute. Columns and foreign keys have name and type attributes.

The ETL code for the OO to DB model transformation consists of four rules: *Class2Table*, *SingleValuedAttribute2Column*, *MultiValuedAttribute2Table*, and *Reference2ForeignKey*. A brief description of each rule is provided in the comments. In ETL, each rule has a unique name and also specifies one source element and at least one target element. A rule can optionally define a guard which is a Boolean expression specifying a condition that must be met by the source element in order to apply the rule on that element. The body of a rule contains the logic specifying how to populate the values of the features of the target model elements. Some of the rules invoke one or more of the three user-defined EOL operations listed at the end of the code: *primaryKeyName*(), *valuesTableName*(), and *toDbType*(). The operation *primaryKeyName*() returns a string that represents the name of the primary key of the new table. The operation *valuesTableName*() returns a string that represents the name of the new table corresponding to a multivalued attribute in the OO model. The operation *toDbType*() returns a string that represents a type in the DB model.

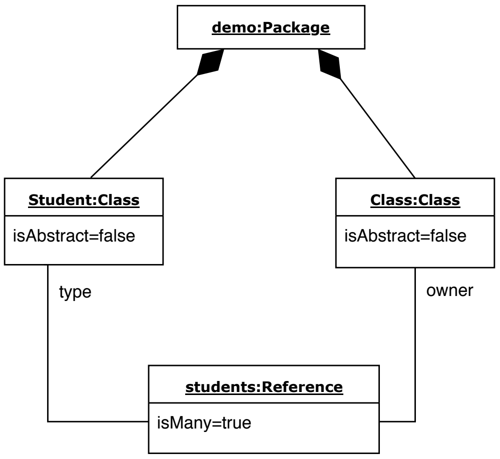


Figure 5: A sample source test model

During the testing of the model transformation program, a tester typically creates a large number of test cases. In each test case, the tester defines two test models: the first one is the source test model and the second one is the target test model. The source test model conforms to the source metamodel, while the target test model conforms to the target metamodel. The assumption is that in order for the test case to pass, the model generated by the model transformation program when the input is the source model must be equivalent to the target model. Figure 5

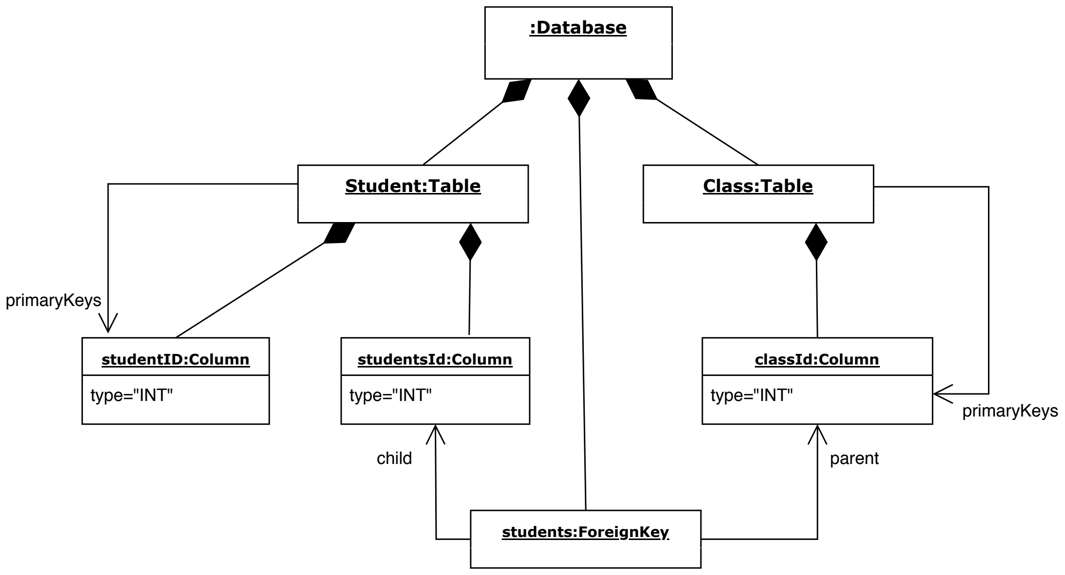


Figure 6: The target model corresponding to the source model in Figure 5

shows a sample source test model. It is an OO model containing two *Classes*: *Student* and *Class* contained in the *Package* named *demo*. Both *Classes* are not abstract (the value of *isAbstract* is *false* in both *Classes*). There is a *Reference* named *students* whose *owner* is *Class* and *type* is *Student*. The target DB model obtained when executing the model transformation is shown in Figure 6. As expected, the *Database* contains two *Tables*: *Student* and *Class*. The primary key columns, *classID* and *studentID*, are created by the rule *Class2Table*. The *ForeignKey* named *students* and the foreign key column *studentsId* are created by the rule *Reference2ForeignKey*. The *parent* and *child* references of the *students ForeignKey* are also set by the same rule to *classId* and *studentsId*, respectively.

The problem that this paper aims to tackle can be summarized as follows. Suppose that a tester has created a large number of test cases, each with its corresponding source and target models. During the maintenance and evolution of the model transformation program, several changes can be made to the program. A change usually involves one or more of the transformation program's rules. When a change is made to the program, a tester needs to rerun the test cases to ensure the correctness of the program. However, instead of rerunning the whole test case suite, this paper proposes a new framework that selects a subset of the test cases in a way that does not reduce the fault detection capability of the original suite. The framework exploits the traceability links between the rules in the transformation program and the source and target models of the related test cases. By applying this regression test selection framework, the tester benefits from the overhead and execution time that are saved when using a smaller subset of test cases for testing.

3. Approach

The proposed framework for regression test selection for model transformation relies on the use of a metamodel that links test cases with their corresponding test items and test artifacts. A test item represents an item under test. In the context of model transformation, a test item corresponds to a rule (or a statement block within the rule). A test case has a name that is used as an identifier for the test case. The test artifacts identify the test models related to the test item, including the source (input) and target (expected) models.

A metamodel that meets such requirements is shown in Figure 7. The metamodel is named as *TestCasesMM*. A test case set consists of zero or more test cases. Each test case is associated with a test item which corresponds to the rule being

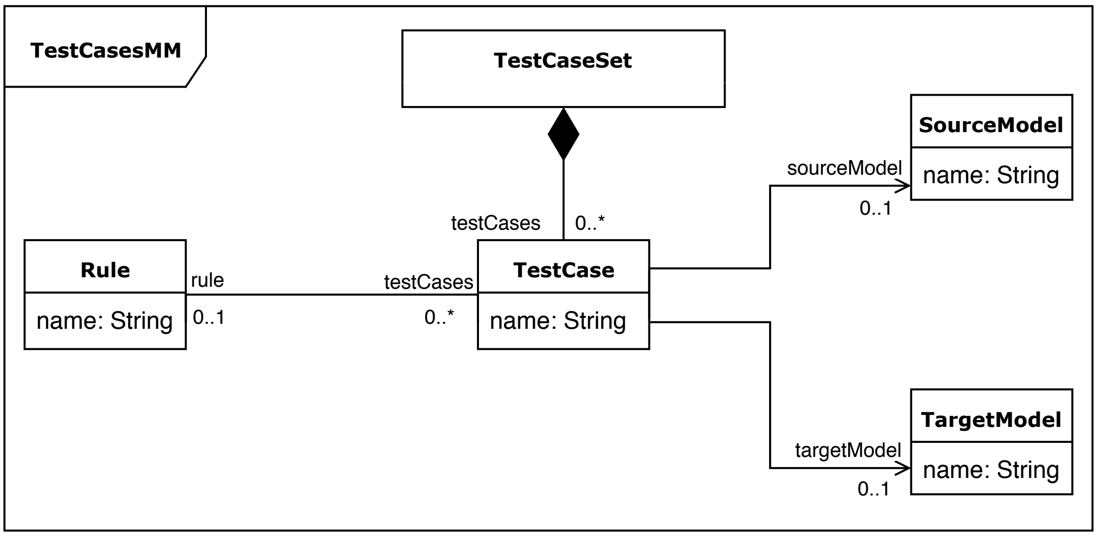


Figure 7: The test cases metamodel

tested by the test case. In cases where a test case is related to more than one rule, the metamodel definition allows to have repeated instances of the *TestCase* meta-class, with the same name, where each one is linked to an instance of the *Rule* meta-class. In addition, a test case has two test artifacts: a source model and a target model. When a rule is changed, the model of the test case set which conforms to *TestCasesMM* can be used to identify the test cases that need to be reexecuted. In addition, the model can be used to identify the source and target models required by each test case.

The traceability links in a test case model can also be used to provide information on the adequacy of the test cases. One example is checking that every rule is linked to at least one test case. If a rule is not linked to any test case, this could indicate that this rule is not covered by the test case set. In this case, a tester would need to reexamine the test case set and add new test cases to improve its coverage. In the Object Constraint Language (OCL) [23] which is the common standard to express constraints on models specified using object-oriented metamodeling technologies, the constraint that every rule is linked to at least one test case can be stated as follows:

```
context Rule
inv RuleLinkedToAtLeastOneTestCase:
self.testCases->notEmpty()
```

Here, for a test case model to be valid, this constraint requires that the collection of test cases linked to any rule is not empty. The keyword **context** is used to specify the model element to which the statement of the constraint applies. This statement is a Boolean expression referred to as invariant. The keyword **inv** is used to specify the invariant: first the name of the invariant is provided, followed by a colon, which is followed by the Boolean expression. In the expression, *self* refers to the model element on which the constraint is evaluated (i.e., whose name follows the keyword **context**). Note that this example is just one possible option for describing constraints on test case models and that our approach does not require it.

We have developed a tool to automate the regression test selection process. The main objective of the tool is to automatically create the test case model which conforms to the metamodel shown in Figure 7. The project files are available on GitHub at https://github.com/ialazzon/RegressionTestSelectionTool. The tool is a Java program that does the following. First, it parses the model transformation program and the EUnit file containing the test case definitions. It also reads all source test model files that are used by the test cases defined in the EUnit file. Then, it automatically creates the test case model, as specified by our approach. Finally, when a tester enters the name of a modified rule in the model transformation program, the tool shows the contents of a new EUnit file defining the test cases selected by our approach. The tester can run this EUnit file to execute the selected test cases, rather than rerunning the original EUnit file containing the full set of test cases. The tool was used in all experiments conducted when validating the approach (see Section 4).

4. Validation

To evaluate our approach, we conducted several experiments using four different model transformation programs. This section starts with a list of two research questions that the experiments attempt to address. This is followed by a discussion of the experimental setup and results. Finally, a discussion on the threats to validity is provided.

4.1. Research questions

We defined two research questions as follows:

- **RQ1:** How does our approach of regression test selection compare with a re-test-all strategy? Investigating this question serves as a sanity check step. If the results of our experiments show no benefit of using our approach, then we can conclude that our approach is not needed and hence a tester would be better off rerunning all test cases in a test suite.
- **RQ2:** How cost-effective is our approach? We want to ensure that the fault detection capability when rerunning a selected set of test cases is not compromised, while at the same time there is a significant saving in the overhead involved when running these test cases.

4.2. Experimental setup and results discussion

To evaluate these research questions, we applied our approach using the automated tool discussed in Section 3 on four different transformation programs. The first one is the *OO2DB* transformation which has been presented as a motivating example in Section 2.2. The second one is the *QN2QPN* transformation which transforms queueing networks into queueing Petri nets. The *QN2QPN* transformation is written in ATL [15]. It was developed by one of the authors and is presented in [24]. The third one is the *BibTeX2DocBook* transformation which transforms a BibTeX model to a DocBook composed document [25]. The fourth one is the *CPL2SPL* transformation which transforms a CPL model to an SPL model [26]. CPL and SPL are two domain-specific languages used in telephony systems. Both *BibTeX2DocBook* and *CPL2SPL* transformations are written in ATL and available in the ATL Zoo [27].

Consider the *OO2DB* transformation. For regression testing, we adopt the test case metamodel, *TestCasesMM*, shown in Figure 7. Figure 8 shows part of the test case set model created by our tool. It shows three test cases: *TC1*, *TC2*, and *TC3*, with their associated rules and test artifacts. For example, the test case *TC1* is designed to test the rule *Reference2ForeignKey*. For this test case, the source model is *OO1* and the target model is *DBExpected1*. Note that the model *OO1* is the model shown in Figure 5 and the model *DBExpected1* is the model shown in Figure 6.

We manually created a total of 15 test cases using EUnit [17]. Five of the test cases were designed to achieve full line coverage. These test cases apply a white-box approach for testing the model transformation code. For example, test case *TC1* whose input model is shown in Figure 5 covers the rule *Reference2Foreign*. Also, two test cases *TC3* and *TC5* are needed to cover the rule *Class2Table* since it has a Boolean condition: one for the case where a class extends some other class and one for the case where no such class exists. The remaining test cases were designed to achieve full partition coverage for all classes, attributes, and associations in the *OO* metamodel. This test design applies the coverage adequacy criteria as defined by the black-box model transformation testing approach in [28]. For example, in the *OO* metamodel (see Figure 3) a *Class* can have zero or more features. Following the approach in [28], if a property

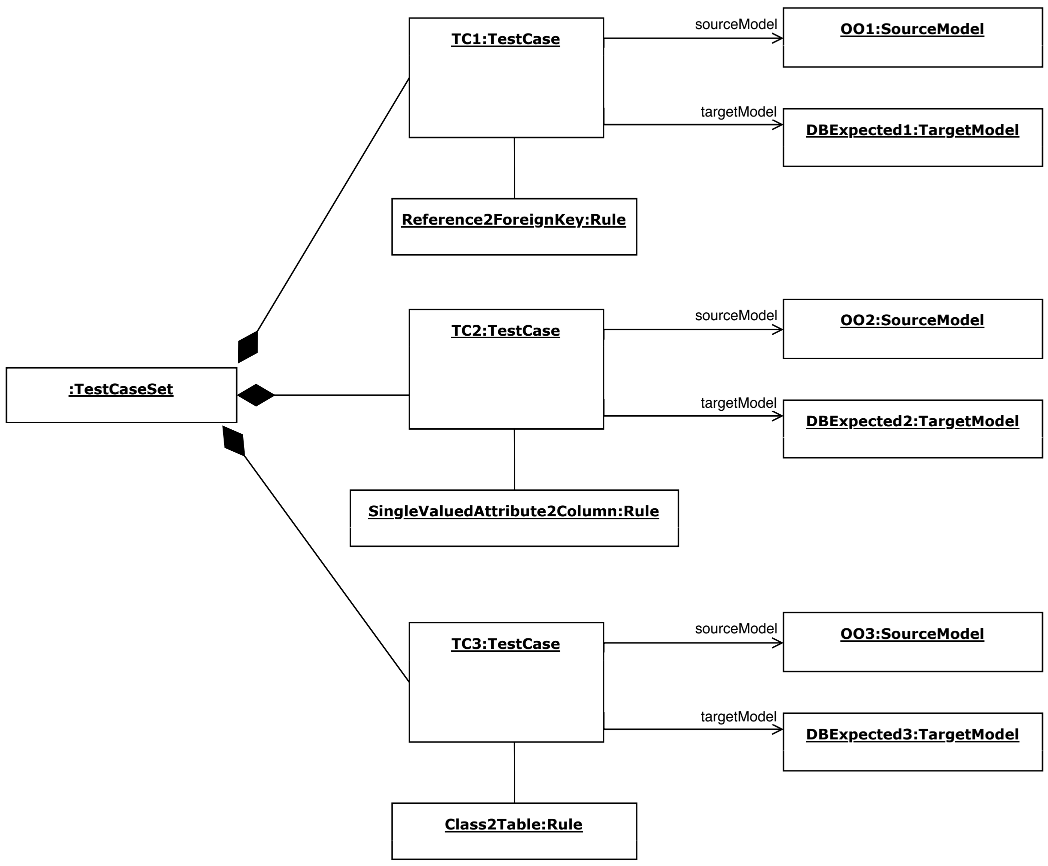


Figure 8: A partial model of the test case set

Table 1: The list of mutants

Mutant Number	Line Number	Original Code	Modified Code
1	102	fkCol.type="INT";	fkCol.type="INTX";
2	54	c.name=a.name;	c.name="name";
3	86	fkCol.type="INT";	fkCol.type="INTX";
4	19	t.columns.add(pk);	Line removed
5	36	childFkCol.type="INT";	childFkCol.type="INTX";

Table 2: The test cases corresponding to the mutants in Table 1 and their results

Mutant Number	Affected Rule	Selected Test Cases	No. of Selected Test Cases	Coverage	No. of Failed Test Cases
1	Reference2ForeignKey	*TC1*, *TC9*	2	13.33%	2
2	*SingleValuedAttribute2Column*	*TC2*, *TC6–8*	4	26.67%	3
3	MultiValuedAttribute2Table	*TC4*	1	6.67%	1
4	Class2Table	*TC3*, *TC5-15*	12	80.00%	12
5	Class2Table	*TC3*, *TC5-15*	12	80.00%	3

has a multiplicity of 0..*, a partition such as $\{\{0\}, \{1\}, \{x\}\}$, with $x \geq 2$, is defined to ensure that the test model contains instances where this property holds with zero, one and more than one object. Test cases *TC6*, *TC7*, and *TC8* include a *Class* which has zero, one, and two attributes, respectively.

We introduced several mutations to model transformation. For each mutant, our tool builds the model of the test case set to trace the affected rules to the test cases that need to be rerun. Table 1 shows a list of the five mutants used in the experiments. For each mutant, Table 2 shows the set of test cases selected by our tool in addition to their total number and coverage and the test case results. Here, coverage refers to the proportion of the number of selected test cases out of the total number of available test cases. The results show that our tool was successful in reducing coverage in every case. It can also be observed that at least one test case failed for each mutant. This indicates that using the traceability links in the test case model was effective in identifying a subset of test cases that need to be rerun and that are able to kill the mutants without requiring to rerun the complete set of test cases. In other words, the reduction in coverage of the selected test cases did not compromise the fault detection capability of our tool.

Consider the *QN2QPN* model transformation. This transformation consists of 10 rules. Table 3 shows the information and results concerning ten experiments using the *QN2QPN* transformation. We created a mutation on a single rule in every experiment. For each experiment, the table shows which rule was mutated and the type of mutation that was applied. The experiments covered the four main types of mutation operators on ATL transformations presented by Troya et al. [29]. These mutation operators were introduced by previous researchers to resemble common semantic faults that programmers make in model transformations [4]. Our experiments were designed to have a good coverage of all mutation operators proposed in literature. For each experiment, the table shows the number of test cases that were run and the execution time in milliseconds (ms) when rerunning all the test cases and when rerunning only the test cases selected by our tool. The percentages shown in red are the reductions (savings) that were achieved when rerunning the selected test cases only vs. rerunning all test cases. Larger values for the reductions represent higher savings.

To report the execution times, we conducted the experiments on a desktop with 4-core CPU running at 2.7 GHz and 8 GBs of RAM. Every measurement was repeated 10 times and the average values are presented here. Before each repetition, a warm-up session was conducted to eliminate the overhead caused by initialization processes in EUnit. The execution time of running a test suite was reported by EUnit as the wallclock time of executing the whole test suite.

Table 3: Results for the QN2QPN model transformation example

ID	Rule Affected	Type of Change	Rerun All		Rerun Selected Only	
			# of test cases	Execution time (ms)	# of test cases	Execution time (ms)
1	Main	Binding – Value Change	34	13064	34 (0%)	13255 (−1%)
2	Server	Filter – Addition	34	13762	23 (32%)	9015 (34%)
3	SourceNode	Binding – Deletion	34	12982	17 (50%)	6804 (48%)
4	SinkNode	In Pattern Element – Class Change	34	11890	7 (79%)	2937 (75%)
5	NonServerNode	Binding – Deletion	34	13375	13 (62%)	5442 (59%)
6	ThinkDevice	Matched Rule – Deletion	34	12256	13 (62%)	5157 (58%)
7	Arc	Out Pattern Element – Addition	34	13446	23 (32%)	8982 (33%)
8	ServiceRequest	Binding – Deletion	34	13277	22 (35%)	8263 (38%)
9	WorkloadRouting	Binding – Value Change	34	12898	22 (35%)	8474 (34%)
10	Workload	Out Pattern Element – Addition and Deletion	34	12013	30 (12%)	10462 (13%)

Table 4: Results for the BibTeX2DocBook model transformation example

ID	Rule affected	Type of change	Rerun all		Rerun selected only	
			# of tests cases	Execution time (ms)	# of tests cases	Execution time (ms)
1	Author	Binding – Value Change	25	7038	24 (4%)	6710 (4.66%)
2	TitledEntry_Title_NoArticle	Binding – Deletion	25	6765	9 (64%)	2813 (58.42%)
3	Article_Title_Journal	Binding – Deletion	25	7007	19 (24%)	5269 (24.80%)
4	Article_Title_Journal	Matched Rule – Deletion	25	6737	19 (24%)	5157 (20.26%)
5	Main	Out Pattern Element – Addition	25	6690	25 (0%)	6690 (0%)
6	UntitledEntry	Binding – Deletion	25	5844	25 (0%)	5844 (0%)
7	TitledEntry_Title_NoArticle	Binding – Value Change	25	6648	9 (64%)	2666 (59.90%)

The results in Table 3 show good reduction in terms of the number of selected cases and the execution time. The exception is in the first rule, *Main*, where there is no reduction in the number of selected test cases. This is because this rule is executed by the transformation on every input model, and hence any input test model will be automatically selected by our tool. The reductions reached more than 75% in Experiment 4. It is also important to mention that in each experiment at least one of the test cases failed the test (as reported by the EUnit tool) in both the Rerun all and Rerun Selected-only cases. This indicates that the fault detection capability was not diminished when rerunning the selected test cases. At the same time, a good savings in terms of the testing execution time were materialized.

Table 4 presents the results concerning the *BibTeX2DocBook* transformation. Note that this transformation consists of nine rules. In our experiments, we randomly selected a total of 25 test cases out of the 100 test cases available in the test suite taken from a previous work on spectrum-based fault detection in model transformations [4]. The test cases in the test suite were semi-automatically created using model generation scripts. We obtained the test suite from [30]. These 25 test cases represent the full test suite that is input to our tool for test case selection. The table shows good reductions in the size of the selected test cases and the execution time as well. However, there is little or no reduction in cases 1, 5, and 6. For case 1, the *Author* rule applies on *Author* elements which appear in almost all of the test cases. Hence, our tool selects all of these test cases resulting in a very small reduction in the size of the selected test cases. For cases 5 and 6, the *Main* and *UntitledEntry* rules apply on source meta-classes whose instances appear in all of the test cases. For example, the *Main* rule which applies on *BibTeXFile* elements is executed on every input model since every *BibTex* model has a root element of type *BibTeXFile.* Also, the *UntitledEntry* rule applies on elements of the *BibTeXEntry* source meta-class which is a superclass for many of the source meta-classes. Note that in each of the cases in Table 4 at least one of the test cases failed when running the selected test cases.

Table 5 presents the results concerning the *CPL2SPL* transformation. This transformation consists of 19 rules. Similar to the BibTex2DocBook transformation case, we randomly selected a total of 35 test cases from the test suite taken from [30]. In all cases, the table shows good reductions in the size of the selected test cases and the execution time as well. In addition, in all cases, rerunning the test cases selected by our tool resulted in at least one test case failure, indicating that the test suites selected by our tool were able to discover the mutants.

Table 6 shows the execution time results for the four case studies. For each case study, the table shows the time it took our tool to generate the test case model. The table shows the confidence intervals on a 95% confidence level. These intervals were obtained using the one-sample t-test [31] which is valid to be used in our case since the same size is small (30 execution time

Table 5: Results for the CPL2SPL model transformation example

ID	Rule affected	Type of change	Rerun all		Rerun selected only	
			# of test cases	Execution time (ms)	# of test cases	Execution time (ms)
1	NoAnswer2SelectCase	Binding – Value Change	35	12656	2 (94.29%)	1118 (91.17%)
2	Busy2SelectCase	Filter – Addition	35	14130	3 (91.43%)	1419 (89.96%)
3	NoAnswer2SelectCase	Binding – Deletion	35	14459	2 (94.29%)	1131 (92.18%)
4	StringSwitch2SelectStat	In Pattern Element – Class Change	35	13224	11 (68.57%)	4458 (66.29%)
5	SwitchedAddress2SelectCase	Binding – Deletion	35	13039	1 (97.14%)	739 (94.33%)
6	Outgoing2Method	Matched Rule – Deletion	35	13191	2 (94.29%)	1087 (91.76%)
7	SubAction2Function	Out Pattern Element – Addition	35	13529	4 (88.57%)	1858 (86.27%)
8	StringSwitch2SelectStat	Binding – Deletion	35	14792	11 (68.57%)	5052 (65.85%)
9	Incoming2Method	Binding – Value Change	35	12716	4 (88.57%)	1889 (85.14%)
10	Proxy2Select	Out Pattern Element – Addition and Deletion	35	14064	5 (85.71%)	1251 (91.10%)

Table 6: The execution times for test case model generation

Case Study	Mean (ms)	The 95% Confidence Interval
OO2DB2	29.73	(25.46, 34.00)
QN2QPN	62.17	(54.62, 69.72)
BibTeX2DocBook	54.63	(48.73, 60.54)
CPL2SPL	75.13	(65.25, 85.01)

results in each case study) and the normality check provides good support for the assumption that the population is normally distributed. It is of interest to note that these observed execution times are very small compared with the test case execution time results noted in the previous tables in this section. In addition, our tool was able to automatically generate the test case models with no considerable cost on part of the tester.

4.3. Threats to validity

There are four basic types of validity threats that can affect the validity of the conclusions of our experiments [32]:

1. *Conclusion Validity:* Threats to the conclusion validity are concerned with factors that affect the ability to draw the correct conclusions based on the observed data. To address this threat, we used confidence intervals based on a 95% confidence level. These intervals were obtained using the one-sample t-test after passing the normality check. Also, we used the wallclock times reported by the EUnit tool for all test case suite execution time results reported in the paper. We have also used a variety of mutation operators in the experiments.
2. *Construct Validity:* This validity is concerned with the relationship between theory and observation. To address this threat, we used standard performance measures and metrics, including rule coverage and execution time. These metrics have been used in other similar work on regression testing for model transformations, such as [3, 4].
3. *Internal Validity:* This validity is concerned with establishing a causal relationship be-

tween the treatment (in this case, the application of our approach) and the results of our evaluation. Threats to this validity include any disturbing factor that might influence the results. As our experiments demonstrated, the execution time of a test case suite is directly proportional to its size. Therefore, the observed reductions in execution time can be justified by the selection of a smaller number of test cases by applying our approach. In addition, in every experiment we made a mutation to a single rule only. Every mutation applied one of the mutation operators proposed in literature [29, 33]. If a rule is mutated, then this would affect any input test model that includes an element on which the rule is applied. Hence, our tool would automatically select the corresponding test case for rerun. Hence, the fault detection capability of the original test case suite is not compromised by the suite of the test cases selected by our tool.

4. *External Validity:* This validity is concerned with generalization. The evaluation applied our approach on two model transformations written in two languages: ETL and ATL. We applied different types of mutation operators, and the test models were of different sizes. The test cases were created manually to achieve a variety of coverage criteria. Yet, we cannot make a firm conclusion that our results can be generalized for all model transformations. More experiments are needed in the future to confirm our findings on a wider scope of model transformations, including different languages, coverage criteria, types of mutations and faults, and input test models.

5. Related work

Alkhazi et al. [3] propose an approach for test case selection for model transformation based on multiobjective search. The approach enables a tester to find the best tradeoff between two conflicting objectives: maximizing rule coverage and minimizing the execution time of the selected test cases. A multiobjective algorithm (NSGA-II) is used to find the Pareto-optimal solutions for this problem. The approach was validated using different transformation programs written in ATL. In comparison to their approach, our approach aims to be a safe test case selection technique. A test selection technique is said to be safe if it selects all modification-revealing tests [7]. Our proposed technique will select any test case for rerun when its input model contains an element that could be affected by a change in one of the model transformation rules. Our approach does not consider the trade-off between rule coverage and execution time. However, our experiments show good saving in test execution time when applying our approach while not compromising the fault detection capability of the full test case suite.

In [20], model transformation traceability is used to enhance the automation of qualifying and improving a set of test models in the mutation analysis of model transformation. The approach relies on a representation of different mutation operators and a traceability mechanism to establish links between the input and output models of each transformation. Patterns are used to identify cases where an input test model lets a given mutant alive. Subsequently, heuristics provide recommendations to generate new test models that are able to kill the mutant. Several aspects of the approach are independent from the transformation language being used, including the traceability and the mutation operator representation. Our approach focuses on regression testing of model transformation rather than mutation analysis. Hence, our approach uses a traceability model linking test cases to the rules in a given model transformation. When a rule is changed, the traceability model can be used to identify the test cases that need to be rerun. Our traceability model differs from that in [20] which maintains links between elements of the input and output models for each mutant. This is a more detailed model suitable for mutation analysis of model transformation.

A multiobjective optimization algorithm is employed in [9] to generate test models for the regression testing of model transformations. The proposed approach assumes that the changes occur in the input metamodel only. In this case, a test model may become invalid when it does not conform to the updated input metamodel. The optimization algorithm has three objectives that define the characteristics of a good solution: maximize coverage of the updated metamodel, minimize the number of input model elements that do not conform to the updated metamodel, and minimize the number of refactorings used to refactor the existing test models. In our approach, we assume that the input and output metamodels are fixed and only the model transformation program may change. While the approach in [9] is useful to determine and update the test models that become invalid due to a change in the input metamodel, our approach utilizes traceability links to determine the test cases that need to be rerun due to an update in the model transformation program.

Honfi et al. [34] presented a method on how model-based regression testing can be achieved in the context of autonomous robots. The method uses optimization for selecting the minimal subset of tests that have a maximum test coverage of the changed components. Although the method is presented in the context of robots, it is applicable to other domains which employ Model-Driven Development (MDD) paradigm. In MDD, models are adopted as the main development artifacts. These models are commonly created using domain specific languages (DSLs). When a model is changed, this can impact the system functions and properties and hence the influenced parts of the system need to be retested. A prototype tool that implements the method using the Eclipse framework [35] and its modeling platform EMF [11] is presented. The tool supports model checkpointing and automatic change detection. Our approach is similar to [34] in terms of employing a metamodel to represent the relationship between the test items and the test cases. However, our approach is focused on applying regression testing to a model transformation. We consider the issues specific to a model transformation which can be more useful to a tester of model transformation programs.

A survey of model transformation testing approaches is provided in [6] and [36]. Gonzálex and Cabot [37] developed a tool, called ATLTest, to generate test input models for ATL transformations. The tool applies a white-box approach for model transformation testing. In the work of Fleurey et al. [28], model fragments are used as a test adequacy criterion which forms the basis of the black-box approach proposed by the authors. Other approaches rely on formal methods to verify the transformation and its associated properties [38–40]. The problem with these approaches is their computational complexity which becomes cumbersome with the scale of the model transformation program.

Zech et al. proposed a model-based regression testing method based on OCL [41]. The method derives test cases for a given system under test (SUT) based on the availability of a class diagram that captures its system design. The approach is based on a Model Versioning and Evolution (MoVE) framework and uses UML testing profile (UTP) to model test cases. The method calculates a *delta* from a base model (the initial development model) and the working model (the current development model). The resulting change set (*delta*) then contains the differences between the two versions of the model. There are three main differences between the method by Zech et al. and our method. First, our method presents a regression testing approach for model transformation while the method by Zech et al. presents a regression testing approach of a software system using its design model. Second, Our approach requires a metamodel of the test case set while the method by Zech et al. does not need any metamodel, but rather uses the facilities provided in an MoVE framework. The last difference is that our approach exploits traceability links between models while the method by Zech et al. does not use any traceability link. A similar model-based regression testing for software systems that works with sequence diagrams is presented in [42]. In [43], Al-Refai et al. provide a framework for model-based regression test selection supporting modifications to UML class and activity diagrams. Using mutation testing, their experimental results demonstrate that the selected test cases achieve the same fault detection capability as that achieved by the complete set of test sets.

The work of Troya et al. [44] proposes an approach for automatically inferring metamorphic relations for testing ATL model transformations. The inferred metamorphic relations can be used to detect faults in model transformations in several application scenarios including regression testing. The metamorphic relations are inferred by exploiting the trace model produced when a transformation is executed. The trace model is composed of traces. When a rule is executed, a trace can be automatically obtained by using tools such as TraceAdder [45]. For the executed rule, a trace links the name of the rule with the elements instantiating the classes of its source metamodel and the new elements that are created by the rule and hence instantiate classes in the target metamodel. The authors used mutation-based testing, similar to what is done in this paper, to evaluate how effective the approach is in detecting faults in regression testing.

PIn [4], Troya et al. presented an approach for debugging and fault localization for model transformations by applying spectrum-based fault localization techniques. The approach is based on the use of a trace model that can be obtained when the test cases are executed. When a test case fails, the approach ranks the transformation rules according to how much they are suspected of having the fault causing its failure. Mutation-based testing is applied to validate the effectiveness of their approach in fault localization.

In [46], Naslavsky et al. presented an approach for selective regression testing that is model-based. In this approach, test cases are selected for retesting based on modifications to the model, rather than to the source code. The approach uses traceability links between model elements and test cases that traverse such elements. As a modeling perspective, the approach adopts UML class and sequence diagrams. While their presented approach is designed for testing general software programs, our work is designed for testing model-to-model transformation programs. In our approach, we exploit the traceability links between test cases and model transformation program elements such as rules.

Our approach for regression testing of model transformation utilizes traceability links between test cases and test artifacts. Traceability has been studied by researchers in the areas of requirements engineering and model-driven development (MDD) for a long time. Winkler and Pilgrim [47] provide an extensive literature survey on traceability in both areas.

Due to the continuous increase in the size and complexity of software, model-based engineering is gaining a lot of interest from the industry and research community [48]. Model-based testing, which is an important part of model-based engineering tests the consistency of information and behavior of source models by applying transformation mechanisms. The automatic nature of model-based testing makes it a more adoptable approach for detecting software defects fast and effectively [49, 50]. Regression testing is a quite a tedious work which is repeated with every sizeable refactoring of the model. Model-based approaches can make this process easier by automating, managing, and documenting efficiently. Yoo *et.al* [8] suggest, based on their extensive survey, that model-based regression testing techniques increase the effectiveness and scalability of the overall testing of the system. Model-based regression testing approaches have several advantages over code-based testing [51, 52]. The effort could be estimated at a very early stage and the tools are largely language independent. The models are mostly abstract, which makes the size of the testable data considerably smaller than the code.

The majority of model-based regression testing approaches exploit UML models for developing test suites for regression testing. UML class models have been used for this purpose along with state machines, sequence diagram and activity diagrams [51–53]. Farooq et al. [52] use state machines to represent changes in the tested parts of the system. The method is also automated using an Eclipse-based tool. Briand et al. [51, 54] presented a sequence diagram-based technique to classify and analyze regression test suites. The approach is complemented with a tool to evaluate the presented models. Finite State Machines (FSMs) have also been used to generate regression test suites. Chen et al. [55] have deployed extended FSMs to model the effects of changes and generate test suites for the modified parts of the system. Korel et al [56] have used a similar approach and have exploited extended FSMs to reduce the size of an existing regression test suite. In both approaches, modifications (updates, additions and deletions) are represented as transitions of extended FSMs. In one study, Vaysburg et al. [57] used extended FSMs for the dependency analysis of the system that represent various interactions between components for the regression test selection process. In another study, Almasri et al [58] conducted an impact analysis using extended FSMs to identify the parts of the system that are affected the most. Feldere et al. [59, 60] also used FSMs to represent all model elements of the system. They proposed a process to identify model elements which trigger change events. The identified models are then changed to make the system consistent.

6. Conclusion and future work

In this paper, we have presented a framework for the regression test selection for model transformation programs. The framework exploits the traceability links in a test case model. In the evaluation, we applied the framework to several model transformation examples and showed the effectiveness and time-saving benefit of the framework. The experiments were performed in the context of the Epsilon platform of integrated tools and languages for model management. We also presented a tool that can automatically build the test case model and thus facilitate the implementation of our proposed framework.

Following this work, there are several avenues for future research. First, the proposed framework needs to be integrated with existing model transformation tools and technologies. For instance, when a designer makes a change to the model transformation code, the relevant test cases identified by our framework can be automatically rerun by the integrated tools. In this case, the designer does not need to manually rerun the test cases. Second, it is recommended to apply the proposed framework to industrial case studies involving larger models and more complex model transformation logic.

7. Acknowledgement

This work has not received any funding.

References

[1] A.R. da Silva, “Model-driven engineering: A survey supported by the unified conceptual mode,” *Computer Languages, Systems & Structures*, Vol. 43, 2015, pp. 139–155.

[2] P. Mohagheghi and V. Dehlen, “Where is the proof? – A review of experiences from applying mde in industry,” in *Proceedings of the European Conference on Model Driven Architecture – Foundations and Applications*, 2008, pp. 432–443.

[3] B. Alkhazi, C. Abid, M. Kessentini, D. Leroy, and M. Wimmer, “Multi-criteria test cases selection for model transformations,” *Automated Software Engineering*, Vol. 27, No. 1, 2020, pp. 91–118.

[4] J. Troya, S. Segura, J.A. Parejo, and A. Ruiz-Cortés, “Spectrum-based fault localization in model transformations,” *ACM Transactions on Software Engineering and Methodology*, Vol. 27, No. 3, 2018.

[5] D. Calegari and N. Szasz, “Verification of model transformations: A survey of the state-of-the-art,” *Electronic Notes in Theoretical Computer Science*, Vol. 292, 2013, pp. 5–25.

[6] G.M.K. Selim, J.R. Cordy, and J. Dingel, “Model transformation testing: The state of the art,” in *Proceedings of the First Workshop on the Analysis of Model Transformations*, 2012, pp. 21–26.

[7] G. Rothermel and M.J. Harrold, “Analyzing regression test selection techniques,” *IEEE Transactions on Software Engineering*, Vol. 22, No. 8, 1996, pp. 529–551.

[8] S. Yoo and M. Harman, “Regression testing minimization, selection and prioritization: A survey,” *Software Testing, Verification and Reliability*, Vol. 22, No. 2, 2012, pp. 67–120.

[9] J. Shelburg, M. Kessentini, and D.R. Tauritz, “Regression testing for model transformations: A multi-objective approach,” in *Proceedings of the International Symposium on Search Based Software Engineering*, 2013, pp. 209–223.

[10] E. Seidewitz, “What models mean,” *IEEE Software*, Vol. 20, No. 5, 2003, pp. 26–32.

[11] *Eclipse Modeling Framework (EMF)*, Eclipse Foundation. [Online]. https://www.eclipse.org/modeling/emf/ [Accessed December 2020].

[12] K. Czarnecki and S. Helsen, "Feature-based survey of model transformation approaches," *IBM Systems Journal*, Vol. 45, No. 3, 2006, pp. 621–645.

[13] *Eclipse Epsilon*, Eclipse Foundation. [Online]. https://www.eclipse.org/epsilon/ [Accessed December 2020].

[14] D. Kolovos, L. Rose, A. García-Domínguez, and R. Paige, *The epsilon book*. Eclipse Foundation., 2018. [Online]. https://www.eclipse.org/epsilon/doc/book/

[15] *ATL – A model transformation technology*, Eclipse Foundation. [Online]. https://www.eclipse.org/atl/ [Accessed December 2020].

[16] *The Epsilon Unit Testing Framework*, Eclipse Foundation. [Online]. https://www.eclipse.org/epsilon/doc/eunit/ [Accessed December 2020].

[17] A. García-Domínguez, D.S. Kolovos, L.M. Rose, R.F. Paige, and I. Medina-Bulo, "EUnit: A unit testing framework for model management tasks," in *Proceedings of the International Conference on Model Driven Engineering Languages and Systems*, 2011, pp. 395–409.

[18] *EMF Compare*, Eclipse Foundation. [Online]. https://www.eclipse.org/emf/compare/ [Accessed December 2020].

[19] *Epsilon – Examples.* [Online]. https://www.eclipse.org/epsilon/examples/ [Accessed December 2020].

[20] V. Aranega, J. Mottu, A. Etien, T. Degueule, B. Baudry, and J. Dekeyser, "Towards an automation of the mutation analysis dedicated to model transformation," *Software Testing, Verification and Reliability*, Vol. 25, No. 5-7, 2015, pp. 653–683.

[21] F. Fleurey, J. Steel, and B. Baudry, "Validation in model-driven engineering: Testing model transformations," in *Proceedings of the First International Workshop on Model, Design and Validation*, 2004, pp. 29–40.

[22] S. Sen, B. Baudry, and J. Mottu, "On combining multi-formalism knowledge to select models for model transformation testing," in *Proceedings of the First International Conference on Software Testing, Verification, and Validation*, 2008, pp. 328–337.

[23] *Object Constraint Language*, Object Management Group, 2014. [Online]. http://www.omg.org/spec/OCL/2.4 [Accessed December 2020].

[24] I. Al-Azzoni, "ATL transformation of queueing networks to queueing Petri nets," in *Proceedings of the International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, 2017, pp. 261–268.

[25] *ATL Transformation Example: BibTeXML to DocBook*, 2005. [Online]. https://www.eclipse.org/atl/atlTransformations/BibTeXML2DocBook/ExampleBibTeXML2DocBook[v00.01].pdf [Accessed December 2020].

[26] F. Jouault, J. Bézivin, C. Consel, I. Kurtev, and F. Latry, "Building DSLs with AMMA/ATL, a case study on SPL and CPL telephony languages," in *Proceedings of the ECOOP Workshop on Domain-Specific Program Development*, 2006.

[27] *ATL Transformations Zoo*, Eclipse Foundation. [Online]. https://www.eclipse.org/atl/atlTransformations/ [Accessed December 2020].

[28] F. Fleurey, B. Baudry, P. Muller, and Y.L. Traon, "Qualifying input test data for model transformations," *Software and System Modeling*, Vol. 8, No. 2, 2009, pp. 185–203.

[29] J. Troya, A. Bergmayr, L. Burgueño, and M. Wimmer, "Towards systematic mutations for and with ATL model transformations," in *Proceedings of International Conference on Software Testing, Verification and Validation Workshops*, 2015, pp. 1–10.

[30] J. Troya, *Implementation of the Spectrum-Based Fault Localization in Model Transformations*, 2018. [Online]. https://github.com/javitroya/SBFL_MT [Accessed December 2020].

[31] D.C. Montgomery and G.C. Runger, *Applied Statistics and Probability for Engineers, 6th Edition.* John Wiley and Sons, 2013.

[32] C. Wohlin, P. Runeson, M. Hst, M.C. Ohlsson, B. Regnell, and A. Wessln, *Experimentation in Software Engineering.* Springer Publishing Company, 2012.

[33] E. Guerra, J. Sánchez Cuadrado, and J. de Lara, "Towards effective mutation testing for ATL," in *Proceedings of the International Conference on Model Driven Engineering Languages and Systems*, 2019, pp. 78–88.

[34] D. Honfi, G. Molnár, Z. Micskei, and I. Majzik, "Model-based regression testing of autonomous robots," in *Proceedings of the International System Design Language Forum*, 2017, pp. 119–135.

[35] *Eclipse*, Eclipse Foundation. [Online]. https://www.eclipse.org/ [Accessed December 2020].

[36] L.A. Rahim and J. Whittle, "A survey of approaches for verifying model transformations," *Software and Systems Modeling*, Vol. 14, No. 2, 2015, pp. 1003–1028.

[37] C.A. González and J. Cabot, "ATLTest: A white--box test generation approach for ATL transfor-

mations," in *Proceedings of the International Conference on Model Driven Engineering Languages and Systems*, 2012, pp. 449–464.

[38] J. Cabot, R. Clarisó, E. Guerra, and J. de Lara, "Verification and validation of declarative model--to-model transformations through invariants," *Journal of Systems and Software*, Vol. 83, No. 2, 2010, pp. 283–302.

[39] K. Ehrig, J.M. Küster, and G. Taentzer, "Generating instance models from meta models," *Software and Systems Modeling*, Vol. 8, No. 4, 2009, pp. 479–500.

[40] J. Troya and A. Vallecillo, "Towards a rewriting logic semantics for ATL," in *Proceedings of the International Conference on Theory and Practice of Model Transformations*, 2010, pp. 230–244.

[41] P. Zech, P. Kalb, M. Felderer, C. Atkinson, and R. Breu, "Model-based regression testing by OCL," *International Journal on Software Tools for Technology Transfer*, Vol. 19, No. 1, 2017, pp. 115–131.

[42] L. Naslavsky, H. Ziv, and D.J. Richardson, "A model-based regression test selection technique," in *Proceedings of the IEEE International Conference on Software Maintenance*, 2009, pp. 515–518.

[43] M. Al-Refai, S. Ghosh, and W. Cazzola, "Supporting inheritance hierarchy changes in model-based regression test selection," *Software and Systems Modeling*, Vol. 18, No. 2, 2019, pp. 937–958.

[44] J. Troya, S. Segura, and A. Ruiz-Cortés, "Automated inference of likely metamorphic relations for model transformations," *The Journal of Systems and Software*, Vol. 136, 2018, pp. 188–208.

[45] F. Jouault, "Loosely coupled traceability for ATL," in *Proceedings of the European Conference on Model Driven Architecture Workshop on Traceability*, 2005, pp. 29–37.

[46] L. Naslavsky, H. Ziv, and D. Richardson, "MbSRT2: Model-based selective regression testing with traceability," in *Proceedings of the International Conference on Software Testing, Verification and Validation*, 2010, pp. 89–98.

[47] S. Winkler and J. von Pilgrim, "A survey of traceability in requirements engineering and model-driven development," *Software and Systems Modeling*, Vol. 9, No. 4, 2010, pp. 529–565.

[48] B. Legeard, "Model-based testing: Next generation functional software testing," in *Practical Software Testing: Tool Automation and Human Factors*, M. Harman, H. Muccini, W. Schulte, and T. Xie, Eds. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Germany, 2010.

[49] A.C. Dias Neto, R. Subramanyan, M. Vieira, and G.H. Travassos, "A survey on model-based testing approaches: A systematic review," in *Proceedings of the International Workshop on Empirical Assessment of Software Engineering Languages and Technologies*, 2007, pp. 31–36.

[50] M. Utting and B. Legeard, *Practical Model--Based Testing – A Tools Approach*. Morgan Kaufmann, 2007.

[51] L.C. Briand, Y. Labiche, and S. He, "Automating regression test selection based on UML designs," *Information and Software Technology*, Vol. 51, No. 1, 2009, pp. 16–30.

[52] Q. Farooq, M.Z.Z. Iqbal, Z.I. Malik, and A. Nadeem, "An approach for selective state machine based regression testing," in *Proceedings of the International Workshop on Advances in Model-based Testing*, 2007, pp. 44–52.

[53] Y. Chen, R.L. Probert, and D.P. Sims, "Specification-based regression test selection with risk analysis," in *Proceedings of the Conference of the Centre for Advanced Studies on Collaborative Research*, 2002.

[54] L.C. Briand, Y. Labiche, and G. Soccar, "Automating impact analysis and regression test selection based on UML designs," in *Proceedings of the International Conference on Software Maintenance*, 2002, pp. 252–261.

[55] Y. Chen, R.L. Probert, and H. Ural, "Regression test suite reduction using extended dependence analysis," in *Proceedings of the International Workshop on Software Quality Assurance*, 2007, pp. 62–69.

[56] B. Korel, L.H. Tahat, and B. Vaysburg, "Model based regression test reduction using dependence analysis," in *Proceedings of the International Conference on Software Maintenance*, 2002.

[57] B. Vaysburg, L.H. Tahat, and B. Korel, "Dependence analysis in reduction of requirement based test suites," in *Proceedings of the International Symposium on Software Testing and Analysis*, 2002, pp. 107–111.

[58] N. Almasri, L. Tahat, and B. Korel, "Toward automatically quantifying the impact of a change in systems," *Software Quality Journal*, Vol. 25, No. 3, 2017, pp. 601–640.

[59] M. Felderer, B. Agreiter, and R. Breu, "Evolution of security requirements tests for service-centric systems," in *Engineering Secure Software and Systems*, Ú. Erlingsson, R. Wieringa, and N. Zannone, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 181–194.

[60] M. Felderer, B. Agreiter, and R. Breu, "Managing evolution of service centric systems by test models," in *Proceedings of the Tenth IASTED International Conference on Software Engineering*, 2011, pp. 72–80.

Appendix: ETL OO2DB code

```
pre {
  "Running ETL".println();
  var db : new DB!Database;
}

// Transforms a class into a table and
// a primary key column
 rule Class2Table
  transform c : OO!Class
  to t : DB!Table, pk : DB!Column {

  t.name = c.name;
  t.database = db;

  // Fill the details of the primary key
  // of the table
  pk.name = t.primaryKeyName();
  pk.type = "INT";
  t.columns.add(pk);
  t.primaryKeys.add(pk);

  // If the class extends some other class
  // create a foreign key pointing towards
  // the primary key of the parent class
  if (c.`extends`.isDefined()){

    var fk : new DB!ForeignKey;
    var childFkCol : new DB!Column;
    var parentFkCol : DB!Column;
    var parentTable : DB!Table;

    parentTable ::= c.`extends`;
    parentFkCol = parentTable.primaryKeys.first();

    childFkCol.name = parentFkCol.name;
    childFkCol.type = "INT";
    childFkCol.table = t;

    fk.database = db;
    fk.parent = parentFkCol;
    fk.child = childFkCol;
    fk.name = c.name + "Extends" + c.`extends`.name;
  }
}

// Transforms a single−valued attribute
// to a column
 rule SingleValuedAttribute2Column
  transform a : OO!Attribute
  to c : DB!Column {

  guard : not a.isMany

  c.name = a.name;
  c.table ::= a.owner;
  c.type = a.type.name.toDbType();
}

// Transforms a multi−valued attribute
// to a table where its values are stored
// and a foreign key
rule MultiValuedAttribute2Table
  transform a : OO!Attribute
  to t : DB!Table, pkCol : DB!Column, valueCol :
  DB!Column, fkCol : DB!Column,
  fk : DB!ForeignKey {

  guard : a.isMany

  // The table that stores the values
  // has an "id" column and a "value" column
  t.name = a.valuesTableName();
  t.database = db;

  pkCol.name = "id";
  pkCol.table = t;
  pkCol.type = "INT";
  valueCol.name = "value";
  valueCol.table = t;
  valueCol.type = a.type.name.toDbType();

  // Another column is added into the table
  // to link with the "id" column of the
  // values table
  fkCol.name = a.name + "Id";
  fkCol.table ::= a.owner;
  fkCol.type = "INT";

  // The foreign key that connects
  // the two columns is defined
  fk.parent = pkCol;
  fk.child = fkCol;
  fk.database = db;
}

// Transforms a referecne into a foreign key
rule Reference2ForeignKey
  transform r : OO!Reference
  to fk : DB!ForeignKey, fkCol : DB!Column {

  fkCol.table ::= r.type;
  fkCol.name = r.name + "Id";
  fkCol.type = "INT";
  fk.database = db;
  fk.parent = r.owner.equivalent().primaryKeys.first();
  fk.child = fkCol;
  fk.name = r.name;

}

operation DB!Table primaryKeyName() : String {
  return self.name.firstToLowerCase() + "Id";
}

operation OO!Attribute valuesTableName() : String {
  return self.owner.name + "_" +
   self.name.firstToUpperCase() + "Values";
}

operation Any toDbType() : String {
  var mapping : OO2DB!TypeMapping;
  mapping = OO2DB!TypeMapping.allInstances().
    select(tm|tm.source = self).first;
  if (not mapping.isDefined()){
    ("Cannot find DB type for OO type " + self +
      ". Setting the default.").println();
    return OO2DB!TypeMap.allInstances().first().
    `default`.target;
  }
  else {
    return mapping.target;
  }
}
```

e-Informatica Software Engineering Journal, Volume 15, Issue 1, 2021, pages: 85–114, DOI 10.37190/e-Inf210105

Mining Non-Functional Requirements using Machine Learning Techniques

Rajni Jindal*, Ruchika Malhotra**, Abha Jain***, Ankita Bansal****

*Dept. of Computer Science Engineering, Delhi Technological University, India
**Dept. of Software Engineering, Delhi Technological University, India
***Dept. of Computer Science, Delhi University, India
****Division of Information Technology, Netaji Subhas University of Technology, India

rajnijindal@dce.ac.in, ruchikamalhotra2004@yahoo.com, abhajain.src@gmail.com, ankita.bansal06@gmail.com

Abstract

Background: Non-Functional Requirements (NFR) have a direct impact on the architecture of the system, thus it is essential to identify NFRs in the initial phases of software development. **Aim**: The work is based on extraction of relevant keywords from NFR descriptions by employing text mining steps and thereafter classifying these descriptions into one of the nine types of NFRs. **Method**: For each NFR type, keywords are extracted from a set of pre-categorized specifications using Information-Gain measure. Then models using 8 Machine Learning (ML) techniques are developed for classification of NFR descriptions. A set of 15 projects (containing 326 NFR descriptions) developed by MS students at DePaul University are used to evaluate the models.

Results: The study analyzes the performance of ML models in terms of classification and misclassification rate to determine the best model for predicting each type NFR descriptions. The Naïve Bayes model has performed best in predicting "maintainability" and "availability" type of NFRs.

Conclusion: The NFR descriptions should be analyzed and mapped into their corresponding NFR types during the initial phases. The authors conducted cost benefit analysis to appreciate the advantage of using the proposed models.

Keywords: requirement engineering, text mining, non-functional requirements, machine learning, receiver operating characteristics

1. Introduction

Non-Functional Requirements (NFRs) are the basic quality constraints which specify the operation of a system [1, 2]. NFRs go hand-in-hand with the Functional Requirements (FRs) and are highly essential to ensure the development of an efficient and a reliable software system that meets the customers needs and fulfills their expectations [3]. Set of NFRs need to be correctly identified in the initial phases of Software Development Lifecycle (SDLC) process as they play a crucial role in the architecture and design of the system which in turn affects the quality of the system [4]. NFRs form the basis for architects to create the technical architecture of the system. This architecture of the system then acts a platform in which the functionality of the system is delivered [5]. Unfortunately, NFRs are discovered in the later phases of software development. This may be due the reason that some requirement engineers tend to overlook NFRs failing to realize their importance and thereby assume them to be implicitly understood [6]. They do not elicit NFRs in the Software Requirement Specification (SRS) document as clearly as they state FRs, but rather state NFRs in a very adhoc and random fashion due to which the final SRS document is organized by functionality with non-functional requirements scattered throughout the document

Submitted: 27 February 2020; Revised: 08 January 2021; Accepted: 19 February 2021; Available online: 28 June 2021

[7, 8]. Failure to identify and analyze NFRs in the early phases can result in unclassified, incomplete or conflicting NFRs, requiring costly rework in later stages of the software development [5]. The work in this paper focuses on mining the descriptions of NFRs which are stated throughout the requirement specification document in an adhoc and random fashion and thereafter classify them into one of the types of NFRs using a suitable Machine Learning (ML) technique. In this work, nine types of NFRs have been considered viz. Availability (A), Look-and-Feel (LF), Legal (L), Maintainability (MN), Operational (O), Performance (PE), Scalability (SC), Security (SE) and Usability (US). The work is based on extraction of relevant keywords from NFR descriptions by employing a series of text mining steps. Firstly, the NFR descriptions are pre-processed to remove irrelevant words from the descriptions. These are the stop words like prepositions, articles, conjunctions, verbs, pronouns, nouns, adjectives and adverbs whose presence will not have any impact on the performance of the prediction models but will rather degrade their performance. Once pre-processing is done, we need to find the words which are relevant in describing the NFR descriptions. One of the methods is to extract the relevant words using fixed set of pre-defined keywords available in the catalogues. These catalogues contain a standardized set of keywords specific to different types of NFRs. However, the main problem of using this approach (standard keyword method) is the difficulty of finding accepted and standardized catalogues for all the types of NFR specified in the datasets [9]. Even for the NFR types whose catalogues were available, it was observed that the keywords specified in the catalogue for that particular NFR type could not classify the NFR descriptions pertaining to that NFR type accurately. This was observed by the authors Cleland-Huang et al. [9] who concluded that classification of NFRs based the keywords extracted from the catalogues result in unclassified, incomplete or conflicting NFRs.

In order to address this problem, the authors in this paper are using a training set to discover a set of keywords specific to each type of NFR. Since the keywords in our study are extracted from pre-categorized requirement specifications, therefore problem of finding accepted and standardized catalogues for all the NFR types is removed.

These keywords are the weighted indicator terms (specific to each NFR type) which are extracted using Information-Gain (IG) measure. IG measure extracts those keywords (also known as indicator terms) from the specifications that most accurately identify the target concept (NFR type). For example, terms such as "authenticate" and "access" represent strong indicator terms for security NFRs as they most accurately define security requirements as compared to other types of requirements. IG measure works by associating a weight to each of the word obtained after pre-processing and then the top-N scoring words are selected as the indicator terms. In our work, the value of N is considered as 10 as the paper by Cleland-Huang et al. [9] showed that good results were achieved when top-10 words were considered as compared to the results achieved when top-5 words or all the words were considered together. Once the indicator terms for each NFR type are identified, prediction models are developed for the classification of future NFR descriptions whose NFR type is not known. The authors in this paper have used in total eight ML algorithms viz. J48 decision tree (J48), Random Forest (RF), Logitboost (LB), Adaboost (AB), Multi-Layer Perceptron (MLP), Radial Basis Function network (RBF), Bagging, Naïve Bayes (NB) for predicting each type of NFR. The usage of large number of ML algorithms allows the authors to perform exhaustive comparison and evaluation. The authors concluded the best ML model for each NFR type which can be used by software practitioners and researchers in classifying an unknown NFR description into its respective type. In addition to this, the authors have also conducted cost benefit analysis to understand and appreciate the advantage of using the proposed models in contrast to not using the models in terms of cost incurred in both the cases.

Thus, there are three main goals of this study which we thrive to achieve:

1. To apply text mining steps to identify indicator terms As discussed, top-10 indicator terms for each NFR type are identified by following

a series of text mining steps. This begins by applying pre-processing steps to remove irrelevant words from the specifications and thereafter applying IG measure to retrieve significant words. IG measure works by associating a weight to each of the word obtained after pre-processing and then the top-N scoring words are selected as the indicator terms specific to each NFR type. These indicator terms serve as independent variables used for model prediction.

2. To develop machine learning models for each type of NFR Corresponding to each NFR type, a separate dataset was considered with top-10 indicator terms of that particular NFR type as independent variables and a binary dependent variable with the value of 1 (if NFR description belongs to that particular NFR type) or 0 (if NFR description does belong to that particular NFR type). Eight ML algorithms are applied to each dataset and therefore eight different prediction models were considered for each NFR type.
3. To conduct cost-benefit analysis Once the prediction models are developed, it is very important to analyze their benefits in terms of the cost incurred. Thus, a cost-benefit analysis is conducted to understand and appreciate the advantage of using the proposed models in contrast to not using the models in terms of cost incurred in both the cases.

Empirical analysis is conducted using an open source PROMISE dataset freely available at http://promisedata.org/repository. A set of 15 projects made by MS students at DePaul University that consisted of a total of 326 NFR descriptions categorized into nine types of NFR ("A", "LF", "L", "MN", "O", "PE", "SC", "SE", "US") were used to evaluate the results. The performance of the ML classifiers was compared and evaluated to find out the ML classifier that best predicted the NFR type using the measures derived from Receiver Operating Characteristic (ROC) analysis viz. Area Under the ROC Curve (AUC) and recall. The results indicated the performance of NB classifier has been best in predicting all the NFRs except SE. Also from cost-benefit analysis, it was concluded that the cost incurred without using our proposed models is more than the cost incurred when using the proposed models. Thus, we suggest that any professional from the industry who would use our models for classifying the NFRs into their types would be in profit.

The paper includes in total eight sections. The current literature has been provided in Section 2. Section 3 explains the background of the research. The methodology behind the research has been highlighted in Section 4. The result analysis is presented in Section 5, and Section 6 provides the discussion of the results in terms of cost/benefit analysis and how the work can be useful for the industry practitioners. Section 7 provides threats to validity. Finally, the paper is concluded in Section 8 highlighting the future work.

2. Related work

The area of NFR classification is an emerging area wherein a lot of research is still being carried out. Different authors have employed different techniques and methodologies in order to classify the descriptions of NFRs into their respective types. Table 1 summarizes the work done in the area of NFR classification with respect to types of NFR into which NFR descriptions have been categorized, ML technique used to perform NFR classification, Natural Language Processing (NLP) techniques used to pre-process the data and the dataset used for conducting empirical validation. (The abbreviations and their full forms used in Table 1 are – NB: Naïve Bayes; DT: Decision Tree; SVM: Support Vector Machine; MNB: Multinomial Naïve Bayes; EM: Expectation Maximisation; DMNB: Discriminative MNB, LDA: Latent Dirichlet Allocation; KNN: K-Nearest Neighbor; RB: Rule-Based; HAC: Hierarchical Agglomerative Clustering; ET: Extra Tree; LR: Logistic Regression; NFR: Non-Functional Requirement; FR: Functional Requirement; A: Availability; LF: Look-and-Feel; L: Legal; MN: Maintainability; O: Operational; PE: Performance; SC: Scalability; SE: Security; US: Usability).

Till now, only a few authors have explored this area and an efficient utilization of resources and manpower is required to devise new methodologies and techniques for classifying the NFRs. The primary work in this area has been doneby the authors Cleland-Huang et al. [9]. The authors have used NFR-classifier which is based on information retrieval approach. The method is based on characterizing the different types of NFR using the concept of keywords. These keywords are used to detect requirements pertaining to a particular type of NFR. The work by the authors Hussain et al. [10] extended the idea of Cleland-Huang et al. and showed that the usage of linguistic knowledge is very helpful in the classification. The work incorporates the usage of a part-of-speech (POS) tagger. Apart from this, the authors Gokyer et al. [11] have used SVM algorithm in order to relate NFRs in a textual data to the quality determining attributes. This was accomplished with the help of a knowledge base provided by an expert. Rashwan et al. [12] too used SVM algorithm for automatic categorization of sentences into different classes based on the field of ontology. Similar work was done in the paper [13] that extracted NFRs from textual documents and used the extracted requirements to improve the descriptions of NFRs supporting other Requirement Engineering (RE) tasks. The papers by the authors Casamayor et al. [14] performed classification using a semi-supervised learning approach which is based on a lesser number of requirements in contrast to the supervised approach. The authors proposed a recommender system based on Multinominal Naïve Bayes classifier in combination with Expectation-Maximization (EM) technique. Zhang et al. [15] incorporated a SVM classifier and repeated the experiments of Cleland-Huang et al. [9] again in 2011. They have reported comparatively higher results of precision, although lower results of recall than Cleland-Huang et al. [9]. They have shown that a model based on the individual words outperformed the models based on multi-words. The paper by Slankas et al. [16] utilized a K-nearest neighbor (KNN) supervised learning algorithm for performing NFR classification and compared its performance with SVM and Naïve Bayes techniques. It was observed that SVM algorithm performed better than Multinomial Naïve Bayes classifier and KNN classifier outperformed optimal Naïve Bayes classifier with a unique distance metric. The authors Singh et al. [17] have incorporated rule-based classification technique in order to identify and classify the requirement sentences into NFR sub-classes using the concept of thematic roles. PROMISE corpus and Concordia corpus have been used to validate the results. The authors Kurtanovic et al. [18] studied how accurately the classification of requirements as FRs and NFRs can be done with supervised ML approach incorporating meta-data, lexical, and syntactical features. Similar work was done by the authors in their paper [19–24] which aimed at identifying NFRs in the informal documents like user comments, commit messages and installation manuals. Apart from this, few authors [25–28] have primarily worked on the extraction and classification of only the security requirements as these were considered the most significant type of NFR essential for the development of secure and reliable software.

The work in this paper is based on extraction of relevant keywords from NFR descriptions by employing a series of text mining steps and thereafter classify them into one of the nine types of NFRs ("A", "LF", "L", "MN", "O", "PE", "SC", "SE", "US") using a suitable ML based prediction model. These keywords (also known as indicator terms) are extracted for each NFR type using IG measure. Once the indicator terms for each NFR type are identified, prediction models are developed for the classification of future NFR descriptions whose NFR type is not known. In this study, eight different prediction models (corresponding to eight different ML techniques) were developed for each type of NFR. Since there were nine types of NFR considered in this work, therefore a total of 72 prediction models were developed. An extensive evaluation and comparison of these 72 models was performed with the aim to identify best prediction model for each NFR type that could accurately classify future NFR descriptions whose NFR type is not known. The literature shows (Table 1) that majority of the studies have worked on very few classifiers (maybe two or three). Most of the studies have used SVM and

NB classifiers for developing prediction models. The usage of large number of ML techniques allows us to provide a fair evaluation and conclude the best prediction model that could most accurately classify each type of NFR description. Once we have found the best prediction model corresponding to each type of NFR, we have also performed cost-benefit analysis. This analysis was done to understand and appreciate the cost of using the proposed models vis-à-vis the cost of not using the models. This analysis is very important from industry point of view when the models are required to be used in real sense. This analysis is also missing in majority of the studies.

Table 1: Summary of the studies pertaining to NFR classification

S. No.	Paper	Types of NFR used	ML techniques used	NLP technique used	Dataset used
1	[9]	A, LF, L, MN, O, PE, SC, SE, US	NFR-Classifier	Stemming, Stop-words removal, tokenization	Promise NFR dataset (http://promisedata.org/repository) created by students of DePaul University. It contains 15 projects consisting of a total of 684 requirements specifications (326 NFR + 358 FR).
2	[10]	Not provided	DT	Stemming, POS, tokenization	Promise NFR dataset created by students of DePaul University. It contains 15 projects consisting of a total of 765 requirements specifications (495 NFR + 270 FR).
3	[11]	PE, MN, US, Integrity, Portability, Deployability, Dependability	SVM	Stemming, POS	Web-based transactional applications implemented at Cybersoft.
4	[14]	A, LF, L, MN, O, PE, SC, SE, US, Portability, Fitness, Functionality	MNB coupled with EM	Stemming, Stop-words removal, Normalization	Promise NFR dataset (http://promisedata.org/repository) created by students of DePaul University. It contains 15 projects consisting of a total of 625 requirements specifications (370 NFR + 255 FR).
5	[15]	SE, PE, A, SC, MN, L, US, O, LF, Palm Operational, Fitness	SVM	Stemming, POS, N-gram, Regular Expression	Promise NFR dataset (http://promisedata.org/repository) created by students of DePaul University. It contains 15 projects consisting of a total of 625 requirements specifications (370 NFR + 255 FR).
6	[25]	Security requirements, security-relevant sentences and security-related sentences	NB	tokenization	Three industrial requirements documents viz. Common Electronic Purse Specification (ePurse), Customer Premises Network specification (CPN) and Global Platform Specification (GP). Total number of requirements is 124, 210, 176 for ePurse, CPN, GP with number of security requirements being 83, 42, 63 for ePurse, CPN, GP.
7	[19]	SE, MN, PE, US, Integrity, Portability, Efficiency, Reliability,	SVM, NB, MNB, DMNB, LDA	Stop-words removal	Three different open-source, partially-commercial database systems: (1) MySQL 3.23: contains 320 KLOC of C and C++ source code. Its source control history was used from 31 July 2000–9 August 2004. (2) MaxDB 7.500:

Table 1 continued

S. No.	Paper	Types of NFR used	ML techniques used	NLP technique used	Dataset used
		Interoperability, Testability, Traceability, Accuracy, Modifiability, Modularity, Correctness, Verifiability, Functionality, Understandability, Flexibility			contains 940 KLOC. Its source control history was used from 29 June 2004–19 June 2006. (3) PostgreSQL 7.3: contains 306 KLOC of C code. Its source control history was used from 9 May 2002–26 August 2004.
8	[12]	SE, US, Efficiency, Functionality, Reliability	SVM	Stemming, tokenization	(1) Promise NFR dataset (http://promisedata.org/repository) created by students of DePaul University, contains 15 projects consisting of a total of 684 requirements specifications (326 NFR + 358 FR). (2) A manually annotatedcorpus containing 6 types of requirement documents, 4 are SRSs of different products (online shopping center, student management system, institute of space physics, hospital patient system), 1 supplementary specification document, and 1 use case document. These documents contain 3064 sentences, manually annotated in four main classes (FR, External and Internal Quality, Constraints and other NFRs).
9	[16]	A, SE, L, LF, MN, O, US, Access Control, Audit, Privacy, Capacity, Performance, Recoverability, Reliability	SVM, KNN	Stop-words removal, POS, Lemmatization, Dependency parsing	A series of 11 documents related to Electronic Health Records (EHRs) (https://github.com/RealsearchGroup/NFRLocator). For requirement specifications, CCHIT Ambulatory Requirements, iTrust, and the PROMISE NFR Data Set (http://promisedata.googlecode.com) were used.
10	[20]	Not provided	SVM, MNB	N-gram	2 specifications from Mercedes-Benz (automotive industry).
11	[26]	Confidentiality, Integrity, Identification, Authentication, Accountability, Privacy Availability	SVM, NB, KNN	tokenization	10,963 sentences in six different documents from healthcare domain.
12	[29]	Not provided	SVM	N-gram	Specifications from Mercedes-Benz (automotive industry).
13	[13]	A, LF, L, MN, O, PE, SC, SE, US, Portability, Fitness, Functionality	RB	Stop-words removal, Lemmatization, Dependency parsing	Promise NFR dataset created by students of DePaul University, contains 15 projects consisting of a total of 625 requirements specifications (370 NFR + 255 FR).

Table 1 continued

S. No.	Paper	Types of NFR used	ML techniques used	NLP technique used	Dataset used
14	[30]	SE, PE, SC, US, Reliability	KNN	Not provided	2 case studies: 1st case study utilized the Predictor Models in Software Engineering (PROMISE) dataset, 2nd case study utilized the European Union eProcurement System"s 26 FRs.
15	[31]	SE, PE, L, A, Safety, Privacy, Accuracy, Portability, Reliability, Interoperability, Accessibility	K-means clustering, HAC	Stemming, Stop-words removal, Lemmatization	Three experimental software Java systems from different application domains viz. SmartTrip (an Android mobile application), SafeDrink with a mobile application interface, BlueWallet (subscription-based Web service)
16	[27]	Security requirements of type Authentication, Authorization, Access control, Cryptography-Encryption, Data integrity	DT	Stemming, Stop-words removal, tokenization	Promise NFR dataset (http://promisedata.org/repository) created by students of DePaul University. It contains 15 projects consisting of a total of 684 requirements specifications (326 NFR + 358 FR). Out of 326 NFR specifications, the total number of security requirement specifications is 58.
17	[28]	Security requirements of type Authentication, Authorization, Access control, Cryptography-Encryption, Data integrity	DT	Stemming, Stop-words removal, tokenization	Promise NFR dataset (http://promisedata.org/repository) created by students of DePaul University. It contains 15 projects consisting of a total of 684 requirements specifications (326 NFR + 358 FR). Out of 326 NFR specifications, the total number of security requirement specifications is 58.
18	[17]	Efficiency (Time behavior, Resource Utilization), Functionality (Suitability, Accuracy, SE), US (Operability, Understandability, Attractiveness)	RB	Stemming, POS, tokenization	(1) Promise NFR dataset (http://promisedata.org/repository) created by students of DePaul University. It contains 15 projects consisting of a total of 635 requirements specifications (370 NFR + 265 FR). (2) A manually annotated corpus containing 6 types of requirement documents, 4 are SRSs of different products (online shopping center, student management system, institute of space physics, hospital These documents contain 3064 sentences, manually annotated in four main classes (FR, External and Internal Quality, Constraints and other NFRs).
19	[21]	Reliability, Portability, PE, US	NB, DT, Bagging	Stemming, Stop-words removal, Lemmatization	Two popular Apps viz. Apple App (iBooks in the books category) and Google Play (WhatsApp in the communication category). Total 21969 raw user reviews (6696 FRom iBooks and 4400 FRom WhatsApp) were obtained.

Table 1 continued

S. No.	Paper	Types of NFR used	ML techniques used	NLP technique used	Dataset used
20	[22]	PE, US, Reliability, Supportability, Functionality	SVM, NB, DT, KNN	Stemming	40 Apps from the App Store falling into 10 categories (books, education, games, health, lifestyle, navigation, news, productivity, travel and utilities). A total of 932,388 reviews were obtained.
21	[23]	Reliability, PE, Lifecycle, Capability Usability, System Interface	SVM, KNN	Lemmatization	User requests of open source projects from sourceforge.net, whose user base consists of both software developers and ordinary software consumers.
22	[18]	PE, US, O, SE	SVM	Stop-words removal, POS, Lemmatization, N-gram	NFR dataset consisting of a total of 625 requirements specifications (370 NFR + 255 FR).
23	[32]	A, MN, US, LF, PE, SC, Operability, Fault Tolerance, Portability, Legal and Licensing	MNB, LDA, K-means, HAC	POS, Regular Expression, Entity Tagging, Temporal Tagging	TERA Promise NFR dataset created by students of DePaul University and was updated in 2010. It contains 15 projects consisting of a total of 625 requirements specifications (370 NFR + 255 FR).
24	[24]	MN, US, Reliability, Efficiency, Portability, Functionality	LDA	Stop-words removal, Normalization	Extracted posts (21.7 million) and comments (32.5 million) of the Stack Overflow from July 31, 2008 to September 14, 2014 provided by the MSR (Mining Software Repositories) challenge.
25	[33]	A, L, LF, MN, O, PE, SC, SE, US, Fault tolerance, Portability	Multinomial NB, Bernoulli NB, Gaussian NB, DT, ET, ETs, KNN, Linear LR, MLP, SVM, Label Propagation, Label Spread	Stemming, Stop-words removal, tokenization	TERA Promise NFR dataset created by students of DePaul University and was updated in 2010. It contains 15 projects consisting of a total of 625 requirements specifications (370 NFR + 255 FR).

3. Background of the research

This section includes a brief overview of the ML techniques used for classification along with the description of performance evaluation measures used for evaluating the performance of the prediction models.

3.1. Overview of machine learning techniques

From the literature survey (Table 1) it was observed that majority of the authors have incorporated ML techniques for NFR classification which fall under supervised learning approaches as compared to the techniques which fall under unsupervised learning approaches or semi-supervised learning approaches. Supervised learning ML techniques have accurately performed NFR classification producing promising results. Keeping this in mind, we intended to work on supervised ML techniques. The techniques under supervised learning can be broadly categorized under the following domains: Ensemble Learners, Neural Networks, Bayesian Networks, and Decision Tree. To have a fair evaluation of techniques under all the domains, we selected 1 to 3 ML techniques under each of the domains. In total, we compared and contrasted eight different ML techniques, viz. RF, LB, AB, Bagging (Ensemble learner), MLP, RBF (Neural network), NB (Bayesian network) and J48 (Decision tree). These techniques are popularly used for binary classification in other fields such medical diagnosis [34, 35] network intrusion detection [36], credit card fraud detection [37], defect and change prediction [38, 39], etc. and have shown promising results. Thus, the authors want to explore them for identifying the type of NFR description. A brief overview of these ML techniques is presented in Table 2. These ML classifiers are implemented using the default control settings of an open source tool, WEKA http://www.cs.waikato.ac.nz/ml/weka/. The default control parameters for each of the ML classifier are also provided in Table 2. We have used the default settings and have not tuned the parameters as over-fitting of the parameters may become a threat to external validity.

3.2. Measures for evaluating models

Once the models are trained, testing is performed to evaluate the performance of the models. The performance of the models would be highly optimistic if the testing is performed on the same dataset as the one on which training is performed. Hence, we have used intra-validation technique where the same dataset is partitioned into two subsets, one of which is used for training, while the other is used for testing. The intra-validation technique used in this study is 10-cross validation technique wherein the entire dataset (326 NFR descriptions) is partitioned into 10 equally sized parts (P1, P2,. . . , P10). As can be seen from the Figure 1, for the first time, one part is used for testing (P1), while remaining 9 parts (P2–P10) are used for training the model [48]. The procedure is continued 10 times such that each instance gets validated once and finally a single estimate is produced by combining all the 10 results [49].

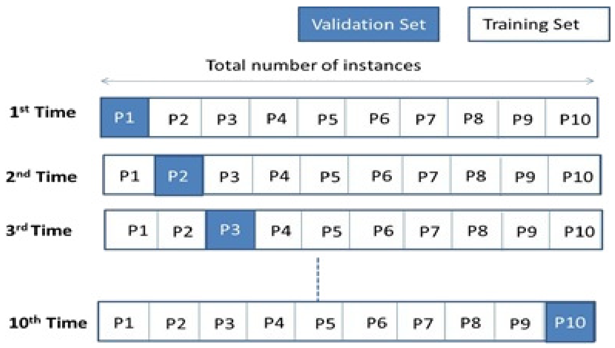


Figure 1. Procedure of 10-fold cross validation technique

For evaluating the performance of the models, we need appropriate performance measures which are suitable to be applied in the given study. In this study, amongst a number of measures available, the authors choose to use two performance measures, recall (also known as recall) and the area under the ROC curve (AUC). The rationale behind using these performance measures has been explained below. Recall is one of the traditional measures which tells that out of actual positive data, how many times the model predicted correctly [50]. Another commonly used traditional measure which is not used in this

Table 2. Description of machine learning techniques and control parameters used in the study

ML Technique	Description	Parameter Settings
J48 decision tree [40, 41]	It is an implementation of C4.5 decision tree algorithm used to handle the classification problems. It generates a binary tree which could be either pruned or unpruned. The pruned tree does not have an influence on the performance of the model while discarding the nodes and branches. It also reduces the risk of overfitting to the training data.	Set confidence factor as 0.25. Set minimum number of objects in leaves as 2 and number of folds as 3. Set seed as 1.
Random forest [42]	It is used for building a number of classification trees, thereby leading to a forest. Each object which needs to be classified acts as an input to the tree in the forest. The process of classification is performed by each tree, and it is said that the tree “votes” for that particular class.	Set the classifier as Decision Tree. Set maximum depth as 0, number of features as 0, seed as 1 and number of trees as 100.
Bagging [42]	In this technique, a sample of data is used to generate various sub-samples of the data which are the training sets and used for making the required predictions by developing the desired model.	Set the classifier as Decision Stump or REPTree. Set number of iterations as 10, weight threshold as 100 and seed as 1. No resampling is used.
Logit-boost [43]	In this technique, the regression technique is used as the base learner and this is followed by performing additive logistic regression. It is the most important type of the Boosting technique.	Set the classifier as Decision Stump. Set number of iterations as 10, number of runs and seeds as 1 and weight threshold as 100. No resampling is used.
Ada-boost [43]	This technique is based on combining different weak learning techniques in order to improve the process of classification, leading to improved results. This is done by first assigning equal weights to all the instances present in the training set and thereafter multiple rounds are conducted and in each round the weights of the examples which have not been correctly classified are increased. This is how the performance of a weak learner is improved.	Set the classifier as Decision Stump. Set number of iterations as 10, number of runs and seeds as 1 and weight threshold as 100. No resampling is used.
Multi-Layer Perceptron [44]	In this technique, a set of input values are mapped to a set of output values wherein learning is done using back-propagation. Firstly the inputs are given to the network. Using the weights applied on each layer and the inputs, the desired output of the network is calculated. Then the error is computed which is difference of the actual value of the output and the calculated value. Based on this computed error value, the weights are updated and accordingly the parameters of the network are adjusted. To achieve the desired performance, this process is repeated again and again.	Set number of hidden layers as a wildcard value “a” = (no. of attributes + no. of classes)/2. Set learning rate as 0.3 and momentum as 0.2. Set sigmoid function for transfer.
Radial Basis Function network [45]	An artificial neural network having a single layer is called RBF network. The activation function used here is the radial functions which are applied to the inputs. These inputs are combined with the weights to produce the desired output of the network.	Set number of clusters as 2 and clustering seed as 1.
Naïve Bayes [46, 47]	This is one of the simplest classifier based on probability wherein the approach for classification is based on Bayes” theorem. The most probable class for a new instance is found out using this technique. A parametric model is used to generate the test data. The Bayes” estimates for the model parameters are calculated using the training data.	Set kernel estimator as false. Set supervised discretization as false.

study is precision. Precision tells that when the model predicts something positive, how many times they were actually positive. Mathematically, recall and precision are defined as follows:

$$Recall = \frac{TP}{TP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

where, *TP* (True Positive): When the document actually belongs to a category "A" (positive) and is predicted by the model to be in category "A" (positive), *FN* (False Negative): When the document actually belongs to a category "A" (positive) and is predicted by the model in category "Not A" (negative), *FP* (False Positive): When the document actually belongs to a category "Not A" (negative) and is predicted by the model to be in category "A". We have focused on recall and did not use precision because of the following two reasons:

1. In this study, *FN* is more significant than FP and as a result *FN* cannot be ignored. If we cannot ignore *FN*, then we have to take into account recall measure. Let us understand why recall is more important than precision in this study. *FN* occurs when a document is predicted by a model to be in category "Not A" when it actually belongs to category "A". *FP* occurs when a document is predicted by the model to be in category "A", when it actually belongs to category "Not A". Now when a *FN* occurs, a document which actually belongs to category "A" is ignored by the stakeholders of the software because it is predicted to be in category "Not A". This may result in delivery of poor quality software and may have serious implications on the industry in terms of its reputation in long term. However, when a *FP* occurs, some extra resources of the industry (in terms of time, money and manpower) may be utilized as the document actually belongs to a category "Not A" (negative) and is predicted by the model to be in category "A". Clearly, *FN* holds a more significant position as releasing poor quality software is more disastrous as compared to utilization of some extra resources. Thus, in this study, we have reported recall and did not consider precision.
2. Moreover, the datasets (9 datasets with binary dependent variable formed from a single dataset) used in this work for model prediction are imbalanced where the number of instances belonging to negative class is more than the number of instances belonging to the positive class. Studies in literature have criticized the use of precision when the models are validated using imbalanced datasets [51, 52].

Instead, to handle the imbalance nature of the datasets, we have used an effective performance measure known as Area Under ROC Curve (AUC). ROC curve is a plot of recall (true-positive) on the y-coordinate versus its 1-specificity (false positive) on the x-coordinate. It is used to measure the accuracy of the model and its values lie in the range of 0 to 1, where an AUC value of 1 indicates the best prediction. When the data is imbalanced, the model is biased towards the majority class while the minority class is predicted with less accuracy. To handle this, studies [51, 53, 54] propose the use of AUC as it is insensitive to class distribution changes. In other words, it is robust to imprecise class distribution and misclassification costs [55].

4. Methodology behind the work

This section discusses the methodology incorporated to accomplish the classification of NFRs. Figure 2 depicts three steps used to develop the predictive models. The steps have been explained in the following subsections.

4.1. Gathering of NFR descriptions

The dataset used in this work for empirical analysis was the same dataset which was used by the authors Cleland-Huang et al. [9]. This dataset consist of a set of 15 projects which were made by MS students at DePaul University available at the open source PROMISE software engineering repository http://promisedata.org/repository. These projects contained a total of 326 NFR descriptions and 358 FR descriptions. These

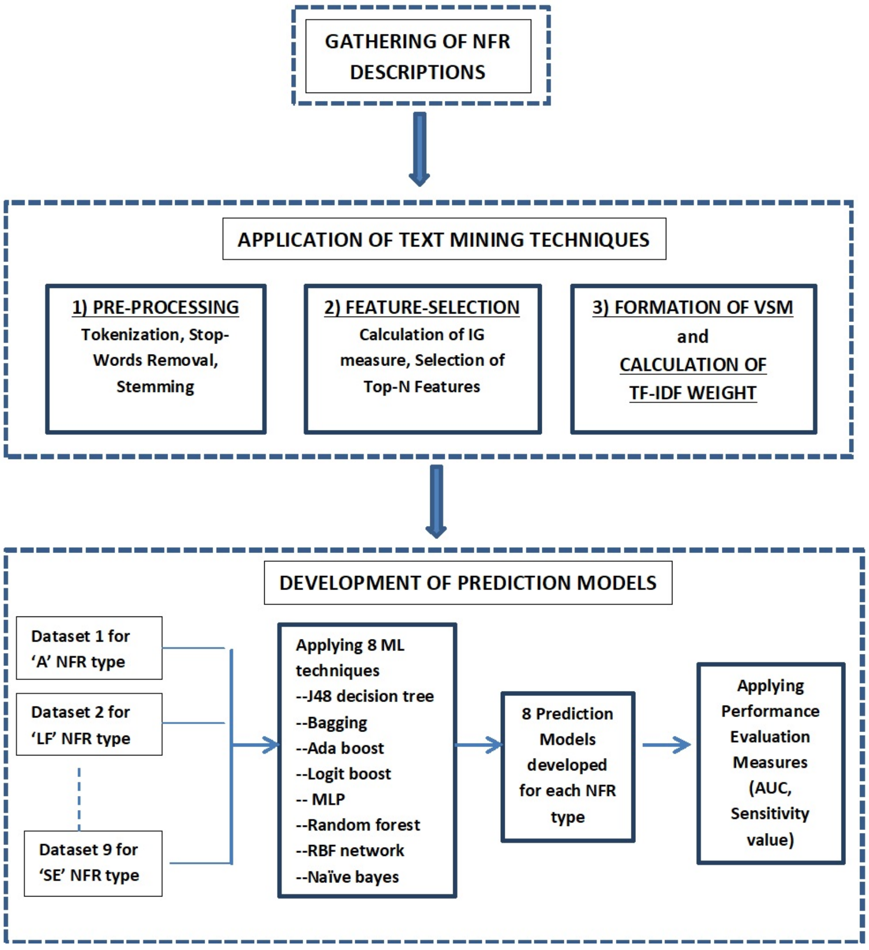

GATHERING OF NFR DESCRIPTIONS
APPLICATION OF TEXT MINING TECHNIQUES
1) PRE-PROCESSING
Tokenization, Stop-Words Removal, Stemming
2) FEATURE-SELECTION
Calculation of IG measure, Selection of Top-N Features
3) FORMATION OF VSM and CALCULATION OF TF-IDF WEIGHT
DEVELOPMENT OF PREDICTION MODELS
Dataset 1 for 'A' NFR type
Dataset 2 for 'LF' NFR type
Dataset 9 for 'SE' NFR type
Applying 8 ML techniques
--J48 decision tree
--Bagging
--Ada boost
--Logit boost
-- MLP
--Random forest
--RBF network
--Naïve bayes
8 Prediction Models developed for each NFR type
Applying Performance Evaluation Measures (AUC, Sensitivity value)

Figure 2. Framework used for classifying NFR descriptions

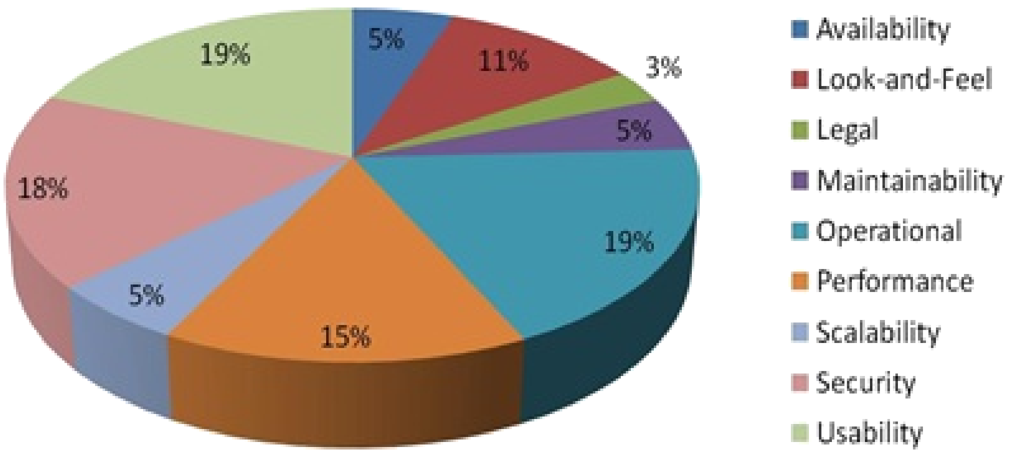


Figure 3. Percentage of requirements belonging to each type of NFR

326 NFR descriptions have been categorized into nine types of NFR viz. availability, look-and-feel, legal, maintainability, operational, performance, scalability, security, and usability. The percentage of NFRs belonging to each NFR type is shown in Figure 3.

4.2. Application of text mining techniques

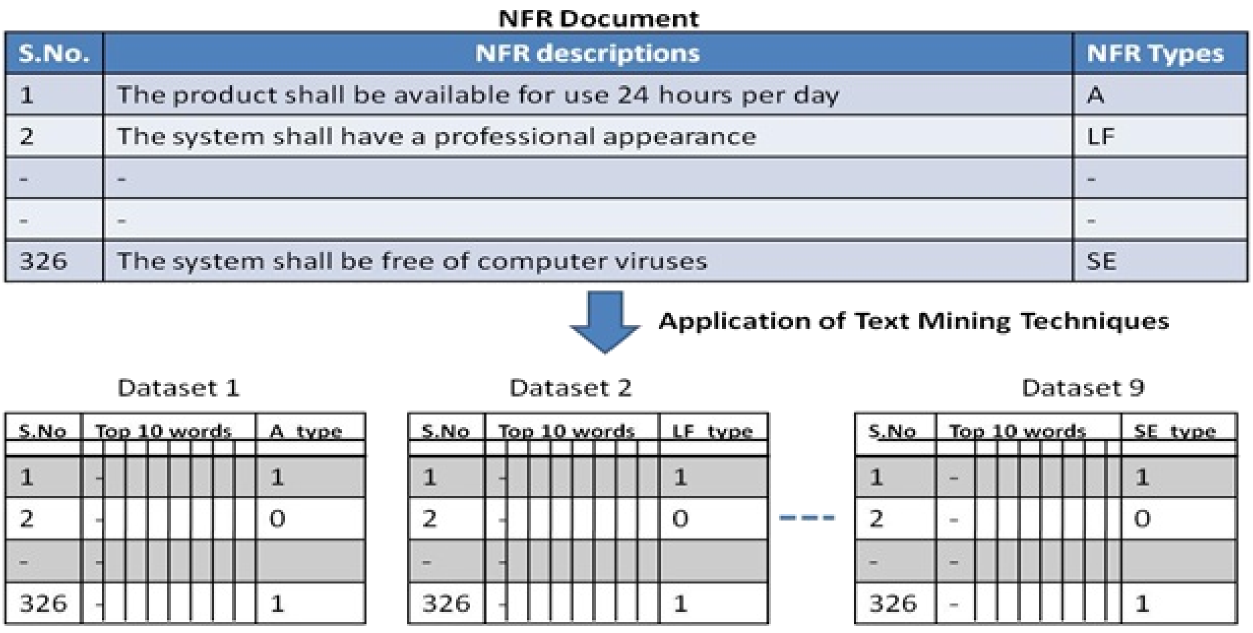
This step concerns with the analysis of NFR descriptions to identify significant keywords (indicator terms) pertaining to each type of NFR. 326 NFR descriptions were extracted from the requirement specification documents and applied to a series of text mining steps. Figure 2 demonstrates the steps of text mining which has been explained below:

1. Pre-processing: Each document is represented using Bag of Words (BOW) representation method in which a document is considered to be a collection of thousands of words which occur in it at least once. Many of these words are not relevant for the learning task and their usage can degrade the performance of a classifier. Thus, a series of preprocessing tasks like tokenization, stop-words removal and stemming are required in order to remove the irrelevant words from the document. Text mining process begins by first converting the entire document in the form of tokens, i.e., a set of words. This is known as tokenization. This is followed by removing the words from the document that do not add any meaning to the data (stop words like prepositions, articles, conjunctions, etc.). Finally, stemming is performed [56]. Instead of stemming, another popular technique which could be used is lemmatization. However, we preferred stemming over lemmatization as the computation time involved in stemming is lesser as compared to lemmatization which is useful incase of large datasets and long texts.
2. Feature-Selection: Once pre-processing is performed, set of relevant words called indicator terms need to be identified specific to each type of NFR. In this work, Information-Gain (IG) measure has been used as the feature selection method. The working of IG measure is based on finding a collection of words from the document that best identify the target concept (NFR type) [57]. It is based on the concept of entropy deduction which occurs when the dataset is split on an attribute [57]. In other words, IG is the amount of information that is gained by evaluating the IG value of each attribute in the dataset. It is defined as the difference of the entropy of the dataset before the split and the entropy of the dataset after the split. Entropy of the entire dataset determines the amount of uncertainty in the information that needs to be assessed. IG measure works by associating a weight to each of the word obtained after pre-processing and then the top-N scoring words are selected as the indicator terms. In our work, the value of N is considered as 10 as the paper by Cleland-Huang et al [9] showed that good results were achieved when top-10 words were considered as compared to the results achieved when top-5 words or all the words were considered together. Table 3 depicts the top-10 indicator terms in decreasing order of IG measure, corresponding to each of the nine NFR types.
3. Vector Space Model: Once feature selection has been done using IG measure, we will have total number of N indicator terms which can be represented as $t_1, t_2, \ldots, t_N$. Each ith document is then represented as a N-dimensional vector consisting of N values written as $(X_{i1}, X_{i2}, \ldots, X_{iN})$. Here, X_{ij} is a TF-IDF (Term Frequency Inverse Document Frequency) weight measuring the importance of the jth term t_j in the ith document. The complete set of vectors corresponding to all the documents under consideration is called a Vector Space Model (VSM).

4.3. Development of prediction models

Once the indicator terms for each NFR type are identified, prediction models are developed for the classification of future NFR descriptions whose NFR type is not known. As depicted in Figure 4, nine datasets are developed corresponding to each NFR type from an initial dataset.

This initial dataset is the original NFR document that consists of 326 NFR descriptions belonging to one of the nine NFR types. Each dataset has the total number of instances as 326 with top-10 indicator terms of that particular NFR type as independent variables and a binary dependent variable.


NFR Document

S.No.	NFR descriptions	NFR Types
1	The product shall be available for use 24 hours per day	A
2	The system shall have a professional appearance	LF
-	-	-
-	-	-
326	The system shall be free of computer viruses	SE

S.No	Top 10 words	A type
1	-	1
2	-	0
-	-	
326	-	1

S.No	Top 10 words	LF type
1	-	1
2	-	0
-	-	
326	-	1

S.No	Top 10 words	SE type
1	-	1
2	-	0
-	-	
326	-	1

Figure 4. Process involved in NFR classification

Table 3. Top-10 indicator terms specific to each NFR type sorted by IG measure

Rank	A	LF	L	MN	O	PE	SC	SE	US
1	achiev	simul	regul	updat	environ	second	simultan	access	easi
2	hour	ship	compli	mainten	interfac	respons	handl	author	train
3	day	sound	disput	chang	window	time	year	ensur	understand
4	pm	interfac	legal	nfl	server	longer	capabl	authent	intuit
5	time	appear	rule	season	user	minut	support	prevent	instruct
6	long	appeal	histori	releas	web	return	number	allow	select
7	onlin	shot	requir	integr	establish	fast	expect	logon	realtor
8	avail	color	complianc	code	oper	flow	concurr	secur	learn
9	technic	compli	conform	pattern	second	add	increas	polici	symbol
10	year	access	standard	offer	custom	prepaid	launch	malici	natur

These top-10 indicator terms (extracted using IG measure) specific to each NFR type are shown in Table 3. Dependent variables will have the value of 1 or 0 depending on the type of NFR. For instance, dataset 1 is pertaining to “A” NFR type, so it will have the value of the dependent variable as 1 for all the NFR descriptions pertaining to “A” NFR type and the value of 0 for all other remaining NFR descriptions. Corresponding to each of the nine datasets, 8 prediction models are developed by employing 8 ML techniques (J48 decision tree, RF, Logitboost, Adaboost, MLP, RBF network, Bagging, NB) on each of the datasets. These prediction models can be used for the classification of future NFR descriptions whose NFR type is not known.

5. Result analysis

This section presents the results of eight different ML techniques when applied to nine different models developed with respect to their corresponding NFR types. The performance measures which have been used in evaluating the performance of these ML techniques are AUC and recall.

In this section, we will broadly discuss the following two Research Questions (RQs):
RQ1: Which ML technique is best for predicting each type of NFR such as performance, security, look-and-feel, etc.?
RQ2: Which NFR has been best predicted in terms of classification and misclassification rate?

5.1. Analysis of RQ1

To address the RQ1, the performance of the ML classification models to predict each category of NFR is depicted in Figure 5. Figure 5 depicts the comparative analysis of the ML models and determines how well these models have performed in predicting different types of NFR. The data

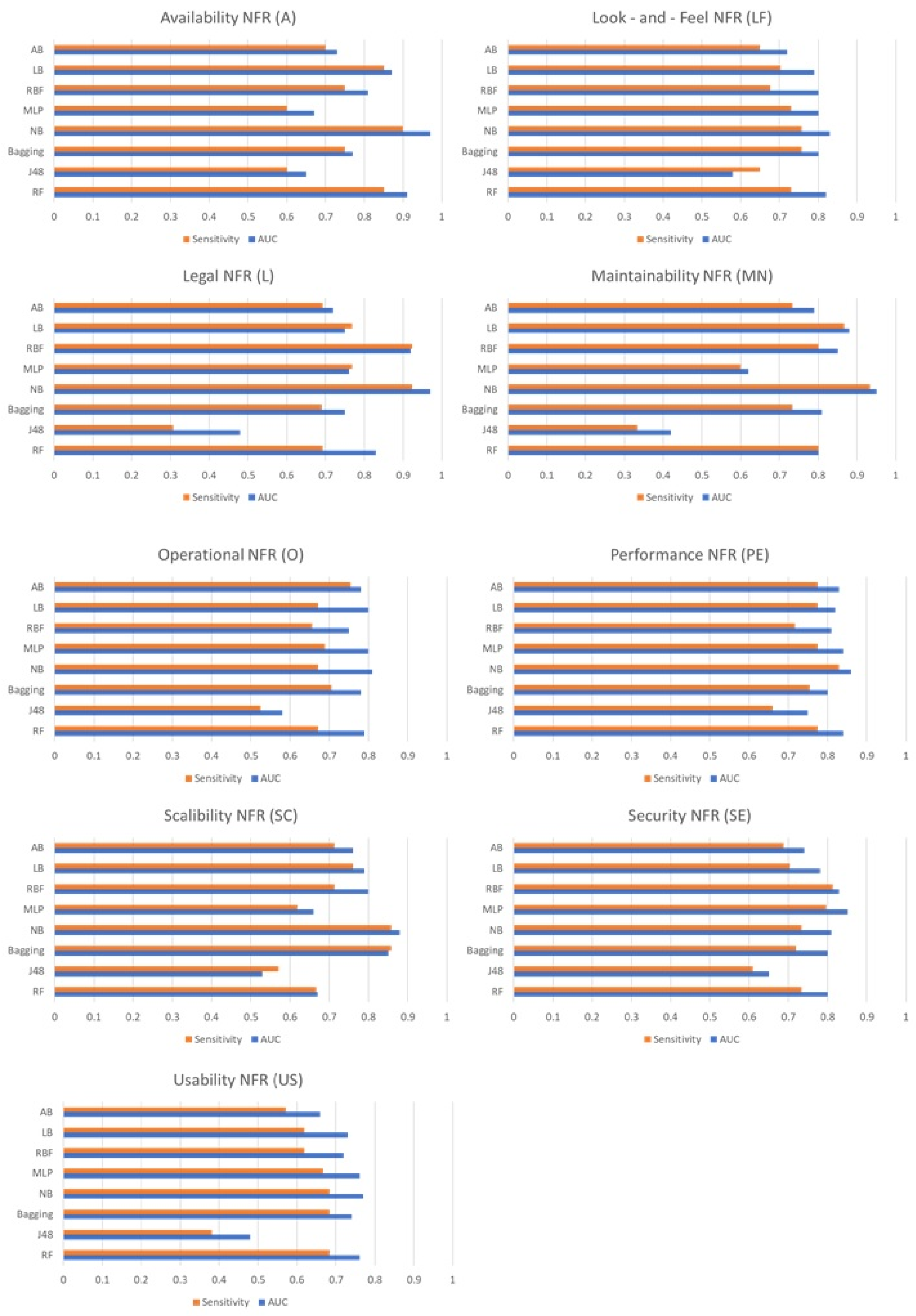


Figure 5. Graphical representation depicting the performance of ML techniques

values corresponding to each figure are shown in appendix (Tables A1–A4). We can observe from the Figure 5 that different NFR types responded differently to each classification method. For example, the performance of "A" NFR in terms of AUC when predicted by different classifiers is quite different, ranging from 0.65 to 0.97. Similar observations can be seen with all other NFR types. In other words, if a particular ML model has given a high value of AUC in predicting a particular type of NFR, it may not be necessary that it is also giving high accuracy in predicting other NFR types. This may be due to the reason that each NFR is very different from the others and thus, the top-10 words selected corresponding to each NFR are very different. Since the classification models are based on these top-10 words, the same classifier is performed differently on different NFRs. Given this scenario, the identification of a suitable classifier to predict each type of NFR will be highly beneficial for researchers and academicians. Figure 5 shows that the NFR "A" has been best predicted by NB giving AUC values as high as 0.97 and recall value as 90.0%. This is followed by RF giving AUC of 0.91 and recall as 85.0%. The graphs show that all NFRs except "SE" are best predicted by NB classifier. NB gives the highest AUC of 0.97, 0.83, 0.97, 0.95, 0.81, 0.86, 0.88, and 0.77 for "A", "LF", "L", "MN", "O", "PE", "SC" and "US" types of NFR, respectively. Their corresponding recall values are also high in the majority of the cases. The probable reason of NB performing well could be due to its assumption of attributes to be independent given the value of class variable [40]. Moreover, NB does not fit nearly as much, so there is no need to prune or process the network. For "SE" NFR, MLP has given the highest AUC of 0.85. After NB technique, RF and RBF techniques have shown the second highest AUC for the majority of the NFRs. The AUC of RF and RBF lies in the range of 0.67 to 0.91 and 0.72 to 0.92, respectively. The performance of the bagging technique can be considered as an average in predicting the NFR descriptions of all types with the values of AUC lying in the range of 0.74 to 0.85. Similar performance has been depicted by LB and AB techniques. The performance of these techniques has been overall good in predicting the NFR descriptions with the values of AUC lying in the range of 0.73 to 0.88 corresponding to LB and the AUC value falls in the range of 0.72 to 0.83 with respect to AB. On the contrary, J48 decision tree technique has not performed well in classifying the NFR descriptions into their respective types as the highest value of AUC obtained is 0.75 with 66.0% recall value. J48 has shown the lowest performance (in terms of both AUC and recall) in predicting all types of NFRs. From the above analysis, it is summarized that the performance of NB has outperformed all other classifiers and it is overall best in predicting the NFRs. We suggest researchers and academicians to use NB models for predicting the NFRs.

5.2. Analysis of RQ2

To address RQ2, NFRs are compared across each other by comparing the classification and misclassification ability of the best ML model for each type of NFR found in RQ1. The classification ability of the ML model is found by comparing the NFRs using AUC and recall values. In this study, the keywords form the basis for the identification of NFR documents into their respective types of NFRs. Thus, the performance of the ML model also depends on the values of these keywords. The NFRs which is predicted with high AUC and recall implies that the keywords pertaining to that NFR type are of great significance for the particular ML model to perform classification. It also implies that the dataset in turn consists of a good number of these requirements and thus the ML models are trained well on such datasets.

Whereas, the misclassifications by the ML model are found by determining the number of False Negative (*FN*) and False Positive (*FP*) of each type of NFR. As we have discussed in Section 3.2, *FN* occurs when the document actually belongs to a category "A" (positive) and is predicted by the model in category "Not A" (negative). Whereas, *FP* occurs when the document actually belongs to a category "Not A" (negative) and is predicted by the model to be in category

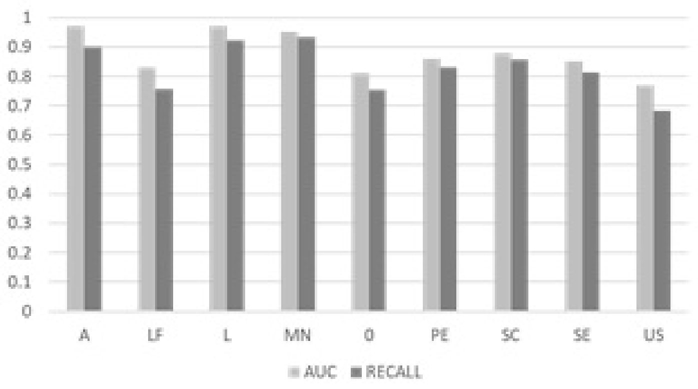


Figure 6. Bar graph showing comparison amongst NFRs in terms of AUC and recall

Table 4. Number of misclassifications of NFRs

NRF Type	Misclassifications			Misclassifications done by ML model
	False Negative (*FN*)	False Positive (*FP*)	Total (*FN+FP*)	
Availability (A)	7 (2.01%)	8 (2.30%)	15 (4.32%)	NB
Legal (L)	8 (2.30%)	5 (1.44%)	13 (3.74%)	NB
Look-and-Feel (LF)	8 (2.30%)	22 (6.34)	30 (8.64%)	NB
Maintainability (MN)	4 (1.15%)	5 (1.44%)	9 (2.59%)	NB
Operational (O)	15 (4.32%)	36 (10.37%)	55 (15.85%)	NB
Performance (PE)	25 (7.20%)	7 (2.01%)	32 (9.22%)	NB
Scalability (SC)	15 (4.32%)	8 (2.30%)	23 (6.62%)	NB
Security (SE)	8 (2.30%)	35 (10.08%)	43 (12.39%)	MLP
Usability (US)	1 (0.28%)	29 (8.35%)	30 (8.64%)	NB

"A". It is also discussed in Section 3.2 that *FN* is more significant that *FP* and thus, a model with lower *FN* is more desirable. The classification ability of the best ML model in predicting each type of NFR can be seen from Figure 6. On the other hand, the number of misclassifications done by the best ML classification model for each type of NFR is shown in Table 4. Table 4 shows the misclassifications for each type of NFR in terms of number of *FN* and *FP* along with their percentages. It can be seen from Figure 6 that "A", "L" and "MN" (in this order) have shown high AUC values, followed by "PE", "SC" and "SE". High recall values are shown by "MN", "L", and "A" (in this order). Thus, we can say that overall (taking AUC and recall together) "L" has been best predicted amongst all NFRs. This might be due to the reason that the "L" type of NFR is comparatively easy to understand and collect and thus, the data consisted of a good number of correctly elicited requirements pertaining to this NFR. In addition to this, Table 4 shows that only 13 "L" type NFRs are misclassified by NB model which is amongst the lowest compared to other misclassification rates. However, amongst these 13 misclassifications, there are more number of *FN* than *FP*, which is not desirable. Hence, such models may not be used for future unknown predictions. In contrast, the misclassification rates to predict "Ă" and "MN" type of NFR are also low (4.32% and 2.59%, respectively). "MN" in fact has been least misclassified amongst all the NFRs. Also, the number of *FN* is less than *FP* for both "A" and "MN". Thus, overall, we can say that NB model has performed best in predicting "MN" and "A" types of NFRs when both classification and misclassification are taken together. The NFR which is predicted with lowest AUC and recall is "US" as can be seen from Figure 6. It can also be seen from Table 4 that

30 "US" type NFR have been misclassified which is amongst the highest. NB has given the highest misclassification rate (15.85%) for "O" type of NFR, implying that 55 "O" types of NFRs have been misclassified as some other NFRs. In terms of AUC and recall also "O" has been predicted with low values. The possible reason of misclassification may be due to the ambiguities caused by the indicator terms (keywords). These keywords form the basis for the identification of NFR documents into their respective types of NFRs and hence need to be carefully analysed to avoid misclassification. However, it has been observed that some of the NFR specifications of different NFR types have been described by the same set of keywords. In other words, some keywords tend to occur across multiple requirements of different NFR types. This gives rise to ambiguities, thus leading to false classification. Similar observations were given by Cleland-Huang et al. [9] and Sharma et al. [5]. Let us understand this with the help of a suitable example. Consider the keyword "colour" which is an indicator term of look-and-feel NFR. Presence of the term "colour" in the requirement sentence "The application shall match the colour of the schema set forth by Department of Homeland Security" clearly shows that the requirement is about look-and-feel NFR. However, the presence of the term "colour" in the requirement sentence "The list of dispute cases must be colour coded for easy identification of dispute cases based upon the dispute case status" does not necessarily represent any look-and-feel NFR, but rather represent "usability" NFR. The hint to identify look-and feel type NFR is the presence of other terms/patterns in the requirement sentence. For example, in the first sentence, the term "match" puts a constraint on "colour of the schema". However, in the second sentence, no such constraint is there to guarantee that it is a look-and-feel NFR. Thus, identification of the NFR using keywords may lead to false classification and a detailed analysis of semantic patterns and structure of NFR descriptions could be done to improve the results. Thus, future work will therefore investigate the possibility of using categories of indicator terms or extended training to improve these retrieval results.

5.3. Comparison with state-of-art

In this section, we discuss the implication of the results where we provide important insights inferred. We have compared our results to the state of the art [9, 12] and have qualitatively examined the wrongly classified cases to generate some useful insights. We have also discussed what could be done which may lead to improved performance of the proposed models.

The comparison in terms of recall values is shown in Table 5. For the purpose of comparison, we have considered the highest value of recall given by the different ML models for predicting each type of NFR. The bar chart in Figure 6 shows the highest recall value (given by different ML models) for each type of NFR. The authors, Rashwan et al. (2013) [12] have used the same dataset as used in our study and have classified the NFR descriptions using SVM classifier. From Table 5, it can be clearly seen that the prediction models proposed in this study have classified NFR descriptions of all types with higher recall values than the model developed by Rashwan et al. [12] using SVM classifier. This is despite the fact that SVM classifier is one of the most popularly used ML techniques in the field of NFR classification as can be seen from the literature survey (Table 1). We can even observe from the table that for some of the NFR types, viz. "MN" and "SC", the model of Rashwan et al. [12] has given extremely low values of recall (below 0.5). A recall value of 0.5 means that for every correct prediction, the next prediction is incorrect. There is no practical usage of such classifiers and they are known as random classifiers. The study [12] has shown that the SVM classifier has performed even worse than a random classifier for "MN" and "SC" NFR types. On the contrary, in this study, "MN" has been predicted with the highest recall value of 0.93 amongst all other NFRs. Similar observations are made when the recall values of NFRs in this study are compared with the recall values obtained in another study by Cleland-Huang et al. [9]. Table 5 shows that the recall values of all NFRs except "US" are higher than the recall values obtained in [9]. Thus, overall we can conclude that the results in this study are higher than the results obtained in both the compared studies [9] and [12].

Table 5. Comparison with state-of art

NRF Type	Recall (State-of-Art) Cleland, Huang et al. [9]	Recall (State-of-Art) Rashwan et al. [12]	Recall (Our Model)
Availability (A)	0.89	0.66	0.90
Legal (L)	0.70	0.61	0.92
Look-and-Feel (LF)	0.51	0.63	0.76
Maintainability (MN)	0.88	0.41	0.93
Operational (O)	0.72	0.66	0.75
Performance (PE)	0.62	0.70	0.83
Scalability (SC)	0.72	0.38	0.86
Security (SE)	0.81	0.70	0.81
Usability (US)	0.98	0.62	0.68

Next, we analyze the NFR type which has been predicted with the lowest recall value in this study. As can be seen from Table 5, "US" NFR type has been predicted with the lowest recall.This might be due to the reason that NFR descriptions pertaining to "US" NFR type may be comparatively difficult to understand and collect. This is also evident from the dataset used in this study which shows that there are only 3% of NFR descriptions pertaining to "US" type of NFR. This low percentage of "US" type NFR descriptions give a low value of recall. Another implication which can be drawn from our results is that our models have not shown exceptional performance with respect to few of the NFR types. This might be because of the reason that NFR identification was done on the basis of indicator terms (keywords). In other words, the indicator terms form the basis of the classification of an unknown NFR type into its correct type. However, it has been observed that some keywords tend to occur across multiple requirements of different NFR types, leading to false classification. Similar observations were given by Cleland-Huang et al. [9] and Sharma et al. [5]. The illustration to explain this has been given in Section 5.2 (last paragraph).

6. Discussion

This section provides a discussion of the results in terms of cost/benefit analysis and how the results retrieved from this study can be useful for industry practitioners. The work in this paper is concerned with the development of nine different classification models specific to each type of NFR with the aim to classify NFRs into the respective types based on their descriptions specified in the requirement specification document. During the elicitation process, requirements analysts may generate large amounts of unstructured SRS documents consisting of the requirement specifications being scattered throughout the document in a random and adhoc fashion. The descriptions are extracted from the document and applied to a series of text mining steps to retrieve indicator terms specific to each NFR, leading to the detection and classification of NFR in the initial phases of SDLC process.

6.1. Analyzing returns on investment

We conducted cost-benefit analysis to evaluate the effectiveness of prediction models used for predicting the type of NFRs. The result section (Section 5) concludes the best ML technique for predicting each type of NFR amongst the various ML techniques used. In other words, we suggest that researchers, practitioners, and academicians may use those ML techniques for predicting the required NFR type. In this section, we discuss the cost of using the proposed model and the cost of not using the proposed model.

To calculate the cost of the model, we have considered a cost matrix having the value of 1 unit for false positives and false negatives (incorrect predictions), whereas 0 unit for true positives and true negatives (correct prediction). The values of 1 unit and 0 unit for incorrect and correct decisions, respectively, are considered as it is well understood that one may need to pay

Table 6. Cost incurred with/without using proposed models

NRF Type	Proposed ML Model	Cost of the Model	Cost without Model	Gain	Profit/Loss
Availability	NB	12	41	29	Profit
Look-and-Feel	NB	24	59	36	Profit
Legal	NB	10	25	15	Profit
Maintainability	NB	9	27	19	Profit
Operational	NB	45	99	55	Profit
Performance	NB	28	80	52	Profit
Scalability	NB	21	28	7	Profit
Security	MLP	34	103	69	Profit
Usability	NB	30	85	55	Profit

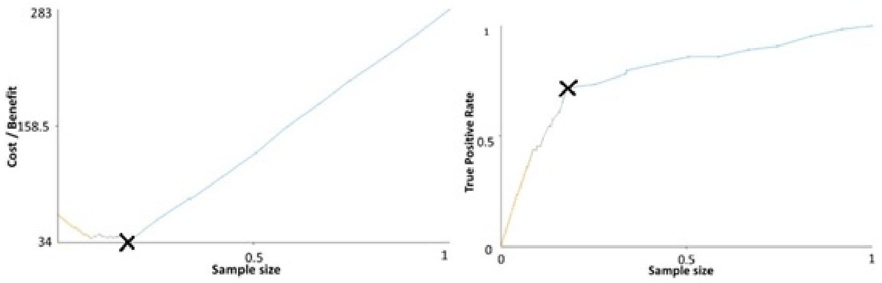


Figure 7. Cost/benefit curve

the price when the model takes wrong decisions while there is no cost involved when the model takes correct decisions. Therefore, the cost of using the model is calculated using the formula: $(No.\ offalsepositives \times 1)+(No.\ offalsenegatives \times 1)$. To conduct the cost-benefit analysis, the models are run at different threshold values and the performance of the models is analysed through the confusion matrix at each threshold value. The cost of the model, measured in terms of false positives and false negatives changes as the threshold changes. The minimum cost obtained is considered as the cost of the classification model (shown in Table 6). This cost is compared to the cost incurred without using the model. The cost incurred without using the model is found by selecting the same number of instances at random [58]. The difference between the values of the cost function by random selection and the value of the cost from the model is called gain. The Gain can be interpreted as the profit obtained by using the classification model instead of random selection of the same number of instances. We can observe from Table 6 that in all the cases, the cost incurred without using the model is more than the cost incurred by using the proposed model. In other words, we suggest that any professional from the industry who would use our model for classifying the NFRs into their types would be in profit from the gain as shown in Table 6.

For the purpose of demonstration, a plot of "cost/benefit curve" and "threshold curve" is depicted for MLP model used for predicting "SE" type of NFR. We have shown the plot corresponding to MLP model as it has given the highest gain of 69 units. Likewise, we can draw for all other models and infer the similar meaning. The "cost/benefit curve" is a plot of sample size (part of selected samples) on X-axis and cost/benefit on the Y-axis [59]. The "threshold curve" is a plot of true positive rate on Y-axis and sample size on X-axis. Threshold curve corresponds to the part of the selected instances ("Sample Size"). In other words, the threshold curve depicts the dependence of the part of "positive" samples retrieved during virtual screening upon the part of the samples selected from the whole dataset used for screening. It should be noted that only those samples are selected during virtual screening, for which the estimated probability of being "positive" exceeds the chosen threshold. The "cost/benefit curve" in Figure 7 shows that the minimum cost is 34 which is increasing reaching the maximum value of 283. The cross symbol denoted by "X" denotes the sample size retrieved during virtual screening. This sample size is important as we take the same number of instances to calculate the cost incurred without using the model. Similar inferences can be drawn for all other "cost/benefit curves" and "threshold curves".

6.2. Implications from industry viewpoint

The results of this work will be of interest to researchers as well as practitioners from the industry, who are interested in finding the type of NFR based on their descriptions in the early phases of software development, thus improving software quality. Timely identification of quality requirements would be of great benefit to the software developers from the industry as these quality requirements play a critical role in the design and architecture of the system. The architecture of the system built acts as the scaffolding in which the functionality of the system is delivered, thus ensuring that the system delivered meets the customer's functional expectations and needs. Consider a situation where a NFR remains undiscovered or is not elicitated properly during the early phase of software development, and is discovered at the later stages of development or when the software is released. In such a situation, the entire technical architecture has to be redesigned, leading to the wastage of limited resources in terms of time, money, and manpower [60, 61]. Thus, to avoid this situation, the models proposed in this study can be used in the early phases for the identification and classification of NFRs. When the software is released and the customer finds that all his requirements (both functional as well as nonfunctional) are met, the customer feels satisfied and happy. This leads to the increasing the reputation and status of the software organization (in which the software is developed) in the market. Thus, customer satisfaction which is of utmost importance in today's scenario is met. Furthermore, practitioners from industry will be able to detect and classify NFRs from a previously uncategorized requirement specification in an automated way, thus avoiding the need for manual evaluation which like all human activities has a tendency to be error prone. It will also help researchers from the industry to extract viewpoints for different NFR qualities of interest. For example, a security analyst could issue a request to retrieve all descriptions related to security issues, or a GUI designer could issue a request to retrieve information about stakeholders" usability or look-and-feel concerns.

Thus, we have seen how industries can use the proposed models for identifying the NFRs and predicting the unknown NFR with its NFR type. Instead of using the proposed models, the identification of NFRs can also be done manually with the help of a human analyst. We have briefly discussed the effort required to conduct the work manually and have shown that the manual identification and prediction of NFRs is not feasible.

Human analyst can be considered as the requirement engineers in this study. Since the SRS documents are large enough and the NFRs are scattered throughout the document, it is very difficult or not feasible for the requirement engineers (human analyst) to perform the task of identifying the NFRs manually. Thus, this motivated the authors to automate this process with the help of algorithms and tools. This automation is done with the help of text mining techniques, which are used for extracting useful information (in the form of indicator terms specific to each NFR type) from a large number of documents (a large document corpus) without requiring humans to actually read and summarize the text. The automation of identification and classification of NFRs broadly consists of the following steps:

1. Use of text mining steps to retrieve the independent variables: A series of text mining steps were applied on the NFRs descriptions to retrieve the independent variables (indicator terms). These steps begin with pre-processing (tokenization, stop words removal, stemming) followed by application of feature selection method and finally apply-

ing TF-IDF weighting. Each of these steps can be studied in detail from Section 4.2. In this work, a dataset consisting of a total of 326 NFR descriptions which are categorized into 9 types of NFR was considered. For each type of NFR, indicator terms (top-10 words) were retrieved. So, we had to run our text mining module nine number of times and each time the above steps of text mining were done. An automated text mining module takes as input all the 326 NFR descriptions at a time which had not been the case if it was done by human analyst. Text mining steps had to be applied to each of the 326 NFR descriptions individually if it was done manually by human analyst. The same procedure had to be done 9 times for each NFR type. This is humanly impossible as each of the above text mining steps is complex in nature, thus consuming a lot of time. Pre-processing step involves natural language complications as we deal with textual requirements which are purely written in a natural language which is followed by feature selection and TF-IDF weighting that involves mathematical calculations. It is therefore not possible to do manual computation on each NFR to retrieve top-10 words and then calculate TF-IDF values for these words.
Thus, we have automated this process with the help of a tool developed by the authors. The input to this tool is a set of 326 NFRs descriptions and the output is top-10 words sorted on the basis of Information-Gain measure (feature selection method). Thus, the entire process is automated and the time taken to produce the output is less than approximately 1 minute. Whereas, to perform the same activities manually, it would take us hours and may not be even feasible for large datasets.
2. Development of prediction models using machine learning classifiers: Once indicator terms (independent variables) for each NFR type are retrieved, then dataset corresponding to each of the 9 types of NFR is made consisting of a binary dependent variable (NFR type) having the value of 1 or 0 depending on the type of NFR. Corresponding to each dataset, eight different prediction models were developed by employing eight ML classifiers on each dataset viz. J48 decision tree, Random Forest (RF), Logitboost, Adaboost, Multi-Layer Perceptron (MLP), Radial Basis Function (RBF) network, Bagging, Naïve Bayes (NB). These ML classifiers are implemented through an open source tool, WEKA. There were in total 72 models (9 datasets * 8 ML classifiers = 72 prediction models) and it was observed that to run each model, CPU time of approximately 30 sec to 1:30 minutes was required.
Had the same task been done manually, the human analyst would have to use his skills and knowledge in classifying NFR description into particular type of NFR. Also, as the number of models is quite large (72), it would have taken a lot of time to complete the computation. Moreover, it would have been very difficult to achieve a high performance as achieved by our models.
Never the less, human computation is always prone to error. Human error is inevitable and normal which may lead to system failure if not handled at the right time. Thus, we conclude that our prediction models are highly recommended in contrast to human analyst (i.e., work done manually) for classification of future NFR descriptions whose NFR type is not known, leading to less computation time and more accuracy.

7. Threats to validity

The empirical validation in this work has certain limitations which may adversely affect the validity of the results. These limitations are discussed in terms of four threats to validity, viz. construct validity, internal validity, external validity, and conclusion validity.
1. Construct Validity
Construct validity is one of the most important threats to validity. It is defined as the extent to which the variables (independent and dependent variables) and the performance pa-

rameters precisely measure the concept they intend to measure [62–64].
This threat can be due to the improper collection of the dependent variable and the independent variables. The dependent variable used in the study refers to the "types of NFRs" and the independent variables are the top few words of the document which determine how important they are within that particular document and also across a group of other documents. The collection of data for the classification of NFRs into their respective types has been done by mining the descriptions of NFRs specified in SRS document using text mining techniques. These text mining techniques cannot ensure complete correctness. This is so as this module is based on a number of pre-processing steps like tokenization, stop-words removal, stemming, etc., before the application of IG measure and Tf-Idf weighting approach could be done. Now, all these pre-processing steps use a set of English vocabulary words available online as their base, which may result in arbitrariness. However, these pre-processing steps were manually evaluated to reduce the amount of randomness and uncertainty in the accuracy and preciseness of the results so obtained. In addition to this, we have used a standard performance measure, viz, Area Under the ROC Curve (AUC) to measure the performance of the models. This measure is widely used in related research and sufficiently measures the performance of the models accurately. Thus, the proper collection of independent and the dependent variable and the use of a stable performance measure minimize the threat to construct validity to a large extent.
2. Internal Validity
"Internal validity is defined as the degree to which conclusions can be drawn about the causal effect of independent variable on the dependent variable" [64]. In this work, independent variables used are a set of pre-processed words obtained using IG measure. These independent variables are not related to each other in any way. All these words together determine the value of dependent variable (type of NFR). It is not possible to determine the causal effect of each independent variable on the dependent variable. In other words, the goal of this study is to develop prediction models for classifying NFRs into various categories rather than discovering the cause-effect relationships. Thus, the threat to internal validity does not exist in the study.
3. External Validity
External validity is defined as the extent to which the results of the study can be generalized universally. It concerns itself with finding out whether the results produced by the study are applicable in different domains or can be replicated in different scenarios for which the results are not evaluated [65]. In other words, external validity could be ensured if we could have applied the same approach on a different dataset and produced the same results. To ensure external validity, more research is needed as in this work, authors have not used the proposed models on some other datasets to identify the type of NFR. Authors have planned to take different datasets and use the same models to identify NFR specifications in their future work. In the future, we will be comparing the results of different datasets across different projects having diverse characteristics.
4. Conclusion Validity
Conclusion validity threats include all those threats which affect the conclusion of the study. In other words, all the threats which may lead to improper results or conclusions of the study are called as conclusion validity threats [65]. The authors in this study have not performed the statistical evaluation of the results using statistical tests. Thus, this leads to a conclusion validity threat. However, to provide strong conclusions, we have compared the performance of 8 ML classifiers to classify 9 types of NFRs. Very few studies in literature have used such a large number of ML classifiers. Thus, such comparison and evaluation may lead to fair conclusion, reducing the threat to conclusion validity.

In addition to this, conclusion validity threat may also occur since we had worked on the dataset consisting of less number of projects. The dataset used in this work for empirical analysis was the same dataset which was used by authors Cleland-Huang et al. [9]. This dataset consists of a set of 15 projects which were made by MS students at DePaul University, containing a total of 326 NFR descriptions. To eliminate this threat, we may include more such projects in the future study leading to stronger conclusions. Also, conclusion validity threat may occur since we have worked on a limited number of basic NFRs. There are more fine-grained NFRs listed by ISO standards 9126 and 25010 [65]. The inclusion of such NFRs may affect the results and change the performance of the models to some extent. However, the NFRs we considered covered most of the quality constraints enforced in our experimental setup. In addition to this, the dataset we used has the requirements pertaining to these basic NFR types. Due to these reasons, we did not feel the requirement to include fine grained NFRs. Inclusion of such fine grained NFRs may also increase the complexity of the work and thus, maybe included only when there is a requirement of extensive analysis.

8. Conclusions and future work

Classification of NFR descriptions into their respective types is very essential for software development meeting the basic quality determining features like security, scalability, maintainability, etc. The NFR descriptions in the software requirement specification document should therefore be analyzed carefully and mapped into their corresponding NFR types. In this paper, text mining steps have been incorporated to mine the NFR descriptions and thereby identify a set of few keywords. These keywords are the top few words which hold the essential information about NFR. Keywords help to classify the NFR descriptions using different ML techniques. Eight different ML techniques viz. J48, RF, LB, AB, MLP, RBF, Bagging and NB have been used to classify the NFR descriptions into nine types of NFR. The results which have been obtained from the study are summarized as follows:

1. With respect to each of the nine NFR types, eight prediction models have been developed corresponding to eight ML techniques.
2. The retrieval of the keywords specific to each type of NFR has been done using IG measure as the feature selection method. These keywords hold essential information about the NFR type and are used to develop prediction models by employing a suitable ML technique. Top-10 words sorted by IG measure have been selected corresponding to each of the nine models.
3. The study analyzes the performance of ML models in terms of classification and misclassification rate to determine the best model for predicting each type NFR descriptions.
4. The performance of each of these nine models is evaluated using ROC analysis. The results indicated that the performance of all the NFRs except "SE" is best predicted by NB classifier. NB gives the highest AUC of 0.97, 0.83, 0.97, 0.95, 0.81, 0.86, 0.88, and 0.77 for "A", "LF", "L", "MN", "O", "PE", "SC", and "US" types of NFR respectively. Their corresponding recall values are also high in majority of the cases. This is so because this technique has a high bias and low variance which works well for the dataset having a small size.
5. This is followed by the performance of RBF and RF technique. The AUC values of RF and RBF lies in the range of 0.67 to 0.91 and 0.72 to 0.92, respectively. Average performance has been depicted by the remaining techniques which are LB, AB, Bagging, and MLP.
6. On the contrary, J48 decision tree technique has not performed well in classifying the NFR descriptions into their respective types. This technique has shown the lowest performance (in terms of both AUC and recall) in predicting all the types of NFRs.
7. Among all the NFRs, it has been observed that most of the classifiers predicted "PE" and "A" type of NFR most accurately. On

the other hand, "US" NFR type has been predicted with lowest accuracy as there are only 3% (lowest percentage) of NFR descriptions pertaining to this NFR type in the dataset, thus giving low value of recall.

8. Overall, we concluded that the performance of NB model has performed best in predicting "MN" and "A" type of NFRs when both classification and misclassifications are taken together.
9. Also, cost-benefit analysis was conducted from which it was concluded that the cost incurred without using our proposed models is more than the cost incurred when using the proposed models.

As our future work, we intend to replicate our empirical study across more datasets of similar type, i.e., academic datasets where requirements are written by students and researchers to obtain generalized and well-formed results. Apart from this, we also intend to conduct more experiments and provide benchmarks for future performance by exploring different types of datasets like industrial datasets where requirements are written to describe or simulate industrial products or informal datasets where requirements are written by end-users as comments, reviews, posts and requests in open source communities (sourceforge.net) or written in different Apps of varying categories (books, education, games, health, lifestyle, navigation, news, productivity, travel and utilities). Moreover, we can analyze the effectiveness of the classification models by incorporating more search-based or evolutionary algorithms instead of basic ML algorithms.

References

[1] M. Glinz, "On non-functional requirements," in *Proceedings of the 15th IEEE International Requirements Engineering Conference*. Delhi, India: IEEE, 2010, pp. 21–26.

[2] M. Glinz, "Rethinking the notion of non-functional requirements," in *Proceedings of the 3rd world congress for software quality*, Munich, Germany, 2005, pp. 55–64.

[3] L. Chung, B. Nixon, E. Yu, and J. Mylopoulos, *Non-Functional Requirements in Software Engineering*. KluwerAcademic, 2000.

[4] S. Amasaki and P. Leelaprute, "The effects of vectorization methods on non-functional requirements classification," in *44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. Prague, Czech Republic: IEEE, 2018, pp. 175–182.

[5] V. Sharma, R. Ramnani, and S. Sengupta, "A framework for identifying and analysing non-functional requirements from text," in *Proceedings of the 4th International Workshop on Twin Peaks of Requirements and Architecture*. New York, USA: ACM, 2014, pp. 1–8.

[6] L. Hakim and S. Rochimah, "Oversampling imbalance data: Case study on functional and non functional requirement," in *Electrical Power, Electronics, Communications, Controls and Informatics Seminar (EECCIS)*. Batu, East Java, Indonesia: IEEE, 2018, pp. 315–319.

[7] D. Méndez-Fernández, S. Wagner, M. Kalinowski, M. Felderer, P. Mafra, A. Vetro, T. Conte, M. Christiansson, D. Greer, C. Lassenius, T. Mannisto, M. Nayebi, M. Oivo, B. Penzenstadler, D. Pfahl, R. Prikladnicki, G. Ruhe, A. Schekelmann, S. Sen, R. Spinola, A. Tuzcu, J. de la Vara, and R.Wieringa, "Naming the pain in requirements engineering," *Empirical software engineering*, Vol. 22, No. 5, 2017, pp. 2298–2338.

[8] R. Svensson, M. Host, and B. Regnell, "Managing quality requirements: A systematic review," in *36th EUROMICRO Conference on Software Engineering and Advanced Applications*. Lille, France: IEEE, 2010, pp. 261–268.

[9] J. Cleland-Huang, R. Settimi, X. Zou, and P. Solc, "Automated classification of non-functional requirements," *Requirements Engineering*, Vol. 12, No. 2, 2007, pp. 103–120.

[10] I. Hussain, L. Kosseim, and O. Ormandjieva, "Using linguistic knowledge to classify non-functional requirements in SRS documents," in *International Conference on Application of Natural Language to Information Systems*. London, UK: Springer, 2008, pp. 287–298.

[11] G. Gokyer, S. Cetin, C. Sener, and M. Yondem, "Non-functional requirements to architectural concerns: ML and NLP at crossroads," in *Proceedings of the 2008 the 3rd international conference on software engineering advances*. Sliema, Malta: IEEE, 2008, pp. 400–406.

[12] A. Rashwan, O. Ormandjieva, and R. Witte, "Ontology-based classification of non-functional requirements in software specifications: A new corpus and svm-based classifier," in *37th Annual Computer Software and Applications Conference*. Kyoto, Japan: IEEE, 2013, pp. 381–386.

[13] T. Nguyen, J. Grundy, and M. Almorsy, "Rule-based extraction of goal-use case models from text," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering.* Bergamo, Italy: ACM, 2015, pp. 591–601.

[14] A. Casamayor, D. Godoy, and M. Campo, "Identification of non-functional requirements in textual specifications: A semi-supervised learning approach," *Information and Software Technology*, Vol. 52, No. 4, 2010, pp. 436–445.

[15] W. Zhang, Y. Yang, Q. Wang, and F. Shu, "An empirical study on classification of non-functional requirements," in *Twenty-Third International Conference on Software Engineering and Knowledge Engineering*, Miami Beach, USA, 2011, pp. 444–449.

[16] J. Slankas and L. Williams, "Automated extraction of non-functional requirements in available documentation," in *1st International Workshop on Natural Language Analysis in Software Engineering.* San Francisco, USA: IEEE, 2013, pp. 9–16.

[17] P. Singh, D. Singh, and A. Sharma, "Rule-based system for automated classification of non-functional requirements from requirement specifications," in *International conference on Advances in Computing, Communications and Informatics.* Jaipur, India: IEEE, 2016, pp. 620–626.

[18] Z. Kurtanovic and W. Maalej, "Automatically classifying functional and non-functional requirements using supervised machine learning," in *25th International Requirements Engineering Conference Workshops.* Lisbon, Portugal: IEEE, 2017, pp. 490–495.

[19] A. Hindle, N.A. Ernst, M.W. Godfrey, and J. Mylopoulos, "Automated topic naming," *Empirical Software Engineering*, Vol. 18, No. 6, 2013, pp. 1125–1155.

[20] D. Ott, "Automatic requirement categorization of large natural language specifications at Mercedes-Benz for review improvements," in *International Working Conference on Requirements Engineering: Foundation for Software Quality.* Essen, Germany: Springer, 2013, pp. 50–64.

[21] M. Lu and P. Liang, "Automatic classification of non-functional requirements from augmented app user reviews," in *21st International Conference on Evaluation and Assessment in Software Engineering (EASE).* Karlskrona, Sweden: ACM, 2017, pp. 344–353.

[22] R. Deocadez, R. Harrison, and D. Rodriguez, "Automatically classifying requirements from app stores: A preliminary study," in *25th International Requirements Engineering Conference Workshops.* Lisbon, Portugal: IEEE, 2017, pp. 367–371.

[23] C. Li, L. Huang, J. Ge, B. Luo, and V. Ng, "Automatically classifying user requests in crowdsourcing requirements engineering," *Journal of Systems and Software*, Vol. 138, 2018, pp. 108–123.

[24] J. Zou, L. Xu, M. Yang, X. Zhang, and D. Yang, "Towards comprehending the non-functional requirements through developers' eyes: An exploration of stack overflow using topic analysis," *Information and Software Technology*, Vol. 84, 2017, pp. 19–32.

[25] E. Knauss, S. Houmb, K. Schneider, S. Islam, and J. Jürjens, "Supporting requirements engineers in recognising security issues," in *International Working Conference on Requirements Engineering: Foundation for Software Quality.* Essen, Germany: Springer, 2011, pp. 4–18.

[26] M. Riaz, J. King, J. Slankas, and L.Williams, "Hidden in plain sight: Automatically identifying security requirements from natural language artifacts," in *International Conference on Requirements Engineering Conference.* Karlskrona, Sweden: IEEE, 2014, pp. 183–192.

[27] R. Jindal, R. Malhotra, and A. Jain, "Automated classification of security requirements," in *2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI).* Jaipur, India: IEEE, 2016, pp. 2027–2033.

[28] R. Malhotra, A. Chug, A. Hayrapetian, and R. Raje, "Analyzing and evaluating security features in software requirements," in *Innovation and Challenges in Cyber Security.* Noida, India: IEEE, 2016, pp. 26–30.

[29] E. Knauss and D. Ott, "(Semi-) automatic categorization of natural language requirements," in *International Working Conference on Requirements Engineering: Foundation for Software Quality.* Essen, Germany: Springer, 2014, pp. 39–54.

[30] R. Maiti and F. Mitropoulos, "Capturing, eliciting, predicting and prioritizing (CEPP) non-functional requirements metadata during the early stages of agile software development," in *Proceedings of the SoutheastCon.* Fort Lauderdale, USA: IEEE, 2015, pp. 1–8.

[31] A. Mahmoud and G. Williams, "Detecting, classifying, and tracing non-functional software requirements," *Requirements Engineering*, Vol. 21, No. 3, 2016, pp. 357–381.

[32] Z. Abad, O. Karras, P. Ghazi, M. Glinz, G. Ruhe, and K. Schneider, "What works better? A study of classifying requirements," in *25th International*

Requirements Engineering Conference Workshops. Lisbon, Portugal: IEEE, 2017, pp. 496–501.

[33] L. Toth and L. Vidacs, "Study of various classifiers for identification and classification of non-functional requirements," in *International Conference on Computational Science and Its Applications.* Springer, 2018, pp. 492–503.

[34] S. Uddin, A. Khan, M. Hossain, and M. Moni, "Comparing different supervised machine learning algorithms for disease prediction," *BMC Medical Informatics and Decision Making*, Vol. 19, No. 281, 2019, pp. 1–16.

[35] Y. Dengju, J. Yang, and X. Zhan, "A novel method for disease prediction: Hybrid of random forest and multivariate adaptive regression splines," *Journal of Computers*, Vol. 8, No. 1, 2013, pp. 170–177.

[36] C. Sinclair, L. Pierce, and S. Matzner, "An application of machine learning to network intrusion detection," in *Proceedings of 15th Annual Computer Security Applications Conference (ACSAC'99).* Phoenix, AZ, USA: IEEE, 1999, pp. 371–377.

[37] E. Aleskerov, B. Freisleben, and B. Rao, "Cardwatch: A neural network based database mining system for credit card fraud detection," in *Proceedings of the IEEE/IAFE Computational Intelligence for Financial Engineering (CIFEr).* New York, NY, USA: IEEE, 1997, pp. 220–226.

[38] S. Rathore and S. Kumar, "An empirical study of some software fault prediction techniques for the number of faults prediction," *Soft Computing*, Vol. 21, 2016, pp. 7417–7434.

[39] A. Okutan and O. Yildiz, "Software defect prediction using bayesian networks," *Empirical Software Engineering*, Vol. 19, No. 1, 2014, pp. 154–181.

[40] T. Patil and S. Sherekar, "Performance analysis of Naïve Bayes and J48 classification algorithm for data classification," *International Journal of Computer Science and Applications*, Vol. 6, No. 2, 2013, pp. 256–261.

[41] J. Qinlan, *C4.5: Programs for machine Learning*, 1st ed. San Mateo, CA: Morgan Kaufmann Publishers, 1993.

[42] M. Danham and S. Sridhar, *Data mining: Introductory Advanced Topics*, 1st ed. Person Education, 2006.

[43] Y. Freund and R. Schapire, "A short introduction to boosting," *Journal of Japanese Society for Artificial Intelligence*, Vol. 14, No. 5, 1999, pp. 771–780.

[44] B. Widro and M. Lehr, "30 years of adaptive neural networks: Perceptron, madaline, and backpropagation," *Proceedings of the IEEE*, Vol. 78, No. 9, 1990, pp. 1415–1442.

[45] D. Broomhead and D. Lowe, "Multivariable functional interpolation and adaptive networks," *Complex Systems*, Vol. 2, 1998, pp. 321–355.

[46] S. Eyheramendy, D. Lewis, and D. Madigan, "On the Naive Bayes model for text categorization," in *Proceedings of the 9th International Workshop on Artificial Intelligence and Statistics*, 2002.

[47] A. McCallum and K. Nigam, "A comparison of event models for Naive Bayes text classification," in *Learning for Text Categorization*, M. Sahami, Ed. AAAI Press, 1998, pp. 41–48.

[48] R. Malhotra, *Empirical Research in Software Engineering – Concepts, Analysis and Applications*, 1st ed. India: CRC Press, 2015.

[49] M. Stone, "Cross-validatory choice and assessment of statistical predictions," *Journal of the Royal Statistical Society*, Vol. 36, No. 2, 1974, pp. 111–147.

[50] Y. Jiang, B. Cukic, and Y. Ma, "Techniques for evaluating fault prediction models," *Empirical Software Engineering*, Vol. 13, No. 15, 2008, pp. 561–595.

[51] H. He and E. Garcia, "Learning from imbalanced data," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 21, No. 9, 2009, pp. 1263–1284.

[52] T. Menzies, A. Dekhtyar, J. Distefance, and J. Greenwald, "Problems with precision: A response to "comments on 'data mining static code attributes to learn defect predictors'"," *IEEE Transactions on Software Engineering*, Vol. 33, No. 9, 2007, pp. 637–640.

[53] F. Provost and T. Fawcett, "Robust classification for imprecise environments," *Machine Learning*, Vol. 42, 2001, pp. 203–231.

[54] R. Shatnawi, "Improving software fault-prediction for imbalanced data," in *Proc. of International Conf. on Innovations in Information Technology.* Abu Dhabi, United Arab Emirates: IEEE, 2012, pp. 54–59.

[55] T. Fawcett, "An introduction to ROC analysis," *Pattern Recogn. Lett.*, Vol. 27, No. 8, 2006, pp. 861–874.

[56] F. Sebastiani, "Machine learning in automated text categorization," *ACM Computing Surveys*, Vol. 34, No. 1, 2002.

[57] T. Menzies and A. Marcus, "Automated severity assessment of software defect reports," in *IEEE International Conference on Software Maintenance (ICSM).* Beijing, China: IEEE, 2008, pp. 346–355.

[58] E. Arisholm and L. Briand, "Predicting fault-prone components in a Java legacy system," in *Proceedings of ACM/IEEE international symposium on empirical software engineering.* IEEE, 2006, pp. 8–17.

[59] R. Malhotra and M. Khanna, "An exploratory study for software change prediction in object-oriented systems using hybridized techniques," *Autom. Softw. Eng.*, Vol. 24, No. 3, 2017, pp. 673–717.

[60] L. Briand, J. Wust, and J. Daly, "Exploring the relationship between design measures and software quality in object-oriented systems," *J. Syst. Softw.*, Vol. 51, No. 3, 2000, pp. 245–273.

[61] L. Briand, J. Wust, and H. Lounis, "Replicated case studies for investigating quality factors in object oriented designs," *Empir. Softw. Eng. J.*, Vol. 6, No. 1, 2001, pp. 11–58.

[62] R. Malhotra and M. Khanna, "Threats to validity in search-based predictive modelling for software engineering," *IET Software*, Vol. 12, No. 4, 2018, pp. 293–305.

[63] A. Dean, D. Voss, and D. Draguljic, *Design and analysis of experiments*, 10th ed. New York: Springer, 1999.

[64] Y. Zhou, H. Leung, and B. Xu, "Examining the potentially confounding effect of class size on the associations between object-oriented metrics and change-proneness," *IEEE Transactions on Software Engineering*, Vol. 35, No. 5, 2009, pp. 607–623.

[65] R. Harrison, S. Counsell, and R. Nithi, "Experimental assessment of the effect of inheritance on the maintainability of object-oriented systems," *Journal of Systems and Software*, Vol. 52, No. 2–3, 2000, pp. 173–179.

A. Appendix

Table A1. Performance evaluation of random forest and J48 decision tree techniques

S. No.	Type of NFR	Random Forest			J48		
		AUC	Sens (%)	CutOff	AUC	Sens (%)	CutOff
1	A	0.91	85.0	0.02	0.65	60.0	0.02
2	LF	0.82	73.0	0.05	0.58	64.9	0.08
3	L	0.83	69.2	0.05	0.48	30.8	0.02
4	MN	0.80	80.0	0.00	0.42	33.3	0.04
5	O	0.79	67.2	0.12	0.58	52.5	0.12
6	PE	0.84	77.4	0.05	0.75	66.0	0.06
7	SC	0.67	66.7	0.01	0.53	57.1	0.05
8	SE	0.80	73.4	0.06	0.65	60.9	0.12
9	US	0.76	68.3	0.09	0.48	38.1	0.18

Table A2. Performance Evaluation of Bagging and Naïve Bayes techniques

S. No.	Type of NFR	Bagging			Naïve Bayes		
		AUC	Sens (%)	CutOff	AUC	Sens (%)	CutOff
1	A	0.77	75.0	0.03	0.97	90.0	0.10
2	LF	0.80	75.7	0.07	0.83	75.7	0.01
3	L	0.75	69.0	0.02	0.97	92.3	0.01
4	MN	0.81	73.3	0.02	0.95	93.3	0.06
5	O	0.78	70.5	0.10	0.81	67.2	0.12
6	PE	0.80	75.5	0.07	0.86	83.0	0.01
7	SC	0.85	85.7	0.04	0.88	85.7	0.01
8	SE	0.80	71.9	0.09	0.81	73.4	0.03
9	US	0.74	68.3	0.10	0.77	68.3	0.06

Table A3. Performance Evaluation of MLP and RBF network techniques

S. No.	Type of NFR	MLP			RBF		
		AUC	Sens (%)	CutOff	AUC	Sens (%)	CutOff
1	A	0.67	60.0	0.00	0.81	75.0	0.01
2	LF	0.80	73.0	0.04	0.80	67.6	0.06
3	L	0.76	76.9	0.01	0.92	92.3	0.05
4	MN	0.62	60.0	0.01	0.85	80.0	0.01
5	O	0.80	68.9	0.10	0.75	65.6	0.13
6	PE	0.84	77.4	0.06	0.81	71.7	0.06
7	SC	0.66	61.9	0.02	0.80	71.4	0.02
8	SE	0.85	79.7	0.08	0.83	81.3	0.08
9	US	0.76	66.7	0.09	0.72	61.9	0.09

Table A4. Performance Evaluation of Logitboost and Adaboost techniques

S. No.	Type of NFR	Logitboost			Adaboost		
		AUC	Sens (%)	CutOff	AUC	Sens (%)	CutOff
1	A70.87	85.0	0.05	0.73	70.0	0.02	
2	LF	0.79	70.3	0.05	0.72	64.9	0.06
3	L	0.75	76.9	0.01	0.72	69.2	0.01
4	MN	0.88	86.7	0.04	0.79	73.3	0.01
5	O	0.80	67.2	0.12	0.78	75.4	0.14
6	PE	0.82	77.4	0.09	0.83	77.4	0.06
7	SC	0.79	76.0	0.02	0.76	71.4	0.02
8	SE	0.78	70.3	0.08	0.74	68.8	0.09
9	US	0.73	61.9	0.11	0.66	57.1	0.11

e-Informatica Software Engineering Journal, Volume 15, Issue 1, 2021, pages: 115–132, DOI 10.37190/e-Inf210106

Software Deterioration Control Based on Issue Reports

Omid Bushehrian*, Mohsen Sayari*, Pirooz Shamsinejad*

*Department of Computer Engineering and Information Technology, Shiraz University of Technology, Shiraz, Iran

bushehrian@sutech.ac.ir, sayari.mohsen@gmail.com, p.shamsinejad@sutech.ac.ir

Abstract

Introduction: Successive code changes during the maintenance phase may cause the emergence of bad smells and anti-patterns in code and gradually results in deterioration of the code and difficulties in its maintainability. Continuous Quality Control (QC) is essential in this phase to refactor the anti-patterns and bad smells.

Objectives: The objective of this research has been to present a novel component called Code Deterioration Watch (CDW) to be integrated with existing Issue Tracking Systems (ITS) in order to assist the QC team in locating the software modules most vulnerable to deterioration swiftly. The important point regarding the CDW is the fact that its function has to be independent of the code level metrics rather it is totally based on issue level metrics measured from ITS repositories.

Methods: An issue level metric that properly alerts us of bad-smell emergence was identified by mining software repositories. To measure that metric, a Stream Clustering algorithm called ReportChainer was proposed to spot Relatively Long Chains (RLC) of incoming issue reports as they tell the QC team that a concentrated point of successive changes has emerged in the software.

Results: The contribution of this paper is partly creating a huge integrated code and issue repository of twelve medium and large size open-source software products from Apache and Eclipse. By mining this repository it was observed that there is a strong direct correlation (0.73 on average) between the number of issues of type "New Feature" reported on a software package and the number of bad-smells of types "design" and "error prone" emerged in that package. Besides a strong direct correlation (0.97 on average) was observed between the length of a chain and the magnitude of times it caused changes to a software package.

Conclusion: The existence of direct correlation between the number of issues of type "New Feature" reported on a software package and (1) the number of bad-smells of types "design" and "error prone" and (2) the value of "CyclomaticComplexity" metric of the package, justifies the idea of Quality Control merely based on issue-level metrics. A stream clustering algorithm can be effectively applied to alert the emergence of a deteriorated module.

Keywords: Code Smells, Issue report, maintainability, document classification

1. Introduction

Mining open-source software repositories and particularly bug repositories managed by Issue Tracking Systems (ITS) such as BugZilla [1] and Jira [2] has recently attracted much attraction among the software research community. Many researchers have studied the possible correlations among stored knowledge in bug repositories and software quality aspects. By interpreting the number of reported bugs on a specific software version as quality indicator of that version, some works have focused on the correlation among the number of previously reported bugs (and hence changes) and the number of bugs in future release [3, 4]. Maintainability has been another quality aspect of interest and some research works have been dedicated to find meaningful correlations

Submitted: 27 October 2020; Revised: 12 June 2021; Accepted: 17 June 2021; Available online: 25 August 2021

between software maintainability and defect metrics extracted from ITS repositories [5].

The maintainability of a software product is usually defined as its readiness to accept successive modification very easily and with minimum effort [6]. These modifications are due to requested new features, reported bugs, performance difficulties or adapting the software to a new environment and are carried out by the maintenance team. Some studies have argued, particularly in open-source software, neither time lags in fixing bugs nor the distribution of bugs can be considered as the direct representative metrics of maintainability [7]. On the other hand, many research works in the field of software maintainability have proved that the maintainability of software after delivery is significantly dependent to its original design quality [8]. As a result, building the predictive models to assist the designers to better assess the maintainability of their future code based on the current design quality metrics at very first stages of development life cycle has attracted much attention in this research field [8–10]. In addition to the original design of the software, the quality of code modifications during the maintenance phase also matters and affects the maintainability of the software and its future defects number and severity [11]. It is now widely accepted that the maintainability of a software product could be measured by: (1) the number of residual or emerged faults in the maintenance phase (corrective maintainability), (2) the extent to which the code is understandable (adaptive maintainability) which is greatly affected by the degree of exploiting the software design patterns and best practices in development process and (3) the extent to which it is modifiable (perfective maintenance) which is again affected by the amount of anti-patterns and bad-smells [12] in the code [5, 7, 8].

Successive changes made by the developers may cause the emergence of bad smells in code and gradually result in code deterioration and lowering the code maintainability indirectly. Hence continuous Quality Control (QC) is essential in this phase to avoid formation of anti-patterns and bad smells in code. However this QC process could be very tedious or even ineffective task without the help of assisting tools. Since the change requests (and issue reports) are recorded and tracked in ITSs, we believe that these software tools could be equipped with a recommender component that enables the QC team to spot software modules (packages) that are most vulnerable to early deterioration and to put those modules in their priority list for code inspection and refactoring. We call this component CDW (Code Deterioration Watch) and it is capable of reporting the packages that need immediate quality check by QC team to see if the refactoring is necessary. The fundamental requirement in designing CDW is that it should be able to estimate the deterioration level of software modules using issue metrics rather than code metrics. This is the key difference between this research and similar studies where CDW does its function merely by relying on ITS repository data and independent of code repository. We believe that this ability is so important due to the fact that code QC has to be carried out based on a priority list in specific time intervals and this priority list is provided by CDW which continuously analyzes incoming issue streams efficiently. The high priority modules for refactoring are those that have become deteriorated and hence error-prone due to the successive changes performed not in compliance with best-practices and standard design patterns.

The function of CDW is simply based on this general hypothesis: “modules absorbing higher number of issues are those with higher number of (born) code smells and are the hot spots for refactoring”. We studied a variety of open-source software repositories from Apache [13] and Eclipse [14] to answer these research questions:

RQ1: Can the QC team evaluate the quality level of the code, which is under successive changes during the maintenance phase, only by observing issue-related metrics such as the number of reported issues on a software package? Is the type of reported issues important in this evaluation?

RQ2: Is there an effective stream clustering method to categorize incoming sequence of issue reports such that a bloated category be truly interpreted as the concentration of frequent changes on a specific software package and hence be reported as possible point of deterioration?

The rest of this paper is organized as follows: Section 2 explains the related works, Section 3 explains the concept of "code deterioration" and its relation to bad-smells, Section 4 presents the proposed continuous Quality Control model, Section 5 reports the results of experiments, the discussion and justification of the results are presented in section 6, Section 7 provides the threats to the validity of the study and finally Section 8 concludes the paper.

2. Related works

In the field of software maintainability, there are many research works dedicated to the maintainability prediction of software based on the knowledge collected in early stages of SDLC [8]. In [9] the superiority of the dynamic metrics over static metrics for maintainability prediction is studied. They concluded that the dynamic metrics outperform the static ones regardless of the machine learning algorithm used for prediction.

Extracting useful knowledge from bug repositories to estimate quality and maintainability of an open-source software has been the subject of many studies. In [3] authors concluded that the number of bugs, as software quality indicator, in the ith release has no significant correlation with the change size of its previous release. In [4] a class level quality assurance metric named Qi was defined and the correlation of the number of defects and Qi was analyzed but they observed no significant correlation among them. In [7] the maintainability of some open- source software was studied empirically and they reported that neither the time lag of reported bugs nor the distribution of bug reports can represent the maintainability indicator.

In [15] a recommender system to advice developers to avoid bad smells and apply quality practices during the programming is presented. They have built a quality model which is continuously updated based on the reported issues and the (detected) bad smells that have triggered the issue. The SZZ algorithm [16] which identifies the root cause of an issue has been applied by this study to track down the earliest change that has given birth to the exception. Subsequently the bad smells detected in code snippet identified as the root cause of the exception is related to the issue. Machine learning algorithm have been applied to build the quality model. In [17] to improve the software quality metrics and remove bad smells a multi-objective evolutionary algorithm is proposed by which the best sequence of refactoring activities is sought in a large search space of possible solutions. To obtain this a predicting model based on time series is applied to estimate the impact of each sequence of refactoring actions on the future software quality. In [10] thirty different software quality prediction models were studied. Using two standard datasets they concluded that regression and LWL outperformed others.

There are some studies on the relationship between the amount of code-smells and the bug proneness level of the software. In [18] it was shown that adding smell-related features (code--smell intensity) to the bug prediction models could improve the accuracy of the prediction models. The impact of presence of anti-patterns on the change and fault-proneness of the classes has been investigated in [19]. The results confirmed that the classes involving anti-patterns are more change and fault-prone. A Systematic Literature Review has been conducted and reported in [20] on the impact of code-smells on software bugs. The adverse effects of bad architectural decisions (architectural smells) on the maintainability of the software in terms of number of forthcoming issues and increased maintenance efforts have been studied in [21].

The classification or clustering of bug reports has also been the subject of some previous studies. The classification of bug reports are used to predict a variety of factors regarding them. For instance in [23] a two-phased classifier has been proposed to predict the files likely to be fixed using the bug report textual description. In [24] a clustering method based on EM (Expectation Maximization) and X-means has been proposed to categorize bug reports according to their subject similarities. They have used topic modeling to vectorize bug reports and subsequently applied a labeling algorithm to characterize each cluster.

There are also some studies on refactoring prioritization. In [25] a machine learning approach for classification of code smell severity to prioritizing the refactoring effort has been presented. They have reported a relatively high correlation between the predicted and actual severity by modeling the problem as an ordinal classification problem. In [26] a semi-automated refactoring prioritization method to assist developers has been presented. They have applied a combination of three criteria: past modifications history, the relevance of the smell to the architecture and the smell type to rank the refactoring activities.

The contribution of this paper is to propose a novel model of Software Quality Control which enables the QC team to judge about the internal quality of software, constantly changed by developers, without the need of source code analysis and merely by monitoring the incoming issue reports and their sequence. To this end first, a thorough correlation analysis between the code quality metrics (the number of bad-smells) and issue-level metrics (issue absorption rate) has been conducted. Subsequently a new stream clustering method is presented to effectively categorize incoming sequence of issue reports such that a bloated category truly indicates the existence of a software package with high issue absorption rate. It is important to note that in contrast to the previous studies on refactoring prioritization, the proposed method uses the issue-level metrics to find the top-module to refactor without needing to analyze or access the source code.

3. Code deterioration and bad-smells

There are a variety of bad-smell and anti-patterns introduced in the literature that may emerge in the code gradually due to the subsequent changes made by the development team [12]. The PMD static source analyzer [22] has presented a very good categorization of bad-smells in its documentations (Code Style, Design, Error-prone, Doc-

Table 1. Bad-smells according to the categorization presented in [22]

Category	Some Bad smells in this category	Example

Design
(46 bad-smells)
	AbstractClassWithoutAnyMethod,
	ClassWithOnlyPrivateConstructorsShouldBeFinal,
	CouplingBetweenObjects, CyclomaticComplexity
	DataClass, ExceptionAsFlowControl
	ExcessiveClassLength, GodClass,
	ImmutableField, LawOfDemeter,
	LogicInversion, LoosePackageCoupling,
	NPathComplexity,…
		DataClass:
		
		public class DataClass {
		public int bar = 0;
		public int na = 0;
		private int bee = 0;
		public void setBee(int n) {
		bee = n;
		}
		}

Error Prone
(98 bad-smells)
	AssignmentInOperand,
	AssignmentToNonFinalStatic,
	AccessibilityAlteration,
	AssertAsIdentifier,
	BranchingStatementAsLastInLoop,
	CallingFinalize,
	CatchingNPE, CatchingThrowable,
	DecimalLiteralsInBigDecimalConstructor,
	DuplicateLiterals, EnumAsIdentifier,
	FieldNameMatchingMethodName,
	FieldNameMatchingTypeName,
	InstanceofChecksInCatchClause,
	LiteralsInIfCondition,
	LosingExceptionInformation,…
		AssignmentInOperand:
		
		public void bar() {
		int x = 2;
		if ((x = getX()) == 3) {
		System.out.println("3!");
		}
		}

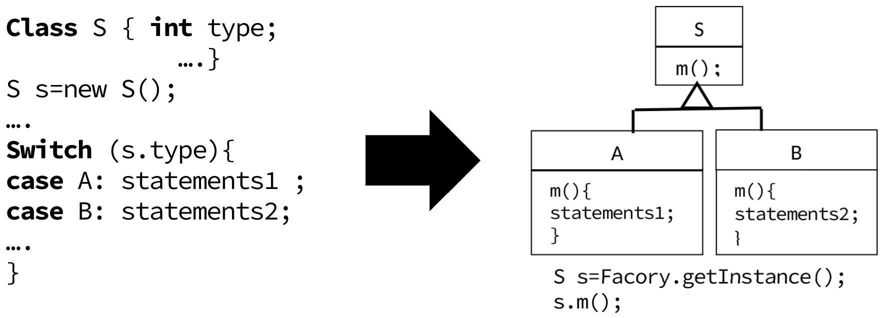


Figure 1. Replacing switch/case statements with polymorphism to reduce the "CyclomaticComplexity" value

Table 2. Issue categories [2]

Issue Type	Example	Issue Ref.	Studied in this paper
Bug	New version of Java 11 seems does not work well	NETBEANS-5636	Yes
New Feature	Have python-archives also take tar.gz	FLINK-22519	Yes
Improvement	Upgrade Kotlin version in Kotlin example to 1.4.x	BEAM-12252	No
Task	Remove landmark directories from web and shim	YUNIKORN-662	No
Sub-Task	Optional removal of fields with UpdateRecord	NIFI-8243	No
Test	Remove some Freon integration tests	HDDS-5160	No
Umbrella	Add SQL Server functions	TRAFODION-3146	No
Documentation	SQL DataFrameReader unescapedQuoteHandling parameter is misdocumented	SPARK-35250	No

umentation, Multithreading, Performance and Security) and among those we have focused on "Design" and "Error-Prone" categories since they refer to much more general forms of anti-patterns compared to other categories that contain more particular subjects. Some bad-smells in either categories along with examples are presented in Table 1.

Apart from these bad-smells we have also analyzed the method-level complexity using two well-known metrics: "Cyclomatic Complexity" and "NpathComplexity" as their high values are very good indicators of missing fundamental design patterns such as strategy, composite, proxy, adapter and many others that are based on polymorphism rather than conditional logics. The former is the number of decision points in the code and the latter is the number of full paths from the beginning to the end of the block of a method [27]. The refactoring practice corresponding to the high method complexity (measured using "CyclomaticComplexity" and "NpathComplexity") is illustrated in Figure 1.

These aforementioned bad-smells are code--level structures (or measures) that alert us of deteriorated code. However the aim of this research is to investigate issue-level metrics that does the same without relying on the code repository and hence make the QC activity possible by merely watching the issue repository. In the Analysis section (Section 5) it will be argued that the number of issues of type "New Feature" reported on a software package is an effective issue-level metric to inform the QC team of the code deterioration extent. See Table 2 for different issue categories and those that are involved in this study.

4. Code deterioration assessment model

The proposed model which incorporates the QC component, called CDW (Code Deterioration Watch) into the ITS, is illustrated in Figure 2. This component keeps track of the reported issues and categorizes them incrementally to detect relatively long sequences of related reports as a sign of possible code deterioration. By analyzing the

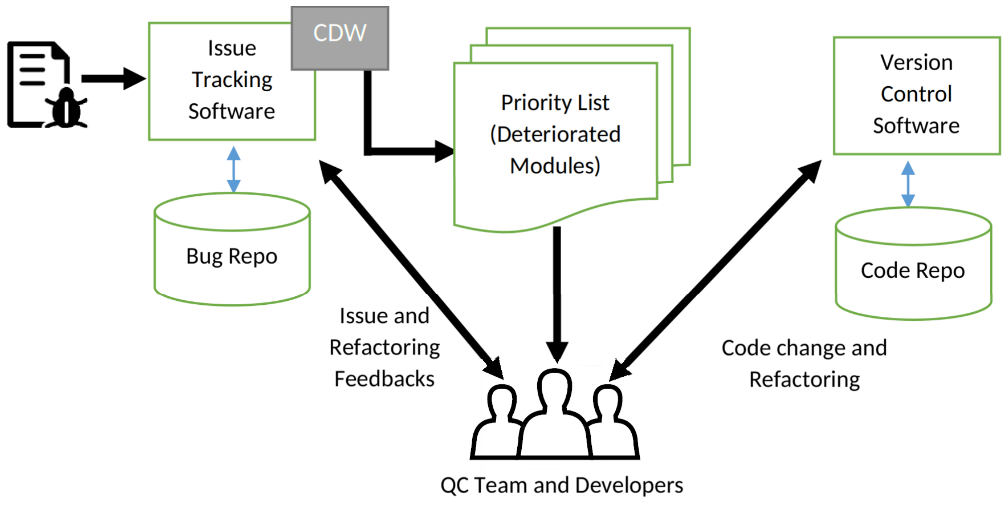


Figure 2. The code quality control based on issue monitoring

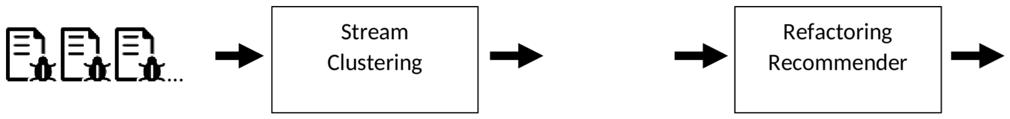


Figure 3. The CDW sub-components

long sequences of reports, CDW prioritizes the software modules to be scrutinized by the QC team for refactoring activities. There are two main sub-components of CDW working together to produce the final refactoring recommendation as shown in Figure 3. The Stream Clustering that splits the incoming sequence of issue reports into a set of chains and the Refactoring Recommender.

Ideally CDW should have the capability of notifying the QC team of the packages that are being changed frequently far more than others as they are code segments very likely to get deteriorated soon and need immediate attention for refactoring (this hypothesis will be verified in Section 6). Even if CDW be able to alert the QC team of existence of such packages without identifying them, it would be very helpful yet. Note that identifying these packages accurately is possible by analyzing the code repository at the later time, however CDW is part of the ITS (and not the version control) and it is supposed to give us insights about the evolution of the software merely by monitoring the incoming sequence of issue reports (issue metrics rather than code metrics).

As the Stream Clustering sub-component clusters the incoming issue reports into chains the idea is to spot Relatively Long Chains (RLC) of incoming issue reports as they tell the QC team that a concentrated point of successive changes has emerged in the software (this correlation will be discussed in Section 6). RLCs are those that their lengths (the number of issue reports in the chain) exceed the average chain length significantly (as much as threshold ß). Note that that if all chains are long, none of them is considered as RLC due to the fact that RLC concept is based on significant size difference in a group and not the absolute size itself.

We define the “target” of each chain as the package which is expected to incur majority of the changes as the issue reports in that chain are being resolved. The “target hit number” of a chain is also defined as the expected change frequency of the chain target. For instance in a chain of three reports: R1, R2 and R3 labeled with {P1, P2, P3}, {P1, P3} and {P1}, respectively, as the packages to be changed, the chain target is P1 and its hit number is three.

Obviously a chain with a relatively high target hit number is telling us that a package (the target) is being changed frequently far more than others. In Section 5 it will be verified that the chain size is a good metric to identify chains with a relatively high target hit number.

4.1. Stream Clustering

A simple stream clustering called "Report-Chainer" is presented here that splits the sequence of issue reports into chains based on their similarity to the previously formed chains. A split threshold controls formation of a new chain. First all documents are vectorized using Tf-Idf method [28] and cosine similarity is applied to compute the similarity of the current issue report with previous ones. If the similarity values are less than the split threshold then it is added to new chain otherwise it is added to the most similar chain (see Algorithm 1).

Vectorizing document d in a collection of N documents using Tf-idf method consists of two steps: first, the frequency of each term t in d is counted (denoted $tf_{t,d}$ and then the value of $tf_{t,d}$ is scaled using the idf_t (inverse document frequency) value:

$$idf_t = \frac{N}{df_t} \tag{1}$$

Where df_t is the number of documents in the collection containing term t. The value of Tf-idf corresponding to term t in document d is calculated using the following formula [28]:

$$Tfidf_{t,d} = tf_{t,d} \times idf_t \tag{2}$$

In fact the Tf-idf method assigns lower weights to the terms with no or very little discriminating power in a document. To compute the similarity of two documents $d1$ and $d2$, vectorized using the Tf-idf method, the cosine similarity has been applied [28]:

$$\mathrm{sim}(d1, d2) = \frac{(V(d1).V(d2))}{(|V(d1)| \times |V(d2)|)} \tag{3}$$

Where $V(d1)$ and $V(d2)$ denote the vector representation of $d1$ and $d2$, respectively, obtained using the Tf-idf method. The advantage of using the cosine similarity method over the ordinary method of computing the vector distances is that the cosine similarity formula is insensitive to the documents' length.

In the next section it will be shown that there is a significant linear correlation between the length of a chain and the magnitude of times that the chain target has been changed. Accordingly the longer chains are good candidates for QC attention. We will also show that, at least for 10 cases studies, not only can a split threshold be found to result a high correlation value but also this value is bounded.

4.2. Refactoring Recommender

The Refactoring Recommender component takes a set of chains C as its input and produces the recommendation by selecting, from the candidate list, those chains whose size differences with the average chain size are greater than threshold β (line 13 in Algorithm 2). These chains are consid-

Algorithm 1. Sequence Clustering Algorithm

```
maxSim = 0
chainNum = –1
thisVec = **TF_IDF**(r)
**for** Each *vec* in $C$ **do**
    Sim = cosineSim(thisVec, *vec*)
    **if** Sim > maxSim **then**
        maxSim = sim
        chainNum = *vec*.chainNum
    **end if**
**end for**
**if** maxSim < t **then**
    chainNum = C.newChain()
**end if**
C.add(thisVec, chainNum)
```

Algorithm 2. Refactoring recommender algorithm

```
1:  **Algorithm** RefactoringRecommender(**Chains** $C$): **List**
2:  List r = $\emptyset$
3:  Package p
4:  int issueCnt = 0
5:  int chainCnt = 0
6:  **for** Each chain $c$ in $C$ **do**
7:      **if** c. numberOfNotVisitedIssues()> $\alpha$ **then**
8:          issueCnt += $c$.numberOfNotVisitedIssues()
9:          chainCnt += 1
10:     **end if**
11: **end for**
12: **for** Each chain $c$ in $C$ **do**
13:     **if** $c$.numberOfNotVisitedIssues() − (issueCnt / chainCnt)) > $\beta$ **then**
14:         p = targetAnalyze($c$)
15:         r.add(p)
16:     **end if**
17: **end for**
18: return r
19:
20: **Algorithm** On_RefactoringCompleted(**chain** $c$)
21: **for** Each Issue $s$ in $C$ **do**
22:     $s$.visited = True
23:     r.add(p)
24: **end for**
```

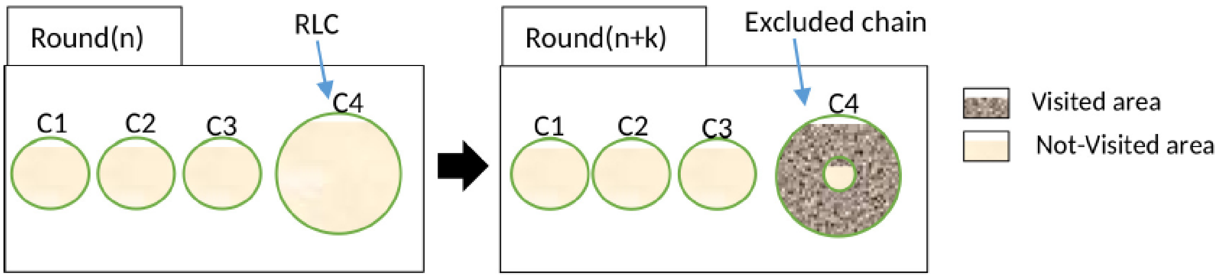


Figure 4. Issues have been categorized in four chains; In round n of recommendation, candidate set is $\{C1, C2, C3, C4\}$ and $C4$ is detected as RLC; during next k rounds $C4$ is excluded from the candidate set and no RLC is reported by the algorithm

ered as RLCs. Subsequently the targetAnalyze() function determines the target package of each RLC (line 14). This function can be carried out using supervised machine learning methods as presented in [23] or be done manually by experts. As soon as a recommended refactoring is performed by the QC team, all issues in the corresponding chain are excluded from the subsequent rounds of process by labeling them as "Visited" (line 20–24). Moreover the chain is excluded from the candidate list in the subsequent rounds of process until the number of "Not-Visited" issues reaches threshold α (line 7). This prevents the algorithm to falsely identify most of the candidate chains as RLCs since recently recommended chain has a few number of "Not Visited" issues and hence moves down the average chain size significantly (see Figure 4). A reasonable value for a could be the average chain size of the current candidate list.

5. Analysis

The objective of our experiments was first to study the correlation between the issue-level metrics and code-level metrics and second to evaluate the proposed *RepChainer* algorithm. There are two issue-level metrics to study: Bug Absorption Degree (BAD) and Feature Absorption Degree (FAD) defined as the number of bugs and the number of new features reported on a software module in a period of time respectively. The code-level metrics are "CyclomaticComplexity", "NpathComplexity" and the number of detected "Design+ErrorProne" bad-smells as explained in Section 3. The software module granularity was chosen to be the package (a set of related classes). The rationale behind choosing the package granularity is to have more specialized task assignment (i.e. refactoring tasks) to developers. Since packages are often focused on specific subjects they can easily be assigned to developers who are specialized in the respective area to refactor all the classes of them. Moreover another set of experiments were conducted to evaluate the accuracy and usefulness of the stream clustering algorithm used in detecting bloated categories(RLCs) of issue reports.

5.1. Dataset

To do the experiments, a dataset with the schema shown in Figure 5 of medium and large scale open-source software repositories from Apache and Eclipse were created. Table 3 summarizes

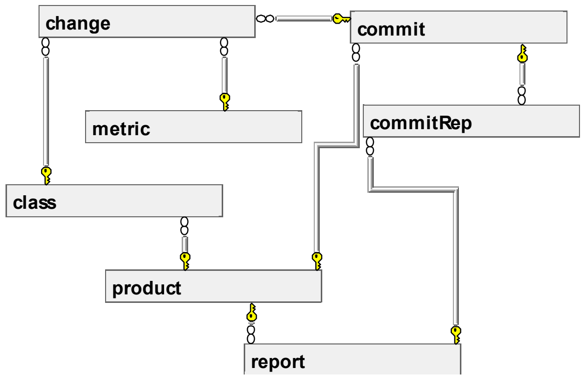


Figure 5. Dataset schema

the products metadata. For each product, its Git [29] repository was cloned and analyzed using a program written with Node.Js. This program extracted all the commits and their associated objects from the repository using the isoMorphic-Git [30] library. During the analysis phase the code metrics: "CyclomaticComplexity", "NpathComplexity" and the number of detected "Design+ErrorProne" bad-smells were also measured using the PMD source code analyzer [22].

The Javascript code snippet to invoke PMD program is shown in Listing 1. A custom or built-in rule set has to be passed to PMD to analyze the source code accordingly. For detecting "Design+ErrorProne" bad-smells the built-in rule sets: category/java/design.xml and cate-

Table 3. Products metadata

Product	License	Language	No. of Classes	No. of packages	No. of reported issues	No. of analyzed commits	ITS
Cloudstack	Apache	Java, Python	16100	639	3451	5565	JIRA
Geode	Apache	Java	27277	961	3805	6565	JIRA
Spark	Apache	Java, Python, Scala	8494	490	13289	15044	JIRA
Camel	Apache	Java	37702	2393	9655	18299	JIRA
Geronimo	Apache	Java	13163	846	2127	3115	JIRA
Hadoop	Apache	Java	27045	1168	3671	15902	JIRA
Hbase	Apache	Java	8255	261	9445	11164	JIRA
Myfaces	Apache	Java	2805	219	1594	2640	Bugzilla
4diac.ide	Eclipse	Java	1598	255	1126	504	Bugzilla
Acceleo	Eclipse	Java	896	210	969	520	Bugzilla
Common	Eclipse	Java	1496	177	105	287	Bugzilla
App4mc	Eclipse	Java	1370	68	131	186	Bugzilla

```
const cp = require('child_process');
cp.execSync("e:\\pmd\\bin\\pmd.bat
−dir e:\\pmd\\input_cloudstack
−format xml −R e:\\pmd\\ru.xml >
e:\\pmd\\output_cloudstack\\out.xml");
cp.execSync("e:\\pmd\\bin\\pmd.bat
−dir e:\\pmd\\input_cloudstack
−format xml −R category/java/design.xml,
category/java/errorprone.xml,>
e:\\pmd\\output_cloudstack\\out.xml");
```

Listing 1. Javascript code snippet to invoke PMD program using a custom ruleset: ru.xml (top) and the predefined "design" and "error prone" rule sets (bottom)

gory/java/errorprone.xml were used. To measure "CyclomaticComplexity" and "NPathComplexity" metrics a custom rule set ru.xml was used as listed in Listing 2.

Due to the size of repositories the analysis took over a week to complete on a cluster of core-i7 PCs during which the analyzer program was running round the clock. To the best of our knowledge such an integrated dataset of issues, measured code metrics and bad-smells and metadata of all commits on Eclipse and Apache products has not been published elsewhere and we are working to make it online soon.

The issues reported on products were imported into "report" table from the respective ITS. Apache-Jira and Eclipse-Bugzilla ITSs are accessible at [31] and [32] respectively.

5.2. Correlation Analysis

Assume that N_i and B_i are the number of resolved issues of type "New Feature" and "Bug" on package P_i, respectively, (issues for which P_i was not modified are excluded) and $I_i = N_i \bigcup B_i$ denotes the number of all resolved issue types on P_i. Moreover $\Delta b_{i,m}$ denotes the variation of code metric m during the successive changes of P_i:

$$\Delta b_{i,m} = m_{\max,i} - m_{\text{init},i}$$
$$m \in \left\{ \begin{matrix} \text{"CyclomaticComplexity"}, \\ \text{"NpathComplexity"}, \\ \text{"Design + ErrorPron"} \end{matrix} \right\} \tag{4}$$

where $m_{\max,i}$ and $m_{\text{init},i}$ are the maximum and initial values of code metric m measured over all

```
<?xml version="1.0"?>
<ruleset name="Custom␣Rules"
xmlns="http://pmd.sourceforge.net/ruleset/2.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema−instance"
xsi:schemaLocation="http://pmd.sourceforge.net/ruleset/2.0.0
␣␣␣https://pmd.sourceforge.io/ruleset_2_0_0.xsd">
 <description>
  My custom rules
 </description>
 <rule ref="category/java/design.xml/CyclomaticComplexity">
  <properties>
   <property name="classReportLevel" value="1" />
   <property name="methodReportLevel" value="1" />
   <property name="cycloOptions" value="" />
  </properties>
  </rule>
 <rule ref="category/java/design.xml/NPathComplexity">
  <properties>
   <property name="reportLevel" value="1" />
  </properties>
 </rule>
</ruleset>
```

Listing 2. PMD custom rule-set applied to measure "CyclomaticComplexity" and "NPathComplexity" metrics

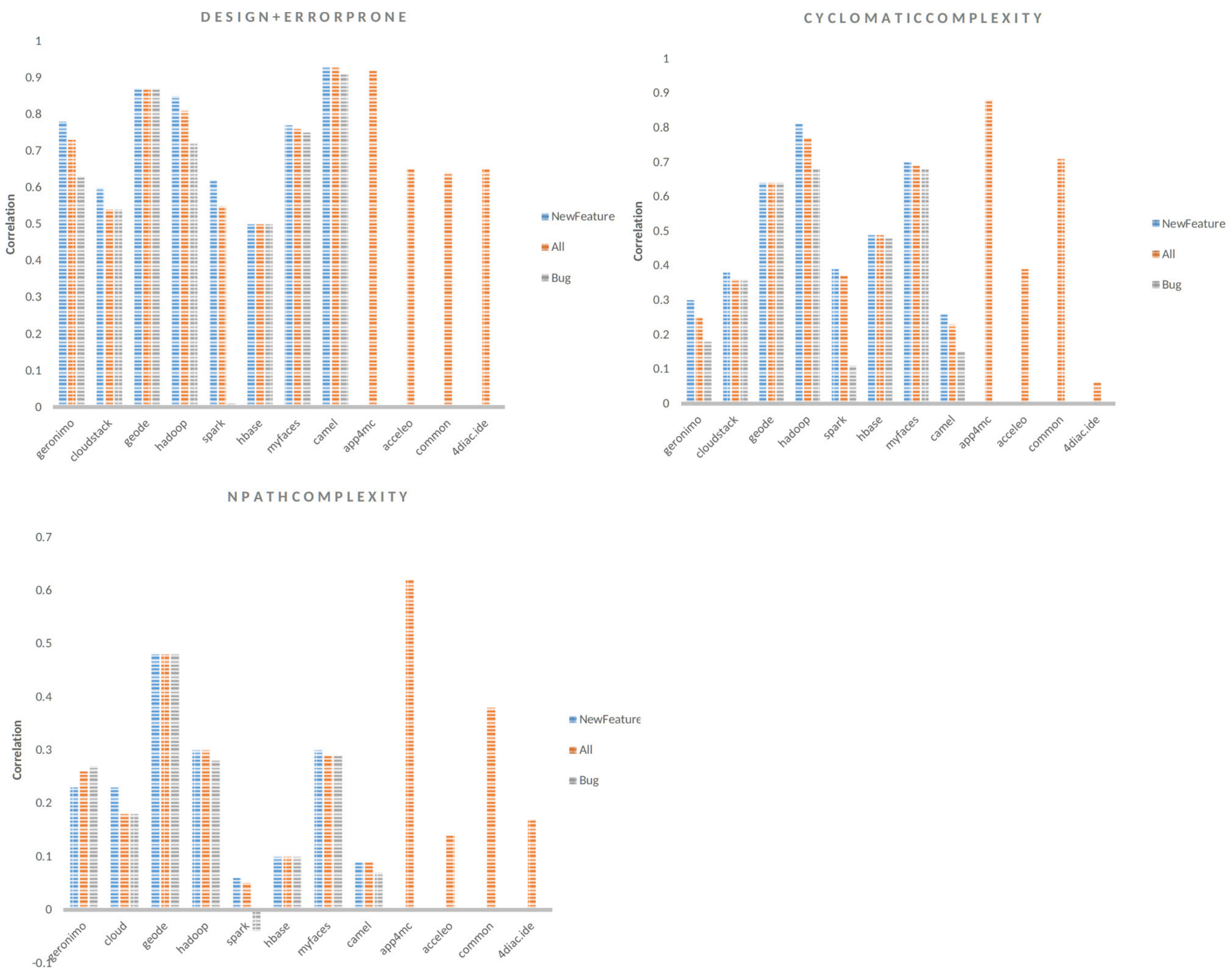


Figure 6. The Pearson correlation analysis on 12 products: (top left): "Design+errorProne" bad smells, (top right) "CyclomaticComplexity" metric and (bottom) "NpathComplexity" metric

variations of classes in P_i:

$$m_{max,i} = \sum^{C \in P_i} \text{Max}_{\text{versions}(c)}\{\text{val}(m_C)\} \quad (5)$$

where versions(c) is the set of all successive versions of class C stored in the Git repository due to subsequent changes made by committers and val(m_C) denotes the measured value of metric m on class C. Likewise:

$$m_{\text{init},i} = \sum^{C \in P_i} \{\text{val}(m_{C0})\} \quad (6)$$

where C_0 denotes the initial version of class C in Git repository. The objective of the correlation analysis is to examine the random variable pairs: $(\Delta b_i, N_i)$, $(\Delta b_i, I_i)$ and $(\Delta b_i, B_i)$ for existence of significant linear correlation. The correlation was analyzed using the Pearson test and the results are illustrated in Figure 6. The main reason that we considered the difference between the max and the initial metric values has been to obtain the added number of smells to the base value during the successive commits. In other words the increase in the number of smells matters most in our study.

For the Eclipse products the separate correlation analysis for BAD and FAD metrics was not possible as the classification of issue reports into "Bug" and "New Feature" is not provided by the Bugzilla ITS. The highest correlation was observed between the "Design+ErrorProne" code metric and the FAD issue metric. With respect to

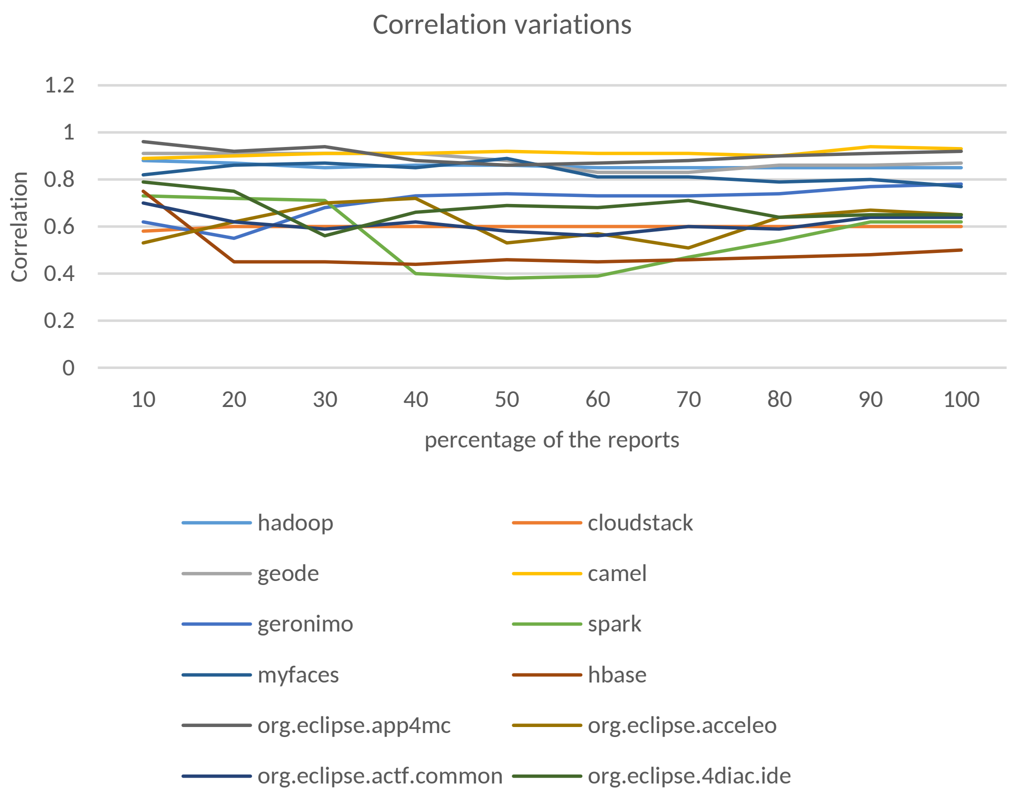


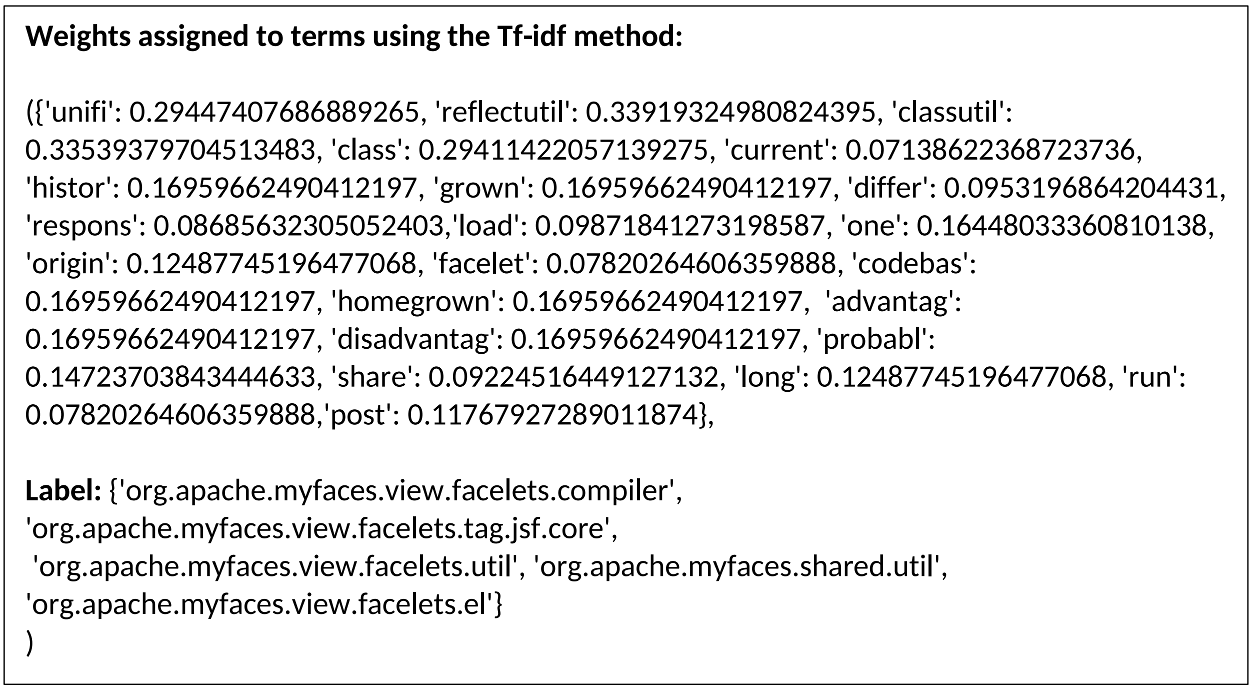
Figure 7. The correction between the number of reports of type "New Feature" and the number of bad-smells of type: "Design+ErrorProne" computed for different dataset sizes

the "CyclomaticComplexity" and "NpathComplexity" metrics, as the indicators of the extent to which polymorphism is missing in the design, good correlation is observed between the "CyclomaticComplexity" metric and the FAD metric. The variation of the correlation values (between the FAD and "Design+ErrorProne" metrics) over the maintenance period has also been studied by stepwise correlation analysis using different repository sizes. As shown in Figure 7 the fractional correlation analysis has been performed on data set sizes ranging from 10% to 100% of the issue reports in the repository. The results of the above analysis will be discussed in Section 6.

5.3. Stream Clustering Analysis

To evaluate the proposed RepChainer algorithm, issue reports from 10 Apache products were analyzed. Each report is associated with a set of commits and the set of packages changed by the respective commits is used as the report label. A sample report and its label is shown in Figure 8.

For each product three streams of reports: small, medium and large size were prepared to evaluate the ReportChainer algorithm. The small size stream was used for finding the best value for split threshold (learning set) and the two other streams used for evaluation. A Python program iterates through threshold values in the range [0.1 0.8] and for each threshold the chains are created and the "chain lengths" and their corresponding "target hit numbers" are extracted to compute the correlation value. The "target hit number" of each chain is determined based on the labels of reports in the chain; for instance for the following chain: (R1, {P1, P2, P3}) → (R2, {P1, P3}) → (R3, {P1}) consisting of three reports R1, R2 and R3 and labels {P1, P2, P3}, {P1, P3} and {P1} the chain target is package P1 and its hit number is three. The objective of this experiment is to show that the longer the chain is the higher the target hit number will be. Note that opposed to many previous studies (for example [23]) that aimed at finding the target package which is a more difficult problem to solve



Weights assigned to terms using the Tf-idf method:

```
({'unifi': 0.29447407686889265, 'reflectutil': 0.33919324980824395, 'classutil':
0.33539379704513483, 'class': 0.29411422057139275, 'current': 0.07138622368723736,
'histor': 0.16959662490412197, 'grown': 0.16959662490412197, 'differ': 0.0953196864204431,
'respons': 0.08685632305052403,'load': 0.09871841273198587, 'one': 0.16448033360810138,
'origin': 0.12487745196477068, 'facelet': 0.07820264606359888, 'codebas':
0.16959662490412197, 'homegrown': 0.16959662490412197,  'advantag':
0.16959662490412197, 'disadvantag': 0.16959662490412197, 'probabl':
0.14723703843444633, 'share': 0.09224516449127132, 'long': 0.12487745196477068, 'run':
0.07820264606359888,'post': 0.11767927289011874},
```

Label: {'org.apache.myfaces.view.facelets.compiler',
'org.apache.myfaces.view.facelets.tag.jsf.core',
'org.apache.myfaces.view.facelets.util', 'org.apache.myfaces.shared.util',
'org.apache.myfaces.view.facelets.el'}
)

Figure 8. A sample vectorized issue report and its label

Table 4. Pearson test results for random variables: (chainLen, hitNumber).

Product	Learning set size (number of reports)	Learned Threshold	Pearson coefficient value	Best Threshold	Best Pearson coefficient value
Cloudstack	26	0.35	1	0.35	1
Geode	507	0.1	0.98	0.1	0.98
Spark	1243	0.7	0.84	0.1	0.97
Camel	796	0.1	0.94	0.1	0.94
Geronimo	673	0.7	0.85	0.1	0.94
Hadoop	550	0.25	0.80	0.1	0.97
Myfaces	192	0.1	0.95	0.1	0.95
Hive	3070	0.1	0.99	0.1	0.99
Hbase	783	0.15	0.95	0.1	0.98
Cassandra	70	0.35	0.93	0.1	0.99

even with supervised machine learning methods, the intent of the ReportChainer is merely forming the chains correctly. The correlation value was computed using the Pearson method implemented in NumPy [33]. The results are listed in Table 4. The samples of (chainLen, hitNumber) pairs are plotted in Figure 9.

According to the observed results, for almost all of the products, the threshold value 0.1 resulted in chains with the highest correlation value. It was observed that for fairly long sequences of reports this threshold value worked fine across Apache products. There were products for which the learned threshold value differed from the best value (for instance CloudStack and Cassandra) due to the relatively few number of samples in the learning sequence: 26 and 70 respectively. Generally it can be argued that the shorter the sequence of reports, the higher threshold value will result in the best set of chains (in terms of the Pearson correlation metric). On the other hand, a lower threshold value creates

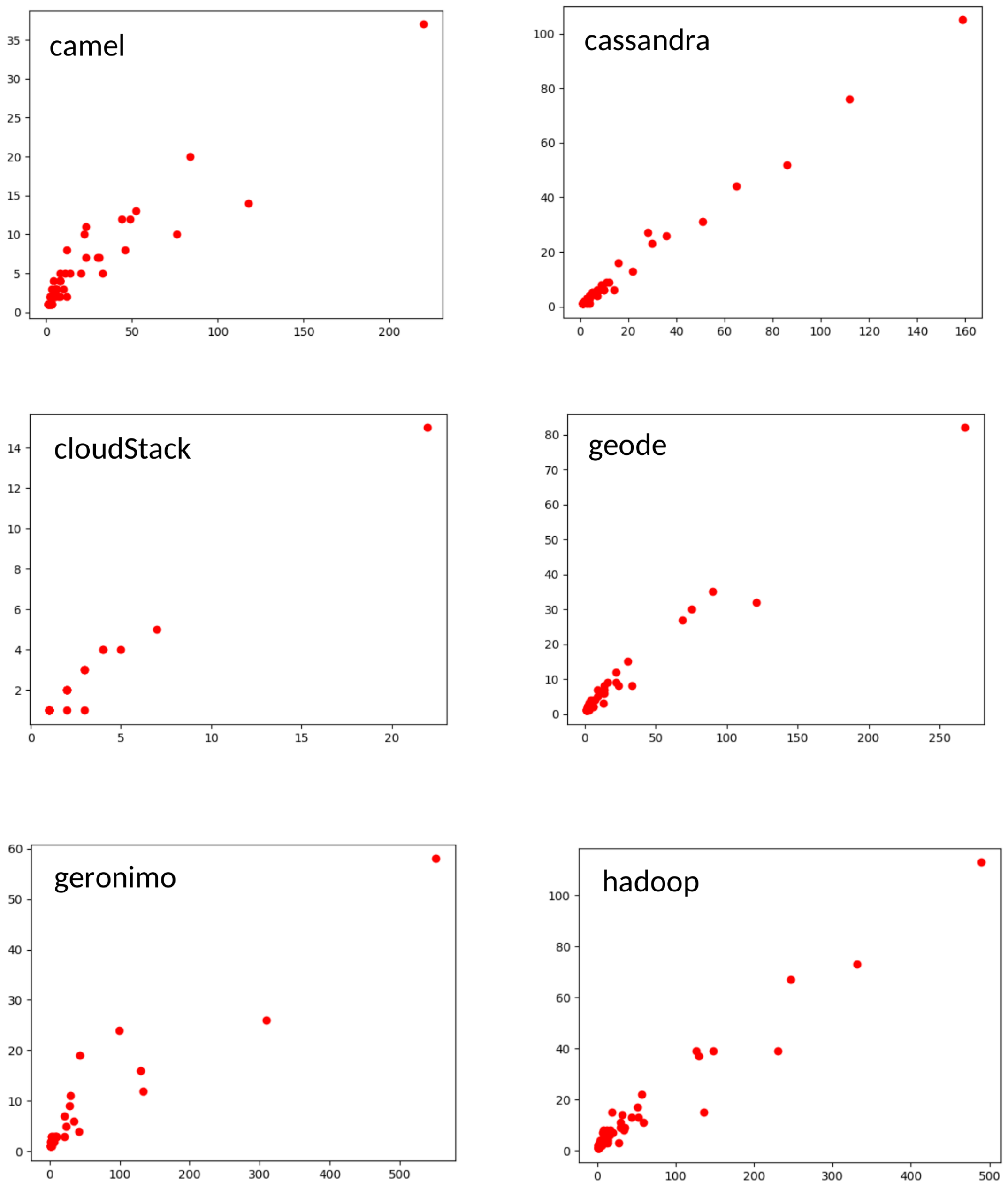


Figure 9. Plot of pairs: (chainLen, hitNumber) for six Apache products: Camel, Cassandra, CloudStack, Geode, Geronimo and Hadoop. The x and y axes are chain length and target hit numbers, respectively. The chains were created with split threshold = 0.1

fewer equal (short) length of chains due to their less join rejection rate. Many equal short size chains prevent the recommender module from working properly. For instance in CloudStack the best threshold learned 0.35, however by examining the resulted chains, many short equal length chains were observed. By choosing the threshold value less (= 0.1) this problem was overcome.

6. Discussion

The results confirmed strong linear direct correlation (0.73 on average) between the FAD metric and the "Design+ErrorProne" deterioration metric. Moreover the FAD metric is correlated with the "CyclomaticComplexity" deterioration metric well. Hence the FAD metric could be applied as a good indicator of the code deterioration level in terms of the number of emerged bad-smells and the extent to which the polymorphism is replaced wrongly by the complex conditional logics in the design. The statistical analysis using the Student"s T distribution showed that the mean value of the correlations (between FAD and "Design+ErrorProne") falls within [0.32, 1.14] with confidence 95%. Moreover as shown in Figure 7, the correlation value is almost independent of the number of received issue-reports on the product and at any time, as the product changes, the FAD metric can be used to measure the relative deterioration level.

To justify the relatively lower correlation values between FAD and "NpathComplexity" metrics, it should be noted that though both "NpathComplexity" and "CyclomaticComplexity" metrics measure the amount of complex conditional logics in the code, the former is very sensitive to the composition of the conditional statements in the code while the latter only counts the number of decision points.

Furthermore to explain why the FAD metric is superior to the BAD metric in terms of the correlation strength to the deterioration code-level metrics, we argue that:

(1) Usually adding a feature involves more (re-)design activities than fixing a bug and it is more likely to incorporate bad- structures into the code compared to fixing bugs.

(2) A good design suppresses the subsequent "New Feature" requests whereas a bad design produces forthcoming related "New Feature" requests: It is easy to see (as we have seen in our industrial experiences) that if a "New Feature" request is designed properly using well-defined design patterns (mostly based on polymorphism), the resulted code will involve the abstract concepts rather than concrete classes and hence the subsequent requests are likely to be variants and special cases of the initially requested feature and could be easily addressed by adding new classes rather than modifying the existing code (Open-Close principle [34]). Hence in this

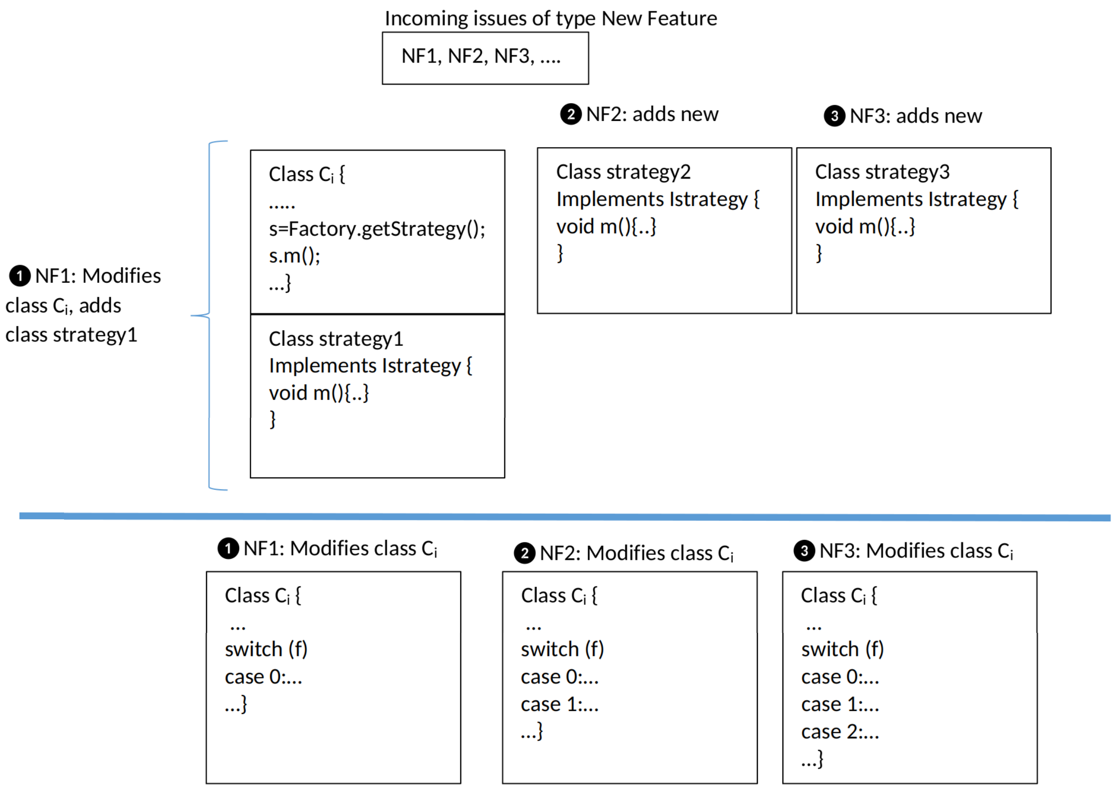


Figure 10. Classes with more anti-pattern constructs absorb more subsequent New Features. Three related New Feature requests: NF1,NF2 and NF3. Using strategy design pattern (top), using conditional constructs (bottom)

case less forthcoming requests of type "New Feature" is expected to be received. In contrast, the more polymorphism and design pattern constructs are replaced with conditional logics (and anti-patterns) in the code, the higher number of subsequent New Feature requests will be generated and absorbed by the module. This concept is illustrated in Figure 10.

To our best knowledge, there has been no study on the impact of different change types on the deterioration level of the code so far. According to the literature [34, 35]. This is widely accepted in the software engineering community that successive changes in the maintenance phase of software gradually causes the software to rot and the symptoms of deterioration to emerge: rigidity, fragility, needless complexity, opacity, immobility and these are due to bad smells in the code. Hence the more changes to the code the more deteriorated the code becomes. The results obtained in our study also corroborates the literature in the sense that "New Features" are sub types of "Change" and hence we could expect to see a good correlation between the number of "New Features" and the deterioration level. Now we can answer to our mentioned research questions:

RQ1: Can the QC team evaluate the quality level of the code, which is under successive changes during the maintenance phase, only by observing issue-related metrics such as the number of reported issues on a software package? Is the type of reported issues important in this evaluation?

Answer: According to the analysis presented in Section 5, yes. Issue reports are clustered using the stream clustering algorithm and the target package of the longest chain is recommended as the package with a high change rate. Due to the strong correlation between the value of FAD metric and the number of bad-smells, this package is presumed to be a priority for refactoring.

RQ2: Is there an effective stream clustering method to categorize incoming sequence of issue reports such that a bloated category be truly interpreted as the concentration of frequent changes on a specific software package and hence be reported as possible point of deterioration?

Answer: According to the correlation analysis results presented in Table 4, the proposed stream clustering algorithm could successfully create sequences of related issue reports such that the issue reports in each sequence are mostly focused on a specific package called target.

7. Threats to validity

As the most important threat to validity of the proposed QC approach to mention is the accuracy of the presented stream clustering algorithm, the parameters of this algorithm has to be fine-tuned according to the issue-report peculiarities and the product characteristics. Another threat to validity is the accuracy of the PMD bad smell detection tool. In [36] a comparative study of bad smell detection tools has been presented. In this study a detailed comparative study of four tools including PMD showed that this tool was able to reach up to 100% precision and 50% recall on particular test-cases and could outperform other tools in precision while rival in the recall value. Another limitation is that the study is delimited in the Object Oriented programming field though the strength of the proposed approach is independent of the used programming language.

8. Conclusions and future works

The continuous modifications of software modules lead to emergence of bad-smells and it is desirable to equip ITSs with assisting tools to notify the QC team about parts of the code that need immediate refactoring attentions. In this paper by creating a dataset of issue reports and their corresponding change information extracted from Apache and Eclipse open-source software repositories, a thorough study was conducted. The results confirmed a significant linear direct correlation between the FAD issue-level metric measured on a software package and the "Design+ErrorProne" and "CyclomaticComplexity" code-level metrics. Hence the packages with higher measured values for FAD can be considered as good candidates of parts highly exposed to deterioration, they need immediate attention of the QC team. A challenging part of the proposed model was to design a stream clustering to be able to split the sequence of reports into chains in a way that the length of the chains be a valid indicator of its target hit number. According to the observed results, for most of the products, the threshold value 0.1 resulted in chains with the highest Pearson correlation between chain lengths and their target hit numbers. As the future work we aim at studying other bad-smell types as well as other latent bug level metrics to predict the deterioration trend of the software during the maintenance phase. Another future work could be studying the same dataset with other smell detection tools apart from PMD.

References

[1] *Bugzilla.* [Online]. http://www.bugzilla.org (Accessed on 2018-06-06).

[2] *Atlassian.* [Online]. https://www.atlassian.com/software/jira (Accessed on 2018-06-06).

[3] L. Yu, S. Ramaswamy, and A. Nair, "Using bug reports as a software quality measure," 2013.

[4] M. Badri, N. Drouin, and F. Touré, "On understanding software quality evolution from a defect perspective: A case study on an open source software system," in *International Conference on Computer Systems and Industrial Informatics.* IEEE, 2012, pp. 1–6.

[5] C. Chen, S. Lin, M. Shoga, Q. Wang, and B. Boehm, "How do defects hurt qualities? An empirical study on characterizing a software maintainability ontology in open source software," in *International Conference on Software Quality, Reliability and Security (QRS)*, 2018, pp. 226–237.

[6] *Standard for Software Maintenance*, IEEE Std. 1219-1998, 1998.

[7] L. Yu, S. Schach, and K. Chen, "Measuring the maintainability of open-source software," in *International Symposium on Empirical Software Engineering*, 2005, p. 7.

[8] R. Malhotra and A. Chug, "Software maintainability: Systematic literature review and current trends," *International Journal of Software Engineering and Knowledge Engineering*, Vol. 26, No. 8, 2016, pp. 1221–1253.

[9] H. Sharma and A. Chug, "Dynamic metrics are superior than static metrics in maintainability prediction: An empirical case study," in *4th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO) (Trends and Future Directions).* IEEE, 2015, pp. 1–6.

[10] S. Shafi, S.M. Hassan, A. Arshaq, M.J. Khan, and S. Shamail, "Software quality prediction techniques: A comparative analysis," in *4th International Conference on Emerging Technologies.* IEEE, 2008, pp. 242–246.

[11] P. Piotrowski and L. Madeyski, "Software defect prediction using bad code smells: A systematic literature review," *Data-Centric Business and Applications*, 2020, pp. 77–99.

[12] M. Fowler, *Refactoring: improving the design of existing code.* Addison-Wesley Professional, 2018.

[13] *Apache.* [Online]. https://projects.apache.org/projects.html (Accessed on 2018-06-06).

[14] *Eclipse.* [Online]. https://www.eclipse.org/ (Accessed on 2018-06-06).

[15] V. Lenarduzzi, A.C. Stan, D. Taibi, D. Tosi, and G. Venters, "A dynamical quality model to continuously monitor software maintenance," in *The European Conference on Information Systems Management.* Academic Conferences International Limited, 2017, pp. 168–178.

[16] S. Kim, T. Zimmermann, K. Pan, and E.J. Whitehead, Jr., "Automatic identification of bug-introducing changes," in *21st IEEE/ACM International Conference on Automated Software Engineering (ASE '06)*, 2006, pp. 81–90.

[17] H. Wang, M. Kessentini, W. Grosky, and H. Meddeb, "On the use of time series and search based software engineering for refactoring recommendation," in *Proceedings of the 7th International Conference on Management of computational and collective intElligence in Digital EcoSystems*, 2015, pp. 35–42.

[18] F. Palomba, M. Zanoni, F.A. Fontana, A. De Lucia, and R. Oliveto, "Smells like teen spirit: Improving bug prediction performance using the intensity of code smells," in *International Conference on Software Maintenance and Evolution (ICSME).* IEEE, 2016, pp. 244–255.

[19] F. Khomh, M. Di Penta, Y.G. Guéhéneuc, and G. Antoniol, "An exploratory study of the impact of antipatterns on class change-and fault-proneness," *Empirical Software Engineering*, Vol. 17, No. 3, 2012, pp. 243–275.

[20] A.S. Cairo, G. de F. Carneiro, and M.P. Monteiro, "The impact of code smells on software bugs: A systematic literature review," *Information*, Vol. 9, No. 11, 2018, p. 273.

[21] D.M. Le, D. Link, A. Shahbazian, and N. Medvidovic, "An empirical study of architectural decay in open-source software," in *International con-*

ference on software architecture (ICSA). IEEE, 2018, pp. 176–17 609.

[22] *PMD static code analyzer.* [Online]. https://pmd.github.io/latest/pmd_rules_java.html (Accessed on 2018-06-06).

[23] D. Kim, Y. Tao, S. Kim, and A. Zeller, "Where should we fix this bug? A two-phase recommendation model," *IEEE transactions on software Engineering*, Vol. 39, No. 11, 2013, pp. 1597–1610.

[24] N. Limsettho, H. Hata, A. Monden, and K. Matsumoto, "Unsupervised bug report categorization using clustering and labeling algorithm," *International Journal of Software Engineering and Knowledge Engineering*, Vol. 26, No. 7, 2016, pp. 1027–1053.

[25] F.A. Fontana and M. Zanoni, "Code smell severity classification using machine learning techniques," *Knowledge-Based Systems*, Vol. 128, 2017, pp. 43–58.

[26] S.A. Vidal, C. Marcos, and J.A. Díaz-Pace, "An approach to prioritize code smells for refactoring," *Automated Software Engineering*, Vol. 23, No. 3, 2016, pp. 501–532.

[27] T.J. McCabe, "A complexity measure," *IEEE Transactions on software Engineering*, No. 4, 1976, pp. 308–320.

[28] C.D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval.* Cambridge University Press, 2008.

[29] *GitHub.* [Online]. https://github.com/ (Accessed on 2018-06-06).

[30] *isomorphic-git.* [Online]. https://isomorphic-git.org/ (Accessed on 2018-06-06).

[31] *Apache's JIRA issue tracker!* [Online]. https://issues.apache.org/jira/secure/Dashboard.jspa (Accessed on 2018-06-06).

[32] *bugs.eclipse.org.* [Online]. https://bugs.eclipse.org/bugs/ (Accessed on 2018-06-06).

[33] *NumPy.* [Online]. https://numpy.org/

[34] R.C. Martin, *Clean code: A handbook of agile software craftsmanship.* Pearson Education, 2009.

[35] R.C. Martin, M. Martin, and M. Martin, *Agile principles, patterns, and practices in C#.* Prentice Hall, 2007.

[36] E. Fernandes, J. Oliveira, G. Vale, T. Paiva, and E. Figueiredo, "A review-based comparative study of bad smell detection tools," in *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering*, EASE '16. ACM, 2016.

e-Informatica Software Engineering Journal, Volume 15, Issue 1, 2021, pages: 133–162, DOI 10.37190/e-Inf210107

A Systematic Reuse Process for Automated Acceptance Tests: Construction and Elementary Evaluation

Mohsin Irshad*, Kai Petersen**

*Ericsson Sweden AB, Karlskrona, Sweden

**Blekinge Institute of Technology, Karlskrona, Sweden and University of Applied Sciences Flensburg, Germany

mohsin.irshad@bth.se, Kai.Petersen@bth.se

Abstract

Context: Automated acceptance testing validates a product's functionality from the customer's perspective. Text-based automated acceptance tests (AATs) have gained popularity because they link requirements and testing.

Objective: To propose and evaluate a cost-effective systematic reuse process for automated acceptance tests.

Method: A systematic approach, method engineering, is used to construct a systematic reuse process for automated acceptance tests. The techniques to support searching, assessing, adapting the reusable tests are proposed and evaluated. The constructed process is evaluated using (i) qualitative feedback from software practitioners and (ii) a demonstration of the process in an industry setting. The process was evaluated for three constraints: performance expectancy, effort expectancy, and facilitating conditions.

Results: The process consists of eleven activities that support development for reuse, development with reuse, and assessment of the costs and benefits of reuse. During the evaluation, practitioners found the process a useful method to support reuse. In the industrial demonstration, it was noted that the activities in the solution helped in developing an automated acceptance test with reuse faster than creating a test from scratch i.e., searching, assessment and adaptation parts.

Conclusion: The process is found to be useful and relevant to the industry during the preliminary investigation.

Keywords: Software components and reuse, software testing, analysis and verification, agile software development methodologies and practices, software quality

1. Introduction

Software testing provides information on the quality of the software product [1]. An essential aspect of software testing is to verify that newly developed features of a product work according to the agreed requirements, and existing functionality is still working correctly (i.e., regression testing) [2]. With the advent of new technologies and processes, software testing's importance has increased as well. Existing testing approaches, such as unit testing and test-driven development, verify that software is working and developed according to the requirements. However, these approaches do not take into account the testing of business requirements. Acceptance testing verifies the business-critical aspects of software product [3]. The business requirements of a product and the user needs are the focus of acceptance testing [4]. The objective is to validate these business requirements before accepting the software product [3]. These business requirements, in the form of acceptance tests, are automated so that these are tested repeatedly and frequently similar

Submitted: 01 February 2021; Revised: 07 August 2021; Accepted: 09 August 2021; Available online: 25 October 2021

to unit tests [5]. Such automated tests are called automated acceptance tests (AATs) [5]. Studies in AAT have identified that AATs are responsible for high development and maintenance costs when the product is still changing frequently [5].

In agile development methodologies (such as Extreme Programming), the acceptance tests are scripted (using any programming language source code) by the development teams to get continuous feedback on the product's quality, and completeness [6]. Such scripted testing is referred to as "automated acceptance testing", and the tests are known as "automated acceptance tests" (abbreviated as AAT in this study). In agile development methodologies development of AATs starts much early and these AATs are executed frequently [6]. Empirical studies on AATs have established that AATs facilitate the organization's knowledge transfer by describing business requirements as test cases [7].

Software reuse improves productivity and quality during software development [8]. Studies have suggested that software reuse is most beneficial when organizations have a "systematic reuse process" [9]. In a systematic reuse process, the reusable artifacts are developed to support "development for reuse" [10], and these reusable artifacts are easy to find and adapt for reuse purposes [9]. The new development utilizes these reusable artifacts, and this is called "development with reuse" in which the identification, evaluation, and adaptation effort and costs are considerably reduced [10, 11]. A reuse process starts with "reuse assessment", which is a two-step activity, i.e., identification and evaluation of software components for reuse [12]. The evaluation of components for reuse involves the evaluation of the technical aspects (i.e., how to reuse) and the economic aspects (cost of reuse vs. benefits of reuse) of the component [12].

Almeida et al. suggested that the systematic reuse process's decisions should be financially justified (i.e., costs vs. benefits of reuse) before the actual reuse takes place [13]. Studies in literature have suggested that in "development for reuse", the costs of developing the reusable artifacts are high because the artifact should have high quality and artifact reused easily [9]. However, after multiple reuse instances, the benefits become more than the costs [14]. To calculate the reuse costs and reuse benefits, several cost models and metrics exist in the literature. However, many of these metrics and models only capture the costs and benefits of artifacts consisting of source code [15].

In scientific literature, solutions/processes that are constructed to solve a specific business problem can be developed using various existing practices [16]. An integrated solution (or a process) combines multiple activities that were not previously related to each other to address a critical business problem, e.g., finding a defect, fixing the defective code, and verifying the new code change [17]. Method engineering is a discipline of engineering that helps construct new solutions/processes from existing methods [18]. In this study, we have utilized method engineering to support the systematic reuse process of AATs.

This idea of a systematic reuse process can be applied to AATs to address the high development and maintenance costs of AATs. Reusable AATs can be developed to support the development of several new AATs, thus reducing the time to develop, increasing maintainability (by decreasing redundancy) and increasing quality of the automated acceptance tests [19]. Reusability (what to reuse, how to reuse, how to calculate reuse costs, and when to reuse) of AATs is a new area, and very few studies have addressed the reuse in AATs. To the best of our knowledge, there is a lack of suitable methods that support the reusability of AATs. The contribution of this investigation are:

– to provide a systematic reuse process that supports economically justified "development with reuse" and "development for reuse" of text-based AATs.
– to evaluate the proposed systematic reuse process for AATs with the help of experienced practitioners.
– to evaluate the proposed process using an industrial scale example.

Section 2 describes the background and related work, Section 3 contains research approach and the results are described in Section 4. Section 5 contains a discussion on the results of this investigation and Section 6 describes threats to the validity of this study. Important conclusions are described in Section 7.

2. Background and related work

This section describes the background and related work on AATs.

2.1. Automated acceptance tests (AATs)

An acceptance test is used to verify that the requirements are developed according to the contract/agreed specification [20][21]. The acceptance tests are derived from the customer requirements, and their focus is to verify the completeness of the implementation of requirements [22]. It was proposed that customers specify these acceptance tests, and these tests can be used as a form of requirements [7]. In a literature review [5], Haugset and Hanssen found the following benefits associated with AATs:

- AATs improve the understanding of the domain and product.
- AATs improve the communication with the customer by involving the customer in the test documentation process.
- AATs improve the system's quality by making it safe to make a code change in the product.

Several AAT formats (to write AATs) exist in literature such as source code, text-based domain-specific languages (DSL) and FIT tables [19, 23, 24]. From the existing literature, we have identified four types of AAT formats used by practitioners and researchers. These four formats are described below:

GUI-based formats. GUI-based acceptance tests are written to verify the functionality of GUI-based applications such as websites and mobile applications [25]. These tests emulate the user's actions (e.g., clicks, inputs) and validate the GUI-based application's output. These GUI actions are based on the scenarios described by the software requirements [25].

Code-based format. In the code-based AATs, programming language (e.g., Java, Python) is utilized to develop test execution logic. The test code implements and executes the scenarios in a programming language. The code-based AAT is triggered by a test framework such as JUnit[1]. These types of acceptance tests can be written using the existing practices and tools used for product development [26]. However, a drawback with this format is the difficulty for a customer to be part of the development of the acceptance test process [7].

Data-driven formats. In the data-driven formats, the acceptance tests are written so that input and output values are written in tabular form [27]. These input and corresponding output values are used with the same test execution logic. FitNesse[2] is the most commonly used framework for data-driven acceptance testing. A literature review on AATs identified that most of the studies present in literature are on the FitNesse (21 out of 26 identified studies) [7]. The data-driven formats are also useful when performing regression testing on systems with varying input and output values [28].

Text-based formats. Software practitioners prefer to write requirements in natural language text [29] and the text-based AAT frameworks (such as Cucumber[3], Robot Framework[4]) support the use of these requirements as the test-cases for software products, i.e., acceptance tests. Behavior-driven format, Keyword-driven format, Scenario by example, are popular text-based formats. The text-based AATs are gaining popularity because they help to promote a common understanding of requirements in the organization and increase collaboration among practitioners [30]. An experiment on comparing text-based AATs and FIT tables revealed that text-based AATs are easier to understand. These tests are developed in shorter duration as compared to FIT tables [31].

In behavior-driven format, the tests are written in template of *Given, When and Then* parts. Each behavior-driven test case validates a busi-

[1]https://junit.org
[2]http://fitnesse.org/
[3]https://cucumber.io/
[4]https://robotframework.org

ness requirement described by the customer [32]. Keyword-driven acceptance tests describe an executable requirement written using domain-specific keywords [33]. In specification by example, example scenarios are described in natural language by the stakeholders. These example scenarios are used to execute automated acceptance tests [34]. Each of these formats requires different tools and development frameworks. These formats allow the developers to write the tests in various styles and do not enforce any control over the vocabulary used in the AATs. Each statement of AAT (e.g., "the mobile is sent to customer" in the Scenario 1 below) is connected with a method in code called fixture/glue-code/hook that perform the actions associated with each statement. This method is written in any programming language (Python, Java, etc.) or a library publishes it (e.g., Selenium[5]).

Existing studies on AATs have discussed code-based and FIT tables (as an AAT format); little work is done on the text-based automated acceptance tests [7]. An example of a text-based automated acceptance test (a customer buying a mobile phone) is shown below using *behavior-driven development* format:

Scenario 1: A customer buys a mobile phone.

Given the customer accesses the web shop

And he enters his name and id-card number

When he selects a mobile phone

And customer pays the money using debit card

Then the mobile is sent to customers address

Another automated acceptance test in *keyword-driven* format is shown below verifying a use case of a customer buying a mobile phone.

Scenario 2: A customer buys a mobile phone.

Open Browser To Page example.com/shop
Input name Sam
Input id 939393
Submit information
Click button Buy to purchase mobile
Close Browser

Existing literature has described several approaches to generate automated test cases from manual acceptance tests automatically. Paiva et al. proposed a process where requirements are transformed into manual acceptance tests using a domain-specific language called Requirements Specification Language (RSL) [35]. Later, these RSL-based test cases are transformed into an automated test-scripts executable using Robot Framework. Later, their proposed process is evaluated using an example application. Soeken et al. provided a semi-automated approach to generate automated acceptance tests from natural language requirements [36]. The phrases from requirements are used to extract stubs executing the test scenarios. Later, the proposed approach was evaluated using an example case. Research studies have also utilized IDE-plugins and tools to generate automated acceptance tests from the use cases such as Fitclipse by Deng et al. [37] and Test-Duo by Hsieh et al. [38]. Such tools are often limited to a specific framework or acceptance test formats.

2.2. Systematic software reuse

Software reuse describes the practice of reusing existing software artifacts for developing a new product or maintaining an old product [39]. The field has been the subject of research for several years, and different aspects of software reuse such as costs, testing, artifacts, the level of reuse, stakeholders, and reuse processes have been investigated thoroughly by the researchers [40–44]. Software reuse can take place at any time during a project's life-cycle, starting from the requirements analysis phase to the maintenance phase [43, 45].

Organizations apply systematic software reuse to improve the quality, and productivity [41]. Various software development artifacts (requirements, test cases, code) can be systematically reused across the entire development process [46]. Lam et al. described a 10-step process to support the systematic reuse of software requirements [47]. The study suggested that a systematic reuse process can be successful when the organization produces and consumes reusable software artifacts. Research studies have identi-

[5]https://www.selenium.dev/

fied the following characteristics of a systematic software reuse process [12, 48]:

– *Development for reuse:* reusable artifacts are produced to support the organization's future software development needs.
– *Development with reuse:* new development occurs using reusable artifacts whenever possible.
– Guidelines (or techniques) to produce reusable artifacts exist and are practiced in the organization.
– Guidelines (or techniques) on reusing an artifact exist and are used in the organization.
– Development teams consider the extra costs and risks associated with the reuse of artifacts.

2.3. Related work: reuse in automated acceptance tests

The concept of text-based AATs is new, and their textual nature differentiates these tests from the conventional code-based test-cases, thus requiring a different approach for reuse.

In a study on reuse of AATs, Landhaußer and Genaid suggested an ontology-based approach (F-TRec) that enables reusing tests written in natural language. They found that approach lacked precision when retrieving test steps for reuse [49]. Crispin and House provided a tool to create AATs that can be reused across different modules [50]. They suggested that before reusing any AATs to develop a new AAT, practitioners should evaluate the effort spent writing and maintaining the new AATs. They also claimed that their proposed method could quickly create new test cases from the existing reusable tests, thus reducing development time and increasing quality. Rahman and Gao recommended an architecture that enables the reuse of Behavior-driven acceptance tests across multiple repositories [19]. They claim that their proposed approach can reduce maintenance costs, which are considered one of the pain points of AATs. Binamungu et al., in their study on the maintenance of BDD tests, found duplication as a critical challenge facing the practitioners [51]. Irshad et al. also found duplication among AATs in their study refactoring BBD specifications [52]. This duplication can be decreased with the help of an increase in reuse.

Liebel et al. identified costs related to writing the automated acceptance tests as one of the major problems of these acceptance tests [25]. In their industrial case study, Hanssen et al. suggested that the benefits of automated acceptance tests should be weighed against the costs associated with these tests [23]. They found that automated acceptance tests verifying the graphical user interface (GUI) are often hard to develop and maintain. GUI-tests are unstable and require more maintenance. Angmo and Sharma evaluated AAT tools and proposed that cost-benefits analysis should be done when considering a tool for automated acceptance tests [53]. Shelly and Frank conducted a literature review on story test-driven development. They found that cost of writing AATs is high, and many organizations do not have a budget to account for this high-cost [54].

Xie describes time as the measure of cost in development and maintenance of AATs [55]. Haugset and Stalhane claim that AAT can benefit organizations in two ways (i) increasing the correctness of requirements and (ii) automated test-ability of requirements. Borg and Kropp described a tool that helps maintain and refactor automated acceptance tests, reducing their maintenance costs [56]. To support reuse and reduce costs, Schwarz et al. introduced and evaluated a plugin (Eclipse[6] IDE plugin) that rapidly develops AATs. They claim that using this plugin can quickly develop automated acceptance tests quickly [57].

In short, we identified the following gaps in the literature, and our investigation attempts to provide a solution to these research gaps.

– **Lack of support to systematic reuse of AATs:** Only one study is identified that provides methods for "developing with reuse", and "developing for reuse". However, the study's approach is limited to the use of a specific tool provided by the authors of the study [50].
– **Lack of generic reuse practices supporting diverse AAT formats:** There is a need for a generic practice supporting the reuse of

[6]https://www.eclipse.org/

all types of AAT formats. Several formats to write text-based AATs exist in research and practice (BDD, keyword-drive, specification by example).

- **Lack of means to calculate benefits of reusing AATs:** AATs are costly to write and reuse, and costs should be considered when reusing AATs [7, 23]. An instrument to calculate the reuse costs of AATs may help perform cost vs. benefits analysis of reuse instances.

This study complements the existing work by providing a systematic reuse process that supports economically justifiable "development for reuse", and "development with reuse" of AATs, independent of any particular text-based AAT formats, independent of any particular tools and frameworks.

3. Research approach

This section describes the research questions, study execution, data collection, and data analysis performed during this investigation.

3.1. Research questions

In order to achieve the objectives of this study, we have devised the following research questions:

- **RQ 1:** How can the cost-effective systematic reuse of text-based AATs be achieved?
  This research question details the establishment of a systematic reuse process for AATs along with suitable activities and techniques used in the process. The proposed process incorporates the cost-benefit aspects of AATs when evaluating reuse opportunities.
- **RQ 2:** How does the systematic reuse process, from RQ 1, perform concerning performance expectancy, effort expectancy, and facilitating conditions in the industrial context?
  RQ 2 addresses the preliminary industrial evaluation of the systematic reuse process of AATs with the help of industry professionals and practical demonstration.

3.2. Study execution

The study is executed using method engineering that is a research framework to develop new tools and methods. Method engineering consists of individual method fragments that can combine to form a project-specific (or product-specific) customized method [58]. Method engineering is applied using the following phases (defined by Mayer [18]):

- *Document motivation:* the motivation to develop a new process is identified and documented.
- *Search for existing methods:* the existing methods are identified that may help in the new process.
- *Tailor existing methods:* the identified methods are adapted to suit the needs of the process.
- *Design method application technique:* a new process is formed using the modified existing methods.
- *Test candidate design elements:* the process (and its components) are evaluated to identify potential shortcomings and modifications.
- *Refine method design:* The process is modified/refined based on the evaluation.

The first four phases help in constructing a new process, i.e., see Figure 1. Later, the last two phases (see Figure 1) evaluates and improve the new process. The study approach and the research questions are described in Figure 1. The first step (construction of process) identifies the motivation, requirements and develops a new systematic reuse process for text-based AATs. The second step (Evaluation and refining of the process) evaluates the new process by assessing the performance expectancy, effort expectancy, and facilitating conditions of the process, as suggested by [61]. The evaluation consists of:

- qualitative feedback on the process from experienced software practitioners (also referred to as static validation in [62]),
- by demonstrating the usage of the proposed process in an industrial application by an author.

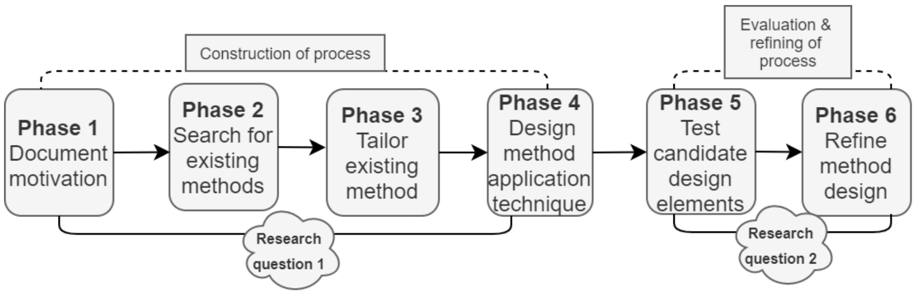

Construction of process
Evaluation & refining of process
Phase 1 Document motivation
Phase 2 Search for existing methods
Phase 3 Tailor existing method
Phase 4 Design method application technique
Phase 5 Test candidate design elements
Phase 6 Refine method design
Research question 1
Research question 2

Figure 1. Research approach inspired by method engineering [18]

The sections below provide the details of the construction of the process and its evaluation (i.e., research method, the data collection, and analysis). The final version of the method is released after incorporating the feedback from the evaluation.

3.2.1. Construction of process – using method engineering

This step is used to construct a process supporting "development with reuse", and "development for reuse" of AATs also considering the costs vs. benefits of AATs, i.e., a systematic reuse process. The details of the four phases of this step are described below.

Phase 1: Document motivation. The existing studies that report the problems related to the reuse of AATs were identified and examined. The authors read the literature reviews and mapping studies on AATs to identify supporting methods for reusing AATs. We have focused on text-based (non-code) AATs only as the reuse of other types of AATs (code, FIT Tables) is already discussed in existing [7]. One of the authors conducted the following step using Google Scholar[7].

– Find and examine the literature reviews/ mapping studies on AATs. Table 1 describes keywords and identified studies.
– Examine the reference/citations in identified literature reviews.
– Manually analyzes the identified studies to find issues linked with AATs.
– Manually analyze the identified studies to find details on the reuse of AATs.

The identified literature reviews/mapping studies on the AATs [5, 7, 54, 59, 60] do not explicitly discuss the reuse of AATs. However, we identified studies using citations and references to review studies. The analysis (documented Section 2) concludes that a supporting process is needed that should address the three requirements/needs: AAT format, AAT reuse process, and AAT reuse costs. These three requirements/ needs are described below.

AAT format: The process should be independent of the AAT format. Different AAT frameworks support other formats, e.g., in Cucumber, BDD format is used, and the Robot Framework supports keyword-driven and BDD formats. The reuse process of AATs should be independent of formats dictated by different AAT frameworks.

AAT reuse process: The reuse process of AAT should support "develop for reuse" and "develop with reuse". The process should contain activities to search for a reusable AAT, assessment of AAT for reuse, and adaption for reusing AATs.

Table 1. Keywords and identified studies

Keywords	Identified review studies
Automated Acceptance Testing review	[5], [7]
Acceptance Testing review	[5], [59]
Story-driven review	[54]
BDD Review	[60]

[7]https://scholar.google.com/

AAT reuse costs: The process should incorporate activities for assessment of reuse costs of an AAT. Previous research has shown the high cost of writing and reusing AATs [53, 63].

Phase 2: Search for existing methods. In this phase, the existing methods are identified to match the needs/requirements of an AATs reuse process identified in the previous phase, i.e., AAT format, AAT reuse process, and AAT reuse costs.

AAT format: The following AAT formats are identified Behavior-driven tests, story-based test format, specification by example format, and keyword-driven tests. These formats were identified using the existing literature reviews on AATs ([7, 23, 59, 64]).

AAT reuse process: Mili et al. [65] suggested three stages for a reuse process: (i) finding the reusable artifact, (ii) assessing the relevance of reusable artifact, and (iii) adaptation of the artifact for reuse. For the first and second stage (finding and assessing relevance), techniques suitable for text-based artifacts were identified that could help in these stages. The two suggested approaches are (i) normalized compression distance (NCD) [66], and text classification using machine learning [67]. For the third stage (an adaptation of AATs for reuse), no existing study provides guidelines on adapting AATs for reuse. Since software requirements and AATs are text-based artifacts; therefore, we propose that the reuse methods used in software requirements can support reuse of text-based AATs [68].

AAT reuse costs: Software reuse cost models are categorized into three categories (by [14]):

1. Return-on-investment models (ROI) that measure benefits after an organization has invested in developing reusable artifacts.
2. Cost-benefit models that are used for making better decisions on reuse investments.
3. Cost-avoidance models that help in calculating costs avoided through reuse of artifacts [14].

We believe that cost-avoidance models are best suited for reusability assessment since these do not assume that an upfront reuse related investment was made, as required by ROI and cost-benefit methods [14]. The methods provided in the review study on cost-avoidance through reuse were identified for capturing reuse costs of AATs [15].

Phase 3: Tailor existing method. The identified methods from the previous phase are tailored to address the needs of a systematic reuse process of AATs.

AAT format: For AAT formats, no tailoring is needed.

AAT reuse process: In the AAT reuse process, to find and assess the AAT candidates for reuse, two techniques (Normalized compression distance and text classification using machine learning) are tailored to be used for AATs. The AATs are text-based, and the AAT content is structured so that each line acts as a single independent, reusable test statement (see examples in Section 2). The two techniques were implemented using scripts (available at [69]) to apply them on the AAT suite. Section 4 described the working of these two techniques.

For the AAT adaption template (to support "develop for reuse"), four templates (Structuring, Matching, Analogy, Parametrization) from the identified review study [68] were selected for their applicability over AATs. The reuse of software requirements inspires these templates. Section 4 describes the details of these templates.

AAT reuse costs: The study on reuse cost avoidance methods described four approaches to calculate reuse cost-avoidance [15]. Only two of the reuse cost methods ([70] and [15]) are applicable over non-code artifacts such as text-based AATs. These two methods and their corresponding metrics do not require any tailoring for AATs.

Phase 4: Design method application technique: In this phase, the identified activities concerning the components of the process are assembled to form a process.

The three identified requirements/needs (AAT format, reuse process, and reuse costs) and their identified methods are converted into activities of a process. As discussed in Section 2.3, AATs can be costly to write and maintain; therefore, a check is introduced in the process that lets practitioners decide when to write a new AAT and when to reuse an existing AAT. Furthermore, it was decided among the authors to divide the activities into two levels, i.e., organizational level activities and test developer-level activities. These activities are described below:

Organizational/Team level activities and decisions: The organization-level activities are decided when setting up the reuse related process, and these are rarely changed. Some examples of these activities are the format used in the organization (e.g., BDD, keywords files), the support for the reuse approach, and the test framework used in the organization. The actor in these activities is the test architect.

Test developer-level activities and decisions: The activities performed during this level are the primary activities relevant to developing a new AAT. An example of such activities is selecting an ATT for reusing purposes. Test developers perform these activities. The activities are guided by the choices made during the organizational activities, e.g., which artifact to reuse, how to modify an artifact for reuse.

The constructed systematic reuse process is documented in Section 4.1.

3.2.2. Evaluation and refining of process

After developing a reuse process, the next step involves the evaluation and refinement of the process. The sections below describe the types of evaluation and refinement steps.

Phase 5: Test candidate design elements. After designing the process and its activities, the next step evaluates the process for its usefulness concerning the reuse of AATs. [62] has suggested that validation with practitioners helps assess the industrial usage of the process before applying it in the industry.

Petersen and Wohlin described six context facets for explaining the context of the evaluation [71]. The evaluation in this study is conducted in the context of large-scale software development. The complexity concerning the reuse of AATs is considered high for large-scale organizations with a considerably large test base. The development process is assumed to be an agile development method where new requirements, development, and testing are conducted in each iteration.

This survey questionnaire's participants belong to two large telecommunication organizations, and the industrial demonstration is conducted in one organization's product verification unit. The details of the evaluation are described below.

Qualitative feedback of software practitioners. In this evaluation, software practitioners provided feedback on the proposed systematic reuse process.

Objective: The objective of this evaluation was to identify generic findings on the applicability of the process and sanity-check the process for performance expectancy, effort expectancy and facilitating conditions as suggested in unified theory of acceptance and use of technology (UTAUT) by [61].

Wohlin et al. describe "exploratory surveys" as a way to validate the proposals before a thorough investigation is conducted [72]. As the first step, the proposed systematic reuse process in this study was evaluated using an exploratory survey to improve the process before implementing and evaluating it in the industry, which requires a longitudinal study. This longitudinal study is planned as the next step after this study because it requires resources, budget, and changes in the organization's current ways of working. Wohlin et al. suggested questionnaires and interviews as two data collection methods during the surveys. We developed a questionnaire and asked the subjects to fill the questionnaire during an online session (for direct interaction like an interview).

Questionnaire design: In this evaluation, a questionnaire (available at [73]) is developed that contained four parts: (a) description of reuse in automated acceptance testing, (b) description of our proposed process, (c) an example of explaining the usage of our proposed process, and (d) practitioners feedback on the proposed process. The questionnaire's design, as per Molléri et al. can be classified as "self-administrated", i.e., online form [74]. This questionnaire is used to execute the industrial evaluation. The details on designing the questionnaire, selecting participants, data collection, and data analysis are provided below.

The following questions (under each UTAUT construct) were part of the questionnaire.

Performance expectancy. This construct describes "the degree to which technology will benefit users" [61].

– Question 1: In your opinion, what are the benefits of using the proposed process?
– Question 2: What are the drawbacks/limitations of the process? How can we improve/revise the process?

Effort expectancy. This construct describes "the degree to which technology is easy to use" [61].

– Question 3: In your opinion, how easy is it to use this process?
– Question 4: Do you have any recommendations to improve the ease of use?

Facilitating Condition. This construct describes "the degree to which technology helps in performing a task" [61].

– Question 5: Are there any other steps that should be added to the process?
– Question 6: Are their steps that should be removed from the process? If yes, then kindly list those items here and also state why do you recommend removing them?

Subjects: Initially, two academic researchers (not authors) working on AATs were invited to review the questionnaire and process for sanity-check. They suggested improvement in the questionnaire (i.e., text, process flow). After incorporating the input of the researchers, we conducted an industrial evaluation.

During the industrial evaluation, five industry participants with knowledge of AATs in large--scale products and considerable working experience are considered. As the reuse of AATs is a relatively new area, it is difficult to find participants with relevant experience. Five participants evaluated the process and provided their feedback on the process. These participants were selected from two large-scale organizations with more than five years of experience with development. The background of the participants is provided in Table 2.

Study execution: The authors have presented the study's purpose, the working of the process, and its activities to the participants in an online meeting. The participating subjects asked questions about things they do not understand during the presentation of the process. The details were provided to the inquiring participants. This step was conducted to overcome the survey questionnaires' critique that subjects might not understand the concepts and processes by reading the questionnaire.

In the next step, the participants were provided the link to the survey questionnaire, and they were asked to fill in the information. The author remained online while the respondents filled out the form to have direct interaction and allow participants to ask follow-up questions. These sessions lasted between 45–65 minutes, as shown in Table 2. These online sessions helped improve the qualitative feedback and relevance of the feedback for the reuse of AATs.

All five participants responded to the survey questionnaire in the presence of an author. In the form of responses to the questionnaire, the feedback is evaluated using the "Constant comparison" method [75]. This method can help in the identification of common themes present in the qualitative data. During the analysis, the responses are divided into themes related to the process's benefits, usefulness, and completeness. Later, the final process is improved based on identified themes corresponding to performance expectancy, effort expectancy, and facilitating conditions.

Demonstrating industrial application of the proposed process – an example. Stous-

Table 2. Background of the software practitioners (P1–P5) participating in the evaluation

	Work experience	Experience in AAT	Role	Product type	Online session
P1	12 years	5+ years	Developer	Large-scale	60 minutes
P2	17 years	5+ years	Architect	Large-scale	45 minutes
P3	12 years	4 years	Test developer	Large-scale	55 minutes
P4	5 years	3 years	Test developer	Large-scale	65 minutes
P5	13 years	2 years	Developer	Large-scale	50 minutes

trup identified that lack of practicality or an excessive level of complexity results in the failure of seemingly successful research projects when these projects are applied in the industry [76]. A demonstration is conducted in an industrial setting using the proposed systematic reuse process to address this concern.

Objective: The purpose of this evaluation is to check the feasibility of implementing the process in the context of a real software development environment using the existing tools and knowledge present in the organization (e.g., metrics to identify reuse costs). This evaluation will help identify lessons regarding each activity before the process is applied and evaluated in the industry without authors' involvement.

The author (working in the organization) evaluated details related to:

– the effort to create an AAT using the proposed process,
– the number of tasks performed in each activity of the proposed process,
– details of tasks performed in each activity of the proposed process.

This demonstration took place in one of the system verification units of an organization that develops a large-scale product consisting of 28 micro-services. The unit of analysis is the AAT suite used by the end-to-end verification team. Four experienced test developers manage the AAT test suite. The AAT suite was introduced in the year 2016 and contained 87 system-level AATs. Each system level AAT is based on a complete use-case. The AATs are written in keyword-driven format (text-based), with fixtures written in Java. The AAT suite is extended when the test developers find a stable product no longer modified by the development teams. The AATs are executed each night using Jenkins to build, execute the AAT, and generate AAT reports.

Table 3. Activities in a systematic reuse process for automated acceptance tests (AATs)

Activity	Input	Output	Actor	Type
A1: Select keywords	Requirements in natural language	Keywords extracted from requirements to find reusable AAT	Test developer	Manual
A2: Select AAT format	AAT formats (an organization can seek help from literature if it does not have a pre-decided format)	Selected AAT type, e.g., BDD	Test Architect	Manual
A3: Select artifact adaptation template	List of adaptation approaches to support "develop for reuse". See 4.1.3	Selected approach to adapt AAT for reuse and to help in search	Test Architect	Manual
A4: Search for reusable AATs	Keywords from A1 and a search technique. Details in 4.1.4	The list of AATs matching keywords	Test developer	Automated
A5: Assess relevance of AATs	The AATs matching keywords from output of A4	The potentially reusable AATs	Test developer	Manual
A6: Select reuse cost method and metrics	The available metrics, the reuse cost calculation method. Details in 4.1.6	A selected reuse cost method	Test developer	Manual
A7: Calculate cost of reuse	The value of metrics and the reuse cost method	Evaluation if reuse is beneficial or not	Test developer	Automated
A8: Develop new AAT by reuse	The selected reusable AAT and adaptation technique from activity A3	A new AAT supporting "development for reuse"	Test developer	Manual
A9: Develop a new AAT	Software requirements, adaptation technique from activity A3	A new AAT supporting "development for reuse"	Test developer	Manual
A10: Add new AAT to repository	A new (or reused) AAT test-case	A new AAT is added to the repository	Test developer	Manual

IntellJ[8] was used as an IDE for developing the AAT. The AATs are stored in a Git[9] repository. The system under test (SUT) is hosted on a remote server, and the development takes place on the local machine. The AAT is executed against SUT from the local machine, but after finalizing the AAT, the AAT execution is automated as part of the AAT suite that is executed frequently using a continuous integration server.

One of the authors (who is not a developer of the existing AATs) applied the process's activities to develop a new AAT with reuse. The new AAT is to verify that a REST interface's performance is within limits decided by the requirements engineers. The activities described in Table 3 are followed to evaluate the proposed process's flow.

According to the classification provided by Lethbridge et al. [77], third-degree data collection was utilized in this study (i.e., historical data on the case or using the compiled information). For the searching and assessment activities, a script (described in 4.1.4) is used to search in the repository and identify the relevant AATs. The search and assessment data is available at [78]. The cost model used in this demonstration needed historical information on the person-hours previously spent on similar AAT [15]. This information was extracted from the Git repository using the "git history <AAT-File-Name>" command. The time taken was noted for each activity by one of the authors.

The data analysis was performed on the collected quantitative and qualitative data. The quantitative data is the time taken during activity, the number of steps performed in each activity (ease of use), and qualitative data involves the test developer's observations (i.e., author). The results from the data analysis are described in Section 4.2.2.

Phase 6: Refine method design. In this phase, the proposed method is modified based on the feedback from the evaluation. The identified themes related to the method's improvement are considered, and the method is modified. The feedback from the evaluation and the changes suggested during the evaluation are described in Section 4.2. The final version of the proposed process is present in Section 4.3.

[8]https://www.jetbrains.com/idea/
[9]https://git-scm.com/

4. Results

This section describes the outcome of the two research questions and the final systematic reuse process. The first research question described a cost-effective systematic reuse process, its activities, and techniques applicable to the activities. The second research question describes the industrial evaluation of the systematic reuse process. Later, the final version of the systematic reuse process is described.

4.1. Constructed solution: A systematic reuse process for AATs

The systematic reuse process for AATs supports activities and techniques to (i) develop for reuse, (ii) develop with reuse, and (iii) methods and metrics to calculate the reuse costs of AATs. Figure 2 shows the constructed process to support systematic reuse of AATs and the details of each activity is described in Table 3.

In the first activity, the practitioner analyzes the requirements and identifies keywords relevant to the new AAT. Next, an AAT artifact format is selected, e.g., BDD. After selecting the artifact, an approach to facilitate "development for reuse" is selected. The next two activities search and assess the relevance of AAT artifacts for reuse. If no suitable artifact is found, then a new AAT is developed from scratch. If existing relevant AATs are found, then the next activity is to calculate the cost of reuse. If the reuse is presumed cheaper, a new AAT is developed by adapting the existing artifact according to the approach selected for development for reuse. In the last activity, a new reusable AAT is added to the repository. Table 3 provides the inputs, outputs and actor of each activity.

In the sections below, each activity in the process is described below with an example.

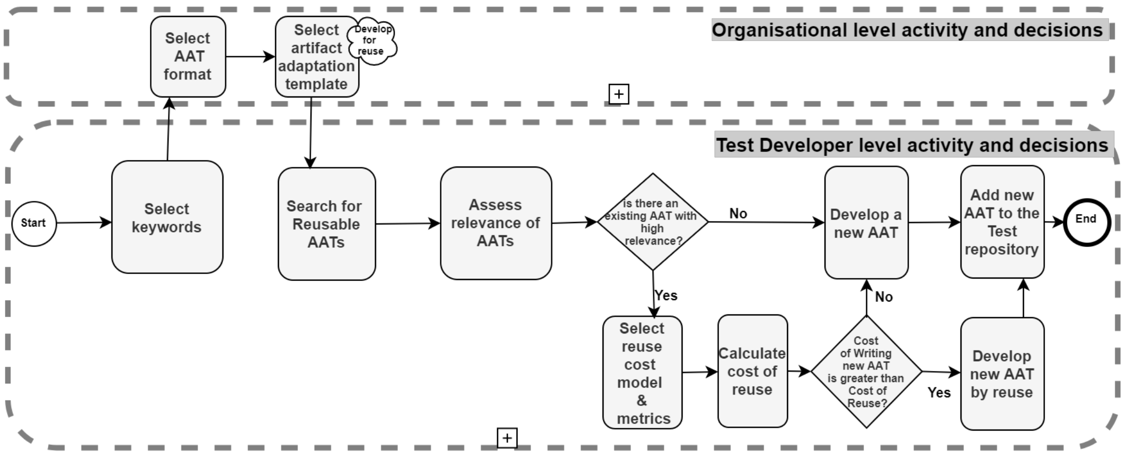

Organisational level activity and decisions
Select AAT format
Select artifact adaptation template
Develop for reuse
Test Developer level activity and decisions
Start
Select keywords
Search for Reusable AATs
Assess relevance of AATs
Is there an existing AAT with high relevance?
No
Yes
Develop a new AAT
Add new AAT to the Test repository
End
Select reuse cost model & metrics
Calculate cost of reuse
Cost of Writing new AAT is greater than Cost of Reuse?
No
Yes
Develop new AAT by reuse

Figure 2. A systematic reuse process for automated acceptance tests (AATs)

4.1.1. A1: Select keywords

Software requirements are often written in natural language [29], and in this activity, relevant keywords are selected that represent the requirements. These keywords will help in searching for reusable AATs in the existing test-base. This is a manual activity performed by a test developer. This activity may be performed in several iterations before finalizing the keywords; e.g., an initial selection of keywords may not provide good search results; therefore, it needs several revisions (or discussion with experts) before reaching the concluding choices.

Example: The test developer has the requirements: (i) "As a user I should be able to register my new account", and (ii) "As a user, I am able to view the products using my account". In this example, the relevant keywords selected from the requirements are "register account", "view products".

4.1.2. A2: Select AAT format

The text-based AATs are written in a variety of formats. Each of these formats has a unique way of writing AATs, e.g., stories, specifications, behaviors, features. Therefore, it is necessary to select the format in which the new AATs are written or reused, e.g., a selection from Behavior-driven tests, story-based tests, feature files, keyword-driven tests. This activity is performed manually.

Example: As an example, we can assume that the organization writes AAT in a Behavior-driven format; therefore, BDD is selected as the format of AAT.

4.1.3. A3: Select artifact adaptation template

In this activity, a reuse adaptation template is selected for writing a new AAT from existing artifacts. The objective of this activity is to support "development for reuse". The new AAT will be modified and saved according to the artifact adaptation template. The four templates supporting development for the reuse of AATs are structuring, matching, analogy, and parameterization as described in Table 4. This selection of artifact adaptation template is a manual activity. A detailed discussion on the reuse adaptation approaches for text-based software requirements is provided by Irshad et al. [68].

Example: In this case, "Structuring" (from Table 4) is selected as reuse adaptation template because BDD test cases are written in structured format of "Given, When, and Then" [32].

4.1.4. A4: Search for reusable AATs

A vital activity of the constructed process is to search for reusable AATs in the repository. This search is conducted using the keywords from activity A1. The search operation can be implemented using an automated script that searches for AATs matching the keywords.

Table 4. Templates supporting "development for reuse". Inspired by [68]

Approach	**Description**
Structuring	Reusable AATs are saved in specific/pre-defined format to reuse, e.g., directory structure.
Matching	Reusable AATs are saved in formats such AATs are retrieved using matching, e.g., supports search using lexical or semantic matching.
Analogy	Reusable AATs can be storied in languages/formats that support retrieval using analogy-based approaches, e.g., special languages supporting analogical matching of AATs.
Parameterization	AATs can support parameterization for reuse, e.g., the use of variables in keywords. Many AAT frameworks (Robot, Cucumber) support this adaptation approach.

We have identified two techniques that are useful for searching for text-based reusable AATs. These techniques are applicable over the text content of AATs only and do not consider the fixture/hooks/glue-code of AAT. We have provided an automated script for each of these techniques as part of the reuse process. These techniques are described below.

Normalized compression distance (NCD) With this technique, the similarity between an AAT and keywords is calculated using a compression algorithm [79]. In this study's context, NCD helps calculate the pair-wise distance between all the AATs present in the test base against the keywords identified. Later, this compression distance helps in the assessment of similar and dissimilar AATs. NCD can be defined by the following equation [79]:

$$\mathrm{NCD}(s1, s2) = \frac{Z(s1s2) - \min\{Z(s1), Z(s2)\}}{\max\{Z(s1), Z(s2)\}}, \tag{1}$$

Here, $s1$ is an automated acceptance test present in the test suite, and $s2$ is a keyword used to search and assess reusable automated acceptance tests. Z represents the compressor used for the calculation of NCD. $Z(s1)$ represents the compressed size of AAT $s1$, $Z(s2)$ represents the compressed size of AAT $s2$ and $Z(s1s2)$ represents the compressed size of the concatenation of $s1$ and $s2$. NCD values lie between 0 and 1, where 0 means that the BDD specifications are similar, while 1 represents that they are entirely different. A script implementing NCD is provided as part of the proposed process [69].

Text-classification using machine learning text-classification can help in identifying reusable AATs by training a classifier using a supervised machine-learning algorithm [67]. For text--classification commonly used machine learning-based algorithms are Naive Bayes and Support Vector Machines [80]. A machine learning-based algorithm improves the classification process because it considers the domain-specific language used in the AAT instead of using Wikipedia or large-text databases present over the internet [67]. The existing AAT suite is used to train the text classifier using each AATs title as a category. Later, this text classifier helps in suggesting the closest matching reusable AAT cases to the selected keywords. The following sequence of steps is followed when using machine-learning based text classification to search for and assess a reusable AAT.

Step 1: Place each AAT in a separate file where each file's name is unique. This is needed to allow ML-algorithm to assign a category to each AAT in the training set.

Step 2: Load the training AAT files into memory.

Step 3: Extract features from AAT files using Bag of Words.

Step 4: Train a classifier from these features.

Step 5: Use the search keywords to query the classifier to identify the reusable AAT.

It is important to note that Step 1 to Step 5 are executed only if new changes are introduced in the AAT suite. A script providing the implementation from Step 1 to Step 5 is provided as part of this proposed process (available at [69]).

Table 5. NCD values generated by the automated-script

Selected keyword	Scenario 1 NCD value	Scenario 2 NCD value
view products	0.620	0.290
register account	0.720	0.850

Example: For the sake of simplicity, we assume that there are only two AAT scenarios in the repository and a test-developer wants to use the keywords (from A1) to search for the closest matching scenario to the keywords. The test-developer uses the automated-script (See [69]) to perform this activity.

Scenario 1: *A user deletes a product in the system*
Given A product is configured in the system
When User sends a Delete request to delete the products
Then User is able to delete the product
Scenario 2: *A user view a product in the system*
Given A product is configured in the system
When User sends a Get request to fetch the products
Then User is able to view the product

The distance measures (such as NCD) represent a mathematical concept of similarity [81, 82]. The similarity is high when the distance between objects (in comparison) is low, i.e., a value closer to "0". An advantage of distance measure is that they can classify the similarity and dissimilarity of two objects on a numerical rating scale by suggesting how closely similar or how different objects are from each other, i.e., 0.90 means very dissimilar, 0.75 means dissimilar, 0.248 means similar, 0.05 means very similar. The text-classification approaches often classify in the binary format, i.e., two objects are alike or different.

The test developer selects "Normalized Compression Distance (NCD)" to retrieve the reusable AATs with help of keywords "register account", "view products". The search activity is performed using the script implementing NCD, and a pairwise comparison of each keyword and scenario is conducted. For each comparison, a value between "0" and "1" is produced. The values are shown in Table 5, e.g., Scenario 2 and keyword "view product" has lower NCD value (0.290), showing higher similarity because keywords "view", "products" are found in Scenario 2 but not in Scenario 1.

4.1.5. A5: Assess relevance of AATs

In this activity, an assessment is made on the relevance of the identified reusable AATs from the searching activity. This assessment involves how closely the identified AATs are related to the requirements. This activity requires domain knowledge and understanding of the existing AATs; therefore, it is a manual activity, e.g., if several closely matching reusable AATs are identified, a manual assessment to select the most suitable reusable AAT.

Example: The search results of activity A4 (in Table 5) contains results from the search operation. Table 5 shows that the NCD values between the keywords and two scenarios. As stated earlier, an NCD value closer to 0 means higher similarity, and a value closer to 1 means lower similarity. Scenario 2 and keyword "view product" has value 0.290, showing higher similarity and relevance as a candidate for reuse.

4.1.6. A6: Select reuse cost method and metrics

In this activity, a method is selected to calculate and compare costs of developing a new AAT by reusing an existing AAT and costs of creating a new AAT from scratch. The metrics required to apply the method are selected in this activity. This activity is necessary because the development of text-based AATs is known for higher costs, and in some cases developing a new AAT from scratch can provide more savings than reusing an AAT. The reuse cost calculation methods and metrics used in the code-based artifacts are not applicable over the text-based AATs because they use "lines of source code" (or indirect metrics such as complexity and function points) as an essential metric to calculate the cost. The selection of the cost model depends on the following factors:

1. Maturity of the existing reuse process in the organization, i.e., ad-hoc or systematic reuse.
2. Easiness to collect the required metrics: Each model uses various metrics to calculate reuse-related costs. Therefore, a key consideration when selecting a model is the availability of the required metrics in the organization.

Manual estimation can be used if the needed metrics are not available or time-consuming to collect these metrics. The manual estimate can be based on (i) the size of a similar task completed previously, (ii) the complexity of the task, and (iii) the experience level of the software practitioners.

We identified two methods and their metrics that could help in calculating the reuse related costs of AATs. These methods are described below.

Amar and Coffey's reuse cost calculation method [70] Amar and Coffey attempted to provide a unified unit of measure capable of calculating the costs that occurred during a reuse instance [70]. They claim that the reuse-related expenses are directly related to the time spent on reuse activities. They claim that the time spent during each of these activities should be considered. The method and metrics proposed by Amar and Coffey are described below.

$$\% \text{ of reuse cost avoided per AAT} = \\ = \left[S - (T+U) \times \frac{\frac{N}{i}}{B} - S \times \frac{M}{B} \right] \times 100 \quad (2)$$

where S is the search hit rate (i.e., the number of attempted searches yielding a reuse instance is divided by the number of total searches), T is time to locate a reusable AAT, U is time to understand the AAT, N is a number of AATs analyzed, i is a number of reuse instances of AAT, M is time to integrate (reuse) the (e.g., after adapting a reusable artifact), and B is time to develop an AAT from scratch. Further details on the working and evaluation of this method are found in study [70].

Irshad et al.'s reuse cost calculation method [15] According to Irshad et al., their proposed metric can calculate cost avoidance by reusing any software artifact [15]. Their provided instrument considers the effort spent reusing an artifact vs. effort spent on developing it from scratch. The basic formula is described below in the context of this study.

$$\text{Reuse Costs} = (O - I) \times H \quad (3)$$

where O is the personnel hours when an AAT is developed from scratch, I is the personnel hours spent on the adaptation of reusable AAT (i.e., changing an artifact to match the needs) and, H is the cost of one personnel hour. According to Irshad et al., historical data or expert opinion can estimate the personal hours when AAT is developed from scratch. Further details on the instrument and its evaluation can be found in the study [15].

Example: For the sake of this example, we can assume that the test developer selects a reuse cost model [15] i.e., Equation 3.

4.1.7. A7: Calculate cost of reuse

In this activity, the selected cost-avoidance method and its metrics are applied to calculate the cost of reusing the identified AATs. The reuse costs are only calculated for the artifact that is deemed relevant. The activity can help in decisions like should a new test be developed or existing ones are modified. This activity can be performed with the help of automated scripts or manually.

Example: Using the metrics and method, the test developer calculates that the new development cost is 40 person-hours, and the cost of Reusing by Adaptation Scenario 2 is 30 person-hours. Therefore, it makes sense to reuse Scenario 2 to develop a new AAT as it saves ten person-hours.

4.1.8. A8: Develop new AAT by reuse

In this activity, a new AAT is developed by reusing the reusable AAT. This activity takes place if the reuse is deemed as cost-effective in activity A7. During this activity, one of the adaptation templates is applied to develop the new AAT that supports future reuse opportunities (present in Section 4.1.3).

Test developer performs this activity manually. When developing a new AAT by reuse (using the proposed process), the following pre-requisites are needed:

– The format in which the new AAT will be written so that it becomes a reusable asset for the future.
– The cost-efficient reusable test case(or test cases) which will be used to develop a new AAT (or parts of a new AAT).

Example: When the reuse is considered beneficial, we assume that the test developer uses "Structuring" as a reuse approach to developing a new AAT scenario that test registration of a user account and viewing products below. The grey colored lines show the reuse from the existing scenario (Scenario 2 described in activity A4).

Scenario 3: *As a user I should be able to create account and view product*
Given A product is configured in the system
AND A login system is present for the product
AND A user is able to access the GUI Registration system
When User Registration is successful
AND User sends a Get request to fetch the products
Then User is able to view the product
AND a new user account is created

4.1.9. A9: Develop a new AAT

In this activity, the development of a new AAT from scratch takes place. This activity happens if the reuse from existing AATs is not possible, i.e., no relevant reusable AAT or reuse has unfavorable cost vs. benefits. While developing the new AATs, the vocabulary and existing rules of writing an AAT are considered. An adaptation templates is selected to develop the new AAT that supports future reuse opportunities (present in Section 4.1.3). A test developer performs this activity manually.

The example from activity A8 (Scenario 3) shows a new AAT if developed from scratch.

4.1.10. A10: Add new AAT to repository

The final activity is to add the newly developed (or reused) AAT into the existing test repository. These steps help in improving and developing test-base. This activity is performed manually.

An example of the repository is a Git repository[10], which is used by the majority of the software development organization to version-control the software artifacts.

4.2. Evaluation and refining of systematic reuse process

We first evaluated the proposed process with experienced practitioners who have first-hand experience working with AATs. Later, the authors assessed the process using it in an AAT suite from a large-scale software product.

4.2.1. Qualitative feedback of software practitioners

The industrial evaluation with five participants is conducted with the help of a questionnaire. The results from each section, based on UTAUT [61], are described below:

Performance expectancy: According to practitioner one, combining the reuse process and the reuse cost is very beneficial. Other participants also found this solution beneficial for the reuse of AATs. The quotes below describe the specific statements from the respondent of the survey questionnaire.

> "It guides a systematic approach, and it will help in optimizing the AAT process. Provided if it does not involve additional execution cost and the process is automated." (P1)
>
> "The possible benefit of the process is the reduction in the effort to write the new test case." (P2)
>
> "In my point of view, this process will really help the software practitioner to select the test cases and use it for automated acceptance

[10]https://git-scm.com/

tests. The use of this process can be cost effective in sense of saving time by selecting the test cases with respect to the score of each test." (P3)

"In case of low assess relevance and less cost of reuse, it will improve the quality of the tests by having already reliable tests. Also, it'll reduce the time to test a functionality that is closer to already existing and selected TC." (P3)

Effort expectancy. The participants, overall, seems happy with respect to the effort expectancy of the proposed process. They suggested that activities in the process are easy to follow. However, they posed a few interesting questions (in quotes below) that we have attempted to resolve in the final design of the proposed process. Some of the quotes from the practitioners are given below:

"The process seems to be simple and easy to use, provided if process tasks are automated." (P3)

"The only thing I have found its hard to implement this process in the existing project because it takes time to change the process and this process has involved many steps but this one time cost of implementation can save a lot in future." (P4)

"The relevance and cost estimation should be automated. The increasing number of reusable artifacts will affect the efficiency of search. So some mechanisms should be introduced to speed up the search, e.g., indexing etc." (P3)

When asked about ease of use, four practitioners marked the process as easy or very easy to use. One practitioner suggested it as "Normal" to use. The results are shown in Table 6.

Facilitating Condition. The participants stated that the solution is right to have, but the organization-level activities should be reconsidered because they add an over-head in the process. The respondents of the survey suggested the following improvements with respect to facilitating conditions.

"Organizational level activities should not be the part of the process, rather these should be the pre-requisites of the process." (P1)

"Steps A2 and A3 can cause possible delay, so they should be a pre-requisite to the process, and not a part of the process." (P3)

"Reassessment after cost evaluation with other closely related test cases." (P4)

From the evaluation, we found that:

- The systematic reuse process saves the time to write new AATs.
- The organizational level activities are one-time activities, and test developers should skip these activities.
- There can be more reuse candidates than one. A new AAT can use parts of multiple existing reusable AAT.
- The scripts implementing search and assessment activities should be part of the systematic reuse process. In the future, the focus should be to automate many of these activities.

4.2.2. Demonstrating industrial application of the proposed process

An industrial AAT was implemented using the activities of the proposed process. A summary of quantitative data captured during the evaluation is shown in Table 7 and qualitative information mentioned in the sections below. The context and details of the industrial demonstration are discussed in the research approach (Section 3.2.2 Phase 5: Test candidate design elements).

Details of tasks in each activity. Each activity inside the proposed process consists of one

Table 6. How easy it is to use the process

Participant	Very Easy	Easy	Normal	Difficult	Very difficult
Participant 1	✓				
Participant 2		✓			
Participant 3		✓			
Participant 4			✓		
Participant 5	✓				

or more tasks that are performed in the activity. The tasks performed in each activity are assessed, and important lessons are documented. It was noted that A4, A5, A6, and A8 activities take more time and contain multiple tasks per activity. The tasks under each activity are described in Table 7 and in the sub-sections below:

A1: Select Keywords: The test developer (author) selected 3 keywords (a) REST (b) <Interface Name> (c) a description of performance requirement.

Select AAT format: From the available choices (BDD or keyword-driven), the test developer selected keyword-driven as AAT format. The existing AATs were also written in keyword-driven format.

Select artifact adaptation template: "Parametrization" is selected as artifact adaptation template because parameterization is by default supported by the keyword-driven frameworks, and existing AAT is also based on keyword-driven tests.

Search for reusable AATs: The searching of AATs was executed using the script provided as part of the proposed process (See [69]). Before the search is executed, libraries required for the script are installed. The existing AATs are checked-out from the repository. Three files, each containing one keyword, are created in the same directory. The NCD script is executed to identify the AATs closely matching the keywords. The first search did not yield AATs that were closely matching with keywords. Later, keywords were changed (mentioned in the activity "select keywords" above) that produced better results. The output of this activity is available online [78].

Assess Relevance of AATs: The output of search activity is sorted in ascending order. The output of the five pairs with lowest NCD values (i.e., similar) is shown in Table 8. The pairs with the lowest NCD values are selected for the analysis. After manually analyzing the content of selected AATs (from pairs with the lowest NCD), two AATs (testcase27, testcase26) are selected for reuse purposes.

Select reuse cost method and metrics: The verification team keeps track of tasks in a ticketing system. The time spent on each task in each phase (in backlog, in development, in Done) is present in the system. Therefore, it was decided to use a person-hour based metric and model. The reuse cost calculation method proposed by Irshad et al. [15] was selected to calculate reuse costs.

Calculate cost of reuse: The development time of testcase27 and testcase26 was four weeks for each test case. We estimated that by reusing (without change) some parts of the testcase26 (related to SUT configuration and test data generation and cleaning), we could save three weeks of development effort. Other parts of testcase27 (related to validation) support parametrization. These parametrization supporting parts were also reused using different values for the parameters. An example of parametrization is found on a link here [83].

Develop new AAT by reuse: The new AAT was developed using the parameterization approach (See example of parametrization [83]. The reusable lines from the testcase27 already supported parametrization. Seven out of twenty-five new AAT lines were reused (by using different parameter values) from the testcase27.

Add new AAT to repository: Once the AAT is ready and approved by the reviewers, the AAT is pushed to the central repository. This task triggers the build on the build server, executing the AAT.

Number of tasks performed in each activity. The number of tasks in an activity may show the effort required to perform the activity. An activity with a large number of tasks may require more effort from the practitioners. The test developer (one of the authors) kept a record of the number of tasks performed in each activity, e.g., searching, changing code. The tasks varied from 1 task to 3 tasks in activity. The number of tasks performed in each activity is described in Table 7.

Effort to create an AAT. To capture this construct, we measured the time taken during each activity. The time taken by each activity is shown in Table 7. The development (writing, testing, refactoring) of the AAT took the most time (1 week). Other activities in the process took less than 30 minutes each. The practitioner tracked the time spent on the task, and the practitioner used minutes to track the precise time. In the work management tool used by the organization,

Table 7. Evaluation: Time spent, no of tasks, the tasks performed during each activity of the process, description of the tasks and comparison with existing (manual) activity

Activity	Time spent	No. of tasks	Description of tasks in the activity	Comparison with existing (manual) activity
A1: Select Keywords	5 minutes	2	(a) Reading requirements description. (b) Deciding suitable keyword.	Not needed in manual process
A2: Select AAT format*	1 minute	1	Selecting "Keyword-driven" as AAT format.	No such activity exists, a practitioner decides the format he has previous experience with.
A3: Select artifact adaptation template	1 minute	1	Selecting "Parametrization" as adaptation template to support develop with reuse.	No such activity exists.
A4: Search reusable AATs*	27 minutes	3	(a) Configuring libraries for NCD script, a one-time task (20 minutes). (b) Writing keywords from activity A1 in separate files (5 minutes). (c) Executing script in the repository (takes 2 minutes).	A practitioner uses his/her experience from test suite.
A5: Assess relevance of AATs*	30 minutes	3	(a) Sort and analyse the output of NCD script (excel file) 10 minutes. (b) Select top relevant AATs and analyse for relevance (10 minutes). (c) Select one AAT most suitable for a new AAT (10 minutes).	A practitioner uses his/her experience of domain.
A6: Select reuse cost method and metrics	11 minutes	2	(a) Analysis of the metric present in the organization/unit (10 minutes). (b) Select suitable cost model (1 minute).	No such activity exists.
A7: Calculate cost of reuse	15 minutes	1	Apply cost model to AAT (15 minutes).	No such activity exists.
A8: Develop new AAT by reuse*	1 week	4	(a) Selecting reusable parts of AAT from task "C" in activity A5 (20 minutes). (b) Adapting reusable parts to fit the newly developed AAT (24 hours). (c) Testing the new AAT (10 hour). (d) Refactoring the new AAT (6 hours).	A test case by reuse is developed using activities, A1–A7.
A9: Develop a new AAT*	0	0	Not performed.	A test case from scratch is developed using activities, A2 and A3.
A10: Add new AAT to repository*	1 minute	3	Using commands: git add <filename> and git commit -m "<Message>" andgit push	Similar to manual process.

* shows activity exists in manual and automated process.

the development of the new AAT took nearly 5-working days.

The time spent on a similar existing AAT was identified from the organization's archived data (using Git history, we found the development task and determined the time spent in the task development phase). A similar AAT was developed in four weeks, as per the development phase of the task. This difference is because, in the newly developed AAT, the test data setup,

Table 8. NCD values of comparison between Keywords and existing AATs

AAT 1	Keywords	NCD value
testcase27	keywords.txt	0.169
testcase26	keywords.txt	0.174
testcase84	keywords.txt	0.294
testcase25	keywords.txt	0.299
testcase83	keywords.txt	0.307

the SUTs state configuration, and the test data deletion parts were reused from an existing AAT. **Magnitude of reused statement.** Utilizing the reuse cost calculation activity is essential to recognize benefits before reusing any AATs. The cost savings through AAT's reuse depends on (i) the number of reusable AAT statements and (ii) the functionality corresponding to the reused statement. For example, only seven out of 25 lines were reused in the demonstration. However, these seven reused lines perform functionality that is time-consuming to develop, so the cost savings were almost 75% (1 week when reusing vs. 4-weeks of development time from scratch). Another example can be a case where many statements are reused, but these statements require a small amount of development time when developing from scratch; in that case, cost savings may not be a lot.

The evaluation from the industrial assessment identified the following lessons:

– Finding suitable keywords may require multiple iterations before finding the most useful reusable AAT.
– There were more than one AATs identified as reusable during the assessment of reusable AAT. There should be a guideline on how to select the best out of the possible reusable artifacts.
– The automated scripts provide a mechanism to search and assess the existing AATs.

4.3. Final version of systematic reuse process

Following changes were introduced in the proposed process, based on the feedback from practitioners and application in an industrial setting:

– A condition is introduced to skip organizational-level activities if these are already defined, as per the participants' suggestions.
– A condition is modified to select multiple candidates and perform reuse cost calculations on each of these candidates.
– A new activity is introduced to evaluate the most suitable candidate from a list of candidates having lower costs and requiring fewer changes. The activity takes input on a list of

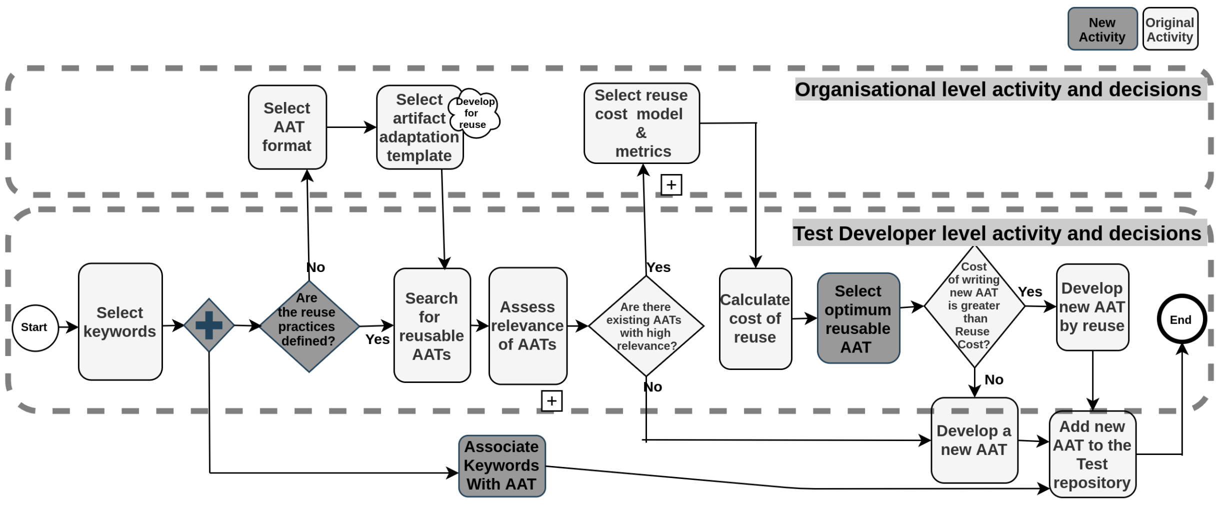

New Activity
Original Activity
Organisational level activity and decisions
Select AAT format
Select artifact adaptation template
Develop for reuse
Select reuse cost model & metrics
Test Developer level activity and decisions
Start
Select keywords
Are the reuse practices defined?
No
Yes
Search for reusable AATs
Assess relevance of AATs
Are there existing AATs with high relevance?
Yes
No
Calculate cost of reuse
Select optimum reusable AAT
Cost of writing new AAT is greater than Reuse Cost?
Yes
No
Develop new AAT by reuse
End
Develop a new AAT
Add new AAT to the Test repository
Associate Keywords With AAT

Figure 3. Final Version: A systematic reuse process for automated acceptance tests (AATs)

AATs with high relevance and low costs and lists the most feasible reusable AAT.
– A new activity, Associate Keywords with AAT, is introduced that stores the selected keywords when storing the new test case in the repository. These keywords help in categorizing the test cases and optimize the search functionality.
– The activities that can be automated with a script's help are mentioned in Table 3.

The final version of the process is shown in Figure 3.

5. Discussion

This section provides a discussion on the analysis of the evaluation, characteristics and benefits of the proposed process.

5.1. Analysis of industrial evaluation

This section describes the findings and analysis of industrial evaluation.

Analysis on performance expectancy. Performance expectancy has implications on using the proposed reuse process if the process is advantageous for the software practitioners [12]. If the process is perceived as advantageous, then it is likely that other software practitioners and the software industry embrace the proposed process. In the evaluation, practitioners provided positive feedback with regards to performance expectancy. The practitioners listed the following advantages regarding performance expectancy of the process:
– A systematic process to support reuse of automated acceptance tests.
– The activities of searching and assessment of a reusable test case can help software practitioners.
– New tests can be developed with less effort.

This feedback was re-confirmed during the industrial demonstration of the process where the test developer (an author) followed the activities and techniques proposed in the systematic reuse process and successfully developed the test case.

Analysis on effort expectancy. Effort expectancy identifies the level of ease to use the proposed reuse process. The easiness of using the process has direct implications on the adaptability of the process in the industry. Software practitioners found the process easy to use. However, to increase the effort expectancy, practitioners suggested that the majority of the process activities should be automated with the help of scripts. Practitioners also suggested that there are many activities involved in the process, and implementing these activities in their existing process can be challenging. They suggested automating these activities to reduce the impact of a large number of activities.

During the industrial demonstration, it was noted that the time spent on the activities of the process and the number of tasks in each activity are low in numbers. The activity with the highest number of tasks (4) and time is taken (1 week) is developing the new AAT. Overall the activities in the process were easy to use for the test developer well familiar with the process (an author).

Analysis on facilitating conditions. The process's impact on the development of AATs is evaluated using the facilitating condition construct. The practitioners, during the evaluation, suggested changes that can help improve the construct of facilitating conditions. Practitioners believed that the proposed process could become better by:
– making few activities pre-requisite to execute (only once) when the process is applied in any organization,
– allowing assessment of more than one reusable AATs before selecting the final reusable AAT.

The tool support and the competence needed to use the process was evaluated in the industrial demonstration. It was noted that tools and libraries need to be installed before running the provided scripts for searching and assessment. The calculation of reuse cost required metrics that were already available in the organization, i.e., better-facilitating conditions.

5.2. Guidelines for AATs reuse

The activities of proposed process act as guidelines for reusing the AAT. In the existing literature, limited studies have discussed the reuse of AAT, and this study provides step-by-step guidance for developing with reuse and developing for reuse. The input, the output, the actor and techniques relevant for each activity are described in Table 3 and Section 4. Practitioners and researchers can use this information as guidelines for supporting the reuse of AATs.

5.3. Compare reuse opportunities

The two activities (i.e., assess the relevance of AAT and reuse cost analysis) in the process can help the practitioners evaluate a reuse opportunity's effectiveness. With these two activities, the practitioners have an instrument to evaluate and compare the value of reusing different AATs. Based on their comparison, they can select the AAT, which is more suitable for their purpose.

5.4. Flexible techniques

Section 4 provides different techniques that apply to the activities of reuse process. The implementation of some of these techniques is also provided to support the practitioners. However, an organization can add its own techniques to search, assess, or calculate reuse costs if it wants to use customized techniques. The reuse process is not bound to fix a set of searching and calculation techniques.

5.5. Support for automation

Several steps in the proposed process are either automated already or have the potential (e.g., selecting keywords) to be automated. This can result in cost savings for the practitioners by (i) reducing time to develop an AAT and (ii) reducing the time to analyze, search, and assess the AATs. In future work, we want to provide automated scripts for activities A1, A6, and A7 from Table 3.

5.6. Verdict on the diversity of AAT-suite

The activities of the process can also be used to assess the diversity in an organization's AAT suite. A higher diversity means that the test suite has more test coverage. The search and assessment using NCD provides pair-wise comparison values of all the AATs. These values can be a good indicator of diversity in the AAT suite. A suite with low diversity could have many pairs with low NCD values (i.e., very similar to each other), indicating that refactoring is needed to diversify the AATs or remove the duplicates.

5.7. Tool support for reuse of AATs:

AATs are text-based artifacts different from traditional code-based test cases. IDE features often support the code-based reuse process to detect duplicates, detect similar usage, provide modularization of code snippets, etc. These basic reuse features are not yet mature enough for non-code artifacts. Therefore, we have provided easy to use techniques and scripts that can be applied to support the reuse of text-based AATs.

5.8. Increased coupling

Existing research literature has described the issue with decreased maintainability among AATs [7]. A key concern when developing by reusing parts of different reusable AATs is an increase in coupling (dependency between test cases) in the test base [84]. This increase in coupling decreases the maintainability of the test cases. Therefore, during the activity A5 (Assessing the relevance of reusable AATs), it is vital to consider the increase in coupling between the reused and reusable AATs.

5.9. Comparison with existing literature

Park and Maurer proposed three strategies that can be used to develop reusable assets in software product lines [85]. These three strategies are (i) proactive mode in which organization makes upfront investment in developing reusable assets, (ii) reactive mode in which reusable assets are developed when needed, and (iii) extractive mode in which existing product is reused [85]. Our proposed process can be classified as a reactive model, in which we develop new reusable assets when there is a need for writing a new test.

In their study on variability management in software product lines, Kiani et al. proposed a method in which reusable assets are developed on demand when the need arises [86]. This is similar to our proposed approach in which reusable AATs are created when there is a need to write a new test, i.e., no upfront costs are required. In addition, de Silva proposed a software product line approach that uses automated acceptance tests to link scoping of requirements, implementation, and testing [87]. Our proposed process compliments this SPL-based study by suggesting a reuse-based approach to derive the reusable AATs along with requirements, implementation, and testing.

In a study by Mink et al., software practitioners suggested that specifying the granular details of automated acceptance tests, i.e., format and details of AATs is cumbersome and requires more time from them [3]. In another study, Mink et al. investigated executable acceptance testing. They found that AATs help in (i) preserving the domain knowledge and (ii) improve the communication among the developers [3]. Our study identified similar findings during the evaluation of the proposed process using experienced software practitioners. Thus, our approach may help the software practitioner specify the details of the automated acceptance tests by providing a specific AAT format and suggesting existing reusable AATs.

5.10. Scalability of Approach

Searching for reusable software artifacts is known as a time-consuming process (with high costs) during software reuse [43]. The cost of searching and retrieving a reusable software artifact grows when new artifacts are added to the repository [43]. Therefore, the scalability of the searching techniques is a vital characteristic to support future reuse opportunities. We evaluated performed an evaluation of the search approach. In an examination (by the authors) with a specification base of 500 AATs, reusable candidates were identified in less than 5 minutes using the scripts provided as part of the proposed process (See script [69].)

6. Threats to validity

Runeson et al. [88] classified the validity threats into four types (reliability, construct validity, internal validity, and external validity). These threats to the study's validity and the measures to address these validity threats are discussed in this section.

Internal validity deals with the case when the factors studied by the researchers are affected by any other factors unknown to the researchers. This threat applies to the design and development of the questionnaire for industrial evaluation. The questionnaire design can be classified as "self-administrated," i.e., online form. As the proposed process is developed for software organization, we evaluated the process using the constructs suggested by the unified theory of acceptance and technology use (UTAUT) [61]. These three constructs (performance expectancy, effort expectancy, and facilitating condition) help the authors to develop evaluation questions related to the proposed process systematically. Each question in the questionnaire is mapped to a construct that it addresses. The details are provided in Section 3. We believe that we have addressed this threat to this investigation's internal validity by following a systematic method to design and develop the questionnaire.

External validity concerns with the generalization of results. The proposed process developed in this study is considered useful for large-scale software organizations. Hence, we evaluated the proposed process using an industrial use-case for large-scale systems. During the evaluation, the process's completeness and usability for large-scale product organizations are evaluated. Furthermore, we involved five experienced practitioners from two large-scale organizations to evaluate the proposed process and provide feedback.

The experienced practitioners involved in this study worked in different roles in large-scale organizations. They were selected because they have a prior understanding of automated acceptance testing and reuse. As reuse of AATs is still a new area, it is difficult to find practitioners who understand these concepts. Furthermore, we involved practitioners from two large-scale organizations to improve the generalizability of the proposed process. However, the authors believe that a reuse process should be applied and evaluated in the industry before the results of this study are considered generalizable, which is part of our future work.

Reliability deals with how the data collection and analysis are dependent on the researcher. The researcher independently conducted the first evaluation to validate a process in the industry setting. This practice is called lab validation by Gorscheck et al. [62]. Since this evaluation was conducted by a software practitioner (an author), the threat to the validity of data collection and analysis exists. To mitigate this threat to the evaluation's validity, in the second part of the evaluation (using experienced practitioners), the data collection was done in an online form without the author's active involvement in filling the form. A critical threat to the reliability of the study is related to the response bias of survey respondents. The responses from the practitioners reflect the belief of those practitioners, and these beliefs may be contrary to real-world contexts. We believe that this threat is relevant to this study, and in future evaluations of the proposed process, work is needed to address this concern. The usage of online form reduces the chance of losing any valuable feedback from practitioners. For data analysis, we used a systematic method called constant comparison to interpret the online questionnaire's feedback.

Construct validity deals with how well the study captures the intended constructs. During the evaluation, each subject involved presented the motivation, background, and walk-through of the proposed process during online sessions. The subjects asked questions about the study, the process, and the questionnaire during these presentations. Furthermore, one of the study authors has considerable experience working in the same domain and industry. Hence, he was able to explain the concepts in the language/terms understood by the subjects. This step helps in reducing confusion or ambiguities related to the reuse process.

Furthermore, the questionnaire (available at [73]) provides a detailed description of concepts, the activities used in the process, techniques used in these activities, and a working example of the proposed process. These details were provided to the respondents to make sure the study captures the intended constructs.

7. Conclusion

The reuse of automated acceptance tests help develop new tests cheaply, quickly, and with high quality. However, the textual nature of these tests makes the reuse of these tests different from code-based tests. In this investigation, we describe a systematic reuse process for text-based automated acceptance tests. We constructed this reuse process using the method engineering and performed an initial evaluation of the reuse process before applying it to the industry.

RQ 1: How can the cost-effective systematic reuse of text-based AATs be achieved? The construction of systematic reuse process starts with the identification of the motivation and requirements of the new process. The identified requirements are (i) the process should consider development with reuse and development for reuse, (ii) the process should be independent of several text-based formats (e.g., BDD, keywords) and frameworks of automated acceptance tests (e.g., Cucumber, Robot Framework), and (iii) the cost of reuse should be calculated before developing a new test by reusing existing tests. For these three requirements, we identified and tailored existing methods present in the literature. The final outcome is a systematic reuse process that supports the reuse of various types of text-based automated acceptance tests. The process activities are divided into two types, i.e., organizational level activities and test developer-level activities. We provided expected input, expected output, actor, examples, and techniques (automated using scripts) suitable for the process's activities.

RQ 2: How does the systematic reuse process, from RQ 1, perform concerning performance expectancy, effort expectancy, and facilitating conditions in the industrial context? After constructing the process, an elementary industrial evaluation assesses the performance expectancy, effort expectancy, and facilitating conditions concerning the process. This evaluation is performed to sanity-check and improves the process before it is ready for a long and detailed industrial evaluation. Initially, five participants with considerable experience in automated acceptance testing provided qualitative feedback on the systematic reuse process. They found that the process can save time and reduce the effort to write and maintain automated acceptance tests. The practitioners suggested that activities are easy to use, and the reuse cost metrics are easy to find and apply using the proposed techniques. They suggested changes in the process, and these changes were incorporated in the final version of the process. This evaluation shows promising results concerning the processes' performance expectancy, effort expectancy, and facilitating conditions.

Later an illustration of the usage of a process in the industry is conducted. During the evaluation, a new test case is developed by reusing existing automated acceptance tests. This evaluation's objective was to identify and sanity-check the tasks in the process's activities and evaluate their complexity. One of the authors, from the same organization, conducted this demonstration. The evaluation helped identify several different granular tasks required to perform in each activity of the process. The number of tasks varies from 1 to 4 between different activities. These identified tasks can act as guidelines when using the process in the industry. The evaluation also recorded the time spent in each activity. It was noted that most of the time is spent developing the new test by reusing existing automated acceptance tests. The development time with the reuse process was 4-times quicker than developing a new artifact from scratch.

In the future, we want to have a longitudinal investigation on the process's transfer and usage in the industry, as application and evaluation of the process may take a long duration and resources. The current study enables the researchers to sanity-check the process before evaluating it in industrial settings. Secondly, in the next study, we want to automate most of the proposed process activities and evaluate the process using automated activities. Furthermore, we want to evaluate the precision and recall of the search and assessment functionality proposed in this process.

References

[1] M.J. Harrold, "Testing: A roadmap," in *Proceedings of the Conference on the Future of Software Engineering*, 2000, pp. 61–72.

[2] W.E. Wong, J.R. Horgan, S. London, and H. Agrawal, "A study of effective regression testing in practice," in *Proceedings., The Eighth International Symposium on Software Reliability Engineering.* IEEE, 1997, pp. 264–274.

[3] G. Melnik and F. Maurer, "Multiple perspectives on executable acceptance test-driven development," in *International Conference on Extreme Programming and Agile Processes in Software Engineering.* Springer, 2007, pp. 245–249.

[4] "Standard glossary of terms used in software testing," International Software Testing Qualifications Board, Standard 3.5, 2020. [Online]. https://www.istqb.org/downloads/glossary.html

[5] B. Haugset and G.K. Hanssen, "Automated acceptance testing: A literature review and an industrial case study," in *Agile Conference.* IEEE, 2008, pp. 27–38.

[6] M. Huo, J. Verner, L. Zhu, and M.A. Babar, "Software quality and agile methods," in *Proceedings of the 28th Annual International Computer Software and Applications Conference, 2004. COMPSAC 2004.* IEEE, 2004, pp. 520–525.

[7] J. Weiss, A. Schill, I. Richter, and P. Mandl, "Literature review of empirical research studies within the domain of acceptance testing," in *42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA).* IEEE, 2016, pp. 181–188.

[8] W.B. Frakes and K. Kang, "Software reuse research: Status and future," *IEEE Transactions on Software Engineering*, Vol. 31, No. 7, 2005, pp. 529–536.

[9] R. Capilla, B. Gallina, C. Cetina, and J. Favaro, "Opportunities for software reuse in an uncertain world: From past to emerging trends," *Journal of Software: Evolution and Process*, Vol. 31, No. 8, 2019, p. e2217.

[10] W.B. Frakes and S. Isoda, "Success factors of systematic reuse," *IEEE software*, Vol. 11, No. 5, 1994, pp. 14–19.

[11] D. Rombach, "Integrated software process and product lines," in *Software Process Workshop.* Springer, 2005, pp. 83–90.

[12] M. Ramachandran, "Software re-use assessment for quality," *WIT Transactions on Information and Communication Technologies*, Vol. 9, 1970.

[13] E.S. de Almeida, A. Alvaro, D. Lucrédio, V.C. Garcia, and S.R. de Lemos Meira, "Rise project: Towards a robust framework for software reuse," in *Proceedings of the International Conference on Information Reuse and Integration.* IEEE, 2004, pp. 48–53.

[14] J.S. Poulin, *Measuring software reuse: principles, practices, and economic models.* Addison-Wesley Reading, MA, 1997.

[15] M. Irshad, R. Torkar, K. Petersen, and W. Afzal, "Capturing cost avoidance through reuse: systematic literature review and industrial evaluation," in *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering.* ACM, 2016, p. 35.

[16] A. Davies, T. Brady, and M. Hobday, "Charting a path toward integrated solutions," *MIT Sloan management review*, Vol. 47, No. 3, 2006, p. 39.

[17] W.E. Wong, "An integrated solution for creating dependable software," in *Proceedings 24th Annual International Computer Software and Applications Conference. COMPSAC2000.* IEEE, 2000, pp. 269–270.

[18] R.J. Mayer, J.W. Crump, R. Fernandes, A. Keen, and M.K. Painter, "Information integration for concurrent engineering (IICE) compendium of methods report," Knowledge Based Systems Inc., Tech. Rep., 1995.

[19] M. Rahman and J. Gao, "A reusable automated acceptance testing architecture for microservices in behavior-driven development," in *Symposium on Service-Oriented System Engineering (SOSE).* IEEE, 2015, pp. 321–325.

[20] G. Meszaros, "Agile regression testing using record and playback," in *Companion of the 18th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications.* ACM, 2003, pp. 353–360.

[21] A.K. Onoma, W.T. Tsai, M. Poonawala, and H. Suganuma, "Regression testing in an industrial environment," *Communications of the ACM*, Vol. 41, No. 5, 1998, pp. 81–86.

[22] P. Hsia, D. Kung, and C. Sell, "Software requirements and acceptance testing," *Annals of Software Engineering*, Vol. 3, No. 1, 1997, pp. 291–317.

[23] G.K. Hanssen and B. Haugset, "Automated acceptance testing using fit," in *42nd Hawaii International Conference on System Sciences.* IEEE, 2009, pp. 1–8.

[24] E. Pyshkin, M. Mozgovoy, and M. Glukhikh, "On requirements for acceptance testing automation tools in behavior driven software development," in *Proceedings of the 8th Software Engineering Conference in Russia (CEE-SECR)*, 2012.

[25] G. Liebel, E. Alégroth, and R. Feldt, "State-of--practice in GUI-based system and acceptance testing: An industrial multiple-case study," in *39th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA).* IEEE, 2013, pp. 17–24.

[26] H. Munir and P. Runeson, "Software testing in open innovation: An exploratory case study of the acceptance test harness for Jenkins," in *Proceedings of the International Conference on Software and System Process*, 2015, pp. 187–191.

[27] G. Melnik and F. Maurer, "The practice of specifying requirements using executable acceptance tests in computer science courses," in *Companion to the 20th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, 2005, pp. 365–370.

[28] M. Hayek, P. Farhat, Y. Yamout, C. Ghorra, and R.A. Haraty, "Web 2.0 testing tools: A compendium," in *International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT).* IEEE, 2019, pp. 1–6.

[29] P. Gandhi, N.C. Haugen, M. Hill, and R. Watt, "Creating a living specification using FIT documents," in *Agile Development Conference (ADC'05).* IEEE, 2005, pp. 253–258.

[30] D. North, "Introducing behaviour driven development," *Better Software Magazine*, 2006.

[31] E.C. dos Santos and P. Vilain, "Automated acceptance tests as software requirements: An experiment to compare the applicability of fit tables and gherkin language," in *International Conference on Agile Software Development.* Springer, 2018, pp. 104–119.

[32] C. Solis and X. Wang, "A study of the characteristics of behaviour driven development," in *37th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA).* IEEE, 2011, pp. 383–387.

[33] R. Hametner, D. Winkler, and A. Zoitl, "Agile testing concepts based on keyword-driven testing for industrial automation systems," in *IECON 2012-38th Annual Conference on IEEE Industrial Electronics Society.* IEEE, 2012, pp. 3727–3732.

[34] E. Bache and G. Bache, "Specification by example with gui tests-how could that work?" in

International Conference on Agile Software Development. Springer, 2014, pp. 320–326.

[35] A.C. Paiva, D. Maciel, and A.R. da Silva, "From requirements to automated acceptance tests with the RSL language," in *International Conference on Evaluation of Novel Approaches to Software Engineering.* Springer, 2019, pp. 39–57.

[36] M. Soeken, R. Wille, and R. Drechsler, "Assisted behavior driven development using natural language processing," in *International Conference on Modelling Techniques and Tools for Computer Performance Evaluation.* Springer, 2012, pp. 269–287.

[37] C. Deng, P. Wilson, and F. Maurer, "Fitclipse: A fit-based eclipse plug-in for executable acceptance test driven development," in *International Conference on Extreme Programming and Agile Processes in Software Engineering.* Springer, 2007, pp. 93–100.

[38] C.Y. Hsieh, C.H. Tsai, and Y.C. Cheng, "Test-Duo: A framework for generating and executing automated acceptance tests from use cases," in *8th International Workshop on Automation of Software Test (AST).* IEEE, 2013, pp. 89–92.

[39] C.W. Krueger, "Software reuse," *ACM Computing Surveys (CSUR)*, Vol. 24, No. 2, 1992, pp. 131–183.

[40] D.M. Rafi, K.R.K. Moses, K. Petersen, and M.V. Mäntylä, "Benefits and limitations of automated software testing: Systematic literature review and practitioner survey," in *Proceedings of the 7th International Workshop on Automation of Software Test.* IEEE Press, 2012, pp. 36–42.

[41] W. Frakes and C. Terry, "Software reuse: metrics and models," *ACM Computing Surveys (CSUR)*, Vol. 28, No. 2, 1996, pp. 415–435.

[42] W. Tracz, "Where does reuse start?" *ACM SIGSOFT Software Engineering Notes*, Vol. 15, No. 2, 1990, pp. 42–46.

[43] T. Ravichandran and M.A. Rothenberger, "Software reuse strategies and component markets," *Communications of the ACM*, Vol. 46, No. 8, 2003, pp. 109–114.

[44] P. Mohagheghi and R. Conradi, "Quality, productivity and economic benefits of software reuse: A review of industrial studies," *Empirical Software Engineering*, Vol. 12, No. 5, 2007, pp. 471–516.

[45] V. Karakostas, "Requirements for CASE tools in early software reuse," *ACM SIGSOFT Software Engineering Notes*, Vol. 14, No. 2, 1989, pp. 39–41.

[46] J.L. Cybulski, "Introduction to software reuse," *Department of Information Systems, The University of Melbourne, Parkville, Australia*, Vol. 11, 1996, p. 12.

[47] W. Lam, J.A. McDermid, and A. Vickers, "Ten steps towards systematic requirements reuse," *Requirements Engineering*, Vol. 2, No. 2, 1997, pp. 102–113.

[48] R.G. Fichman and C.F. Kemerer, "Incentive compatibility and systematic software reuse," *Journal of Systems and Software*, Vol. 57, No. 1, 2001, pp. 45–60.

[49] A. Genaid et al., "Connecting user stories and code for test development," in *Third International Workshop on Recommendation Systems for Software Engineering (RSSE).* IEEE, 2012, pp. 33–37.

[50] L. Crispin and T. House, "Testing in the fast lane: Automating acceptance testing in an extreme programming environment," in *XP Universe Conference.* Citeseer, 2001.

[51] L.P. Binamungu, S.M. Embury, and N. Konstantinou, "Maintaining behaviour driven development specifications: Challenges and opportunities," in *25th International Conference on Software Analysis, Evolution and Reengineering (SANER).* IEEE, 2018, pp. 175–184.

[52] M. Irshad, J. Börster, and K. Petersen, "Supporting refactoring of BDD specifications – An empirical study," *Information and Software Technology*, 2022.

[53] R. Angmo and M. Sharma, "Performance evaluation of web based automation testing tools," in *5th International Conference – Confluence The Next Generation Information Technology Summit (Confluence).* IEEE, 2014, pp. 731–735.

[54] S. Park and F. Maurer, "A literature review on story test driven development," in *International Conference on Agile Software Development.* Springer, 2010, pp. 208–213.

[55] Q. Xie, "Developing cost-effective model-based techniques for GUI testing," in *Proceedings of the 28th International Conference on Software Engineering.* ACM, 2006, pp. 997–1000.

[56] R. Borg and M. Kropp, "Automated acceptance test refactoring," in *Proceedings of the 4th Workshop on Refactoring Tools.* ACM, 2011, pp. 15–21.

[57] C. Schwarz, S.K. Skytteren, and T.M. Ovstetun, "AutAT: An eclipse plugin for automatic acceptance testing of web applications," in *Companion to the 20th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications.* ACM, 2005, pp. 182–183.

[58] B. Fitzgerald, N.L. Russo, and T. O'Kane, "Software development method tailoring at motorola,"

Communications of the ACM, Vol. 46, No. 4, 2003, pp. 64–70.

[59] P. Raulamo-Jurvanen, M. Mäntylä, and V. Garousi, "Choosing the right test automation tool: a grey literature review of practitioner sources," in *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*, 2017, pp. 21–30.

[60] A. Egbreghts, "A literature review of behavior driven development using grounded theory," in *27th Twente Student Conference on IT.*, 2017.

[61] V. Venkatesh, J.Y. Thong, and X. Xu, "Consumer acceptance and use of information technology: Extending the Unified Theory of Acceptance and Use of Technology," *MIS quarterly*, 2012, pp. 157–178.

[62] T. Gorschek, P. Garre, S. Larsson, and C. Wohlin, "A model for technology transfer in practice," *IEEE software*, Vol. 23, No. 6, 2006, pp. 88–95.

[63] M. Finsterwalder, "Automating acceptance tests for GUI applications in an extreme programming environment," in *Proceedings of the 2nd International Conference on eXtreme Programming and Flexible Processes in Software Engineering.* Addison-Wesley Boston MA, 2001, pp. 114–117.

[64] S. Park and F. Maurer, "An extended review on story test driven development," University of Calgary, Tech. Rep., 2010.

[65] H. Mili, F. Mili, and A. Mili, "Reusing software: Issues and research directions," *IEEE Transactions on Software Engineering*, Vol. 21, No. 6, 1995, pp. 528–562.

[66] R. Feldt, S. Poulding, D. Clark, and S. Yoo, "Test set diameter: Quantifying the diversity of sets of test cases," in *International Conference on Software Testing, Verification and Validation (ICST).* IEEE, 2016, pp. 223–233.

[67] W.H. Gomaa, A.A. Fahmy et al., "A survey of text similarity approaches," *International Journal of Computer Applications*, Vol. 68, No. 13, 2013, pp. 13–18.

[68] M. Irshad, K. Petersen, and S. Poulding, "A systematic literature review of software requirements reuse approaches," *Information and Software Technology*, Vol. 93, 2018, pp. 223–245.

[69] M. Irshad, *Source Code for Scripts*, 2021. [Online]. https://zenodo.org/record/4765079

[70] L. Amar and J. Coffey, "Measuring the benefits of software reuse-examining three different approaches to software reuse," *Dr Dobbs Journal*, Vol. 30, No. 6, 2005, pp. 73–76.

[71] K. Petersen and C. Wohlin, "Context in industrial software engineering research," in *3rd International Symposium on Empirical Software Engineering and Measurement.* IEEE, 2009, pp. 401–404.

[72] C. Wohlin, M. Höst, and K. Henningsson, "Empirical research methods in software engineering," in *Empirical methods and studies in software engineering.* Springer, 2003, pp. 7–23.

[73] M. Irshad, "Questionnaire: The reusability of automated acceptance tests," 2021. [Online]. https://zenodo.org/record/4765102

[74] J.S. Molléri, K. Petersen, and E. Mendes, "An empirically evaluated checklist for surveys in software engineering," *Information and Software Technology*, Vol. 119, 2020, p. 106240.

[75] B.G. Glaser, A.L. Strauss, and E. Strutzel, "The discovery of grounded theory; strategies for qualitative research," *Nursing research*, Vol. 17, No. 4, 1968, p. 364.

[76] J. Stoustrup, "Successful industry/academia cooperation: From simple via complex to lucid solutions," *European Journal of Control*, Vol. 19, No. 5, 2013, pp. 358–368.

[77] T.C. Lethbridge, S.E. Sim, and J. Singer, "Studying software engineers: Data collection techniques for software field studies," *Empirical software engineering*, Vol. 10, No. 3, 2005, pp. 311–341.

[78] M. Irshad, "Search and assessment data," 01 2021. [Online]. http://shorturl.at/juIZ6

[79] P.M. Vitányi, F.J. Balbach, R.L. Cilibrasi, and M. Li, "Normalized information distance," in *Information theory and statistical learning.* Springer, 2009, pp. 45–82.

[80] B.Y. Pratama and R. Sarno, "Personality classification based on Twitter text using Naive Bayes, KNN and SVM," in *Proceedings of the International Conference on Data and Software Engineering*, 2015, pp. 170–174.

[81] J.C. Corrales, *Behavioral matchmaking for service retrieval*, Ph.D. dissertation, Université de Versailles-Saint Quentin en Yvelines, 2008.

[82] S.S. Choi, S.H. Cha, and C.C. Tappert, "A survey of binary similarity and distance measures," *Journal of Systemics, Cybernetics and Informatics*, Vol. 8, No. 1, 2010, pp. 43–48.

[83] "Parameterize BDD tests," 2021. [Online]. https://support.smartbear.com/testcomplete/docs/bdd/parameterize.html

[84] G. Gui and P.D. Scott, "Coupling and cohesion measures for evaluation of component reusability," in *Proceedings of the 2006 International Workshop on Mining Software Repositories*, 2006, pp. 18–21.

[85] S. Park and F. Maurer, "Communicating domain knowledge in executable acceptance test driven

development," in *International Conference on Agile Processes and Extreme Programming in Software Engineering*. Springer, 2009, pp. 23–32.

[86] A.A. Kiani, Y. Hafeez, M. Imran, and S. Ali, "A dynamic variability management approach working with agile product line engineering practices for reusing features," *The Journal of Supercomputing*, 2021, pp. 1–42.

[87] I.F. Da Silva, "An agile approach for software product lines scoping," in *Proceedings of the 16th International Software Product Line Conference Volume 2*, 2012, pp. 225–228.

[88] P. Runeson, M. Host, A. Rainer, and B. Regnell, *Case study research in software engineering: Guidelines and examples*. John Wiley and Sons, 2012.

e-Informatica Software Engineering Journal, Volume 15, Issue 1, 2021, pages: 163–184, DOI 10.37190/e-Inf210108

Multi-view learning for software defect prediction

Elife Ozturk Kiyak[*], Derya Birant[**], Kokten Ulas Birant[**]
[*] *Graduate School of Natural and Applied Sciences, Dokuz Eylul University, Izmir, Turkey*
[**] *Department of Computer Engineering, Dokuz Eylul University, Izmir, Turkey*
elife.ozturk@deu.edu.tr, derya.birant@deu.edu.tr, ulas.birant@deu.edu.tr

Abstract

Background: Traditionally, machine learning algorithms have been simply applied for software defect prediction by considering single-view data, meaning the input data contains a single feature vector. Nevertheless, different software engineering data sources may include multiple and partially independent information, which makes the standard single-view approaches ineffective.
Objective: In order to overcome the single-view limitation in the current studies, this article proposes the usage of a multi-view learning method for software defect classification problems.
Method: The Multi-View *k*-Nearest Neighbors (MVKNN) method was used in the software engineering field. In this method, first, base classifiers are constructed to learn from each view, and then classifiers are combined to create a robust multi-view model.
Results: In the experimental studies, our algorithm (MVKNN) is compared with the standard *k*-nearest neighbors (KNN) algorithm on 50 datasets obtained from different software bug repositories. The experimental results demonstrate that the MVKNN method outperformed KNN on most of the datasets in terms of accuracy. The average accuracy values of MVKNN are 86.59%, 88.09%, and 83.10% for the NASA MDP, Softlab, and OSSP datasets, respectively.
Conclusion: The results show that using multiple views (MVKNN) can usually improve classification accuracy compared to a single-view strategy (KNN) for software defect prediction.

Keywords: Software defect prediction, multi-view learning, machine learning, *k*-nearest neighbors

1. Introduction

Predicting defects (bugs) in source codes is one of the most valuable processes of the software development life cycle (SDLC) to achieve high quality and reliability in software. The defects that exist in the software lead to not only waste time and money but also severe consequences during deployment. Therefore, detecting and fixing defects in the initial stages of SDLC are crucial requirements to produce robust and effective software systems. For this purpose, so far, many studies on software defect prediction have been conducted by utilizing machine learning techniques. For instance, software modules have been classified as buggy or non-buggy, which refers to the binary classification, or the number of bugs predicted, which refers to the regression problem [1].

Software defect prediction (SDP) generally includes the following stages to recognize defect-prone software components. (i) First, software modules are obtained from software repositories. (ii) After that, the features (software metrics) are extracted from the software modules and each module is labeled to indicate whether the module contains a defect or not. (iii) A classification model is constructed on the training labeled data. (iv) Finally, the constructed model is utilized to estimate the defect-proneness of new unseen software modules.

The standard SDP studies have used the traditional machine learning techniques which are basically working on single-view data. However, the SDP problems can involve data with multiple views (i.e., multiple feature vectors). The conventional classification algorithms (i.e., *k*-nearest

Submitted: 22 March 2021; Revised: 10 August 2021; Accepted: 10 August 2021; Available online: 3 November 2021

neighbors – KNN) simply concatenate all multiple views into a single view for learning. However, this simple view-concatenation approach may produce undesirable prediction results since each view has its own specific characteristics. Therefore, multi-view learning (MVL) methods are needed to individually explore diverse information from several different feature vectors and, hence, to increase learning performance by taking into account the diversity of various views. For this purpose, in this work, we propose the usage of a multi-view learning approach for software defect prediction problems.

The fundamental contributions of this paper can be pointed out as follows. (i) This study uses the *Multi-View k-Nearest Neighbors (MVKNN)* algorithm in the software engineering area. (ii) It compares the MVKNN and KNN algorithms for software defect prediction. (iii) This study is also original in that it investigates the effects of the number of neighbors (the parameter k) on the software defect prediction performance.

In the experimental studies, we demonstrated the effectiveness of the multi-view learning approach on the SDP. Our MVKNN method was tested on 50 datasets obtained from different software bug repositories. The experiment results show that our MVKNN algorithm achieved higher accuracy values than the KNN algorithm on most of the datasets.

The remainder of this article is basically organized in the following manner. In Section 2, we give an overview of the previous studies related to the topic. In Section 3, we explain the methods used in this study. In Section 4, we describe the main characteristics of the datasets, present the experimental studies, and discuss the results. The last part (Section 5) provides concluding remarks and intended future works.

2. Related work

In *single-view learning*, a classification method is applied to the entire feature set or the specific part of the feature set [2]. In *multi-view learning*, various distinct feature sets (views) are evaluated [3]. These views can be collected from diverse sources, or a single raw data can be separated into different feature sets. For example, in web mining, the textual content can be considered as one view, and image data can be represented as another view. Similarly, a single document can have multiple representations (views) in different languages. Another typical example is the music emotion recognition, in which lyrics (view 1) and song (view 2) can be considered as multiple views. The views can be multiple measurement modalities such as jointly represented audio signals + video signals in television broadcast, biological + chemical data in drug discovery, or images from different angles.

Recently, multi-view learning (MVL) has been combined with different machine learning paradigms such as active learning [4], ensemble learning [5, 6], multi-task learning [7], and transfer learning [8]. In the literature, many studies on MVL have been focused on classification and/or regression problems [4, 5, 8]; however, recently, some studies have focused on a clustering task [3, 6]. Until now, MVL has been used in different fields such as health [5, 9], finance [6], and transportation [7]. However, MVL studies are considerably limited in software engineering, especially for software defect prediction.

2.1. Related studies on single-view learning for software engineering

Researchers have concentrated on machine learning (ML) methods in software defect prediction. However, they have applied ML techniques on the whole feature set or the selected feature subset. In the literature, a massive number of existing SDP studies have built classification models on specific features which have been determined by using a feature selection method. For example, Laradji et al. [10] used an ensemble classifier and claimed that the reduction of unimportant and irrelevant features improves the performance of defect prediction. They applied a greedy-forward selection method to obtain a subset of software metrics. Agarwal and Tomar [11] applied the F-score feature selection method which was utilized to determine the important software metric set that was distinctly affecting the defect classification in software projects. The study conducted by Wang et al. [12] used the filter-based feature ranking techniques to determine how many metrics should be selected when constructing a defect prediction model. First, the top 10, 15, and 20 metrics were chosen according to their scores, and then three different classification models were constructed. Although different feature sets achieved high accuracy for different classifiers, their results showed that a predictor could be constructed with only three features.

In the literature, ML methods have been usually used to perform *within-project defect prediction (WPDP)*. However, when historical data was insufficient within a project, the *cross-project defect prediction (CPDP)* approach [13] has been applied to employ the information from other projects. Moreover, some studies used advanced ML paradigms in software defect prediction, such as ensemble learning [14], semi-supervised learning [15], and transfer learning [13].

All the aforementioned studies performed experiments on a single feature vector (view). However, in our study, we used several feature sets (views) to learn diverse information obtained from various data sources, and therefore to increase learning performance by taking into account the diversity of different views.

2.2. Related studies on multi-view learning for software engineering

Recently, lots of information about a subject have been acquired easily, as well as various kinds of data (i.e., image, audio, text, video) have been obtained from multiple sources. As many MVL studies [3] indicated that the information acquired by using data gathered from multiple sources can be more valuable than the information obtained from single-view data. Thus, multi-view learning has been used in a variety of areas for different purposes, such as text classification [16], image classification [17], face identity recognition [18], and speech recognition [19].

Multi-view learning in software engineering is concerned with the problem of learning from data that describe a particular SE problem and is represented by multiple distinct feature sets (called views). Many SE problems can be expressed with different data views. One of the most well-known examples is that software engineering data can consist of several views such as the module dependency graph (view 1), execution logs (view2), and the vocabulary (view 3) used in the source code. Since all of them collectively describe a software system, the integration of all these unique and complementary views can be jointly used for analysis [20]. A software system can also be investigated from different perspectives, relating to a variable group or representation scheme depicting that domain. For instance, evolutionary information is a set of group variables describing the co-changing frequency of software units. Another example in the software engineering area is the identification of programming language from different views. For example, source code classification can be performed by applying multi-view learning to the same piece of code data obtained as text (view1) and image (view2) [21]. In the software defect prediction area, as in this study, the software metrics extracted from source codes can be divided into different views, considering that they are obtained from different perspectives.

In the literature, only a limited number of studies have focused on multi-view learning for software defect classification. In 2017, Phan and Nguyen [22] applied a multi-view convolutional neural network to predict software defects from assembly instruction sequences. They represented each instruction with two views: the content (view 1) and the group (view 2). For each view, convolutional layers were applied and merged before feeding them to the fully-connected layers. In 2019, Chen et al. [23] proposed a multi-view NN-based heterogeneous transfer learning model built by partitioning features into groups for software defect prediction. However, in the former study, source code was used as a dataset, and in the latter study, different feature sets were evaluated. Our approach is leveraged by combining KNN algorithms separately implemented to each view.

2.3. Related studies on KNN for software defect prediction

By means of its simplicity and ease of implementation, the KNN algorithm has been implemented in many ML applications. For defect prediction, several studies have been developed using the KNN algorithm in various ways, such as weighted KNN [24], hybrid model using Naive Bayes and KNN [25], boosting-based KNN [26], and KNN regression [27]. Although these studies are related to defect prediction, experiments were conducted using single view data. Different from these previous studies, we used an improved version of KNN for multi-view software defect data.

Until now, the KNN algorithm has been employed in various multiple learning studies in different areas. For example, multi-label learning (ML-KNN) [28], multi-instance learning (FuzzyKNN) [29], multi-class learning (DEMST--KNN) [30], and multi-task learning (ssMTTL--KNN) [31]. Unlike these previous studies, we used the multi-view KNN method proposed in [32].

3. Material and methods

In this section, the KNN and MVKNN algorithms are briefly described.

3.1. *K*-nearest neighbors

K-nearest neighbors (KNN) is a typical supervised ML algorithm utilized in both classification and regression problems. The KNN algorithm works on labeled data samples and uses them to classify a new sample based on its similarity to the k closest neighbors. In this study, we chose KNN as a machine learning algorithm since it has many advantages such as simplicity, easy implementation, efficiency, and ease of understanding the results [33]. The advantages of KNN also include that it supports incremental data; therefore, re-training is not required for the newly-arrived observations. The other advantage of KNN is that it can be successfully used for nonlinear data. Moreover, it has the ability to predict both categorical and discrete variables and to deal with noisy data. Furthermore, it can be directly used for multi-class classification, in addition to binary classification. KNN has been proven to be an effective method in various studies [24–31] and thus, it has been widely used in many applications.

Considering k as the number of nearest neighbors and n labeled data samples, $D = \{s_1, s_2, \ldots, s_n\}$ be the training set in the form of $s_i = (x_i, c_i)$, where x_i is the feature vector of the sample with d-dimension, denoted by $x_i = (x_{i1}, x_{i2}, \ldots, x_{id})$ and comes from data space X, and c_i is the class label that s_i assigned to and it is from a set of classes $Y = \{c_1, c_2, \ldots, c_m\}$. A classifier is a function in the form of $f : X \rightarrow Y$ that maps a new data sample X onto an item of Y. The KNN algorithm starts with a new sample $s' = (x', ?)$ whose class label is unknown. Distance $\text{dis}(x', x_i)$ is calculated between s' and all s_i in the dataset D. The most widely-used distance measures are Euclidean and Manhattan distance metrics. Then, the k closest neighbors to s' in the training samples are selected. Finally, class label c' is assigned to s' based on the majority class of neighbors.

3.2. Multi-view k-nearest neighbors

Our algorithm, called *multi-view k-nearest neighbors* (MVKNN), was proposed in [32]. It is an advanced version of KNN that combines individual models developed for each view.

Let $S^v = (X^v, c)$ be a sample of a view v, where X^v is the feature set of v such that $X^v = \{x_1^v, x_2^v, \ldots, x_d^v\}$ for $v = 1, 2, \ldots, V$ and c refers to the class label of the sample, where V is the number of views. The dataset D consists of n samples and denoted by $D = \{(X_i^1, X_i^2, \ldots, X_i^V, c_i), i = 1, 2, \ldots, n\}$, where X_i is the i^{th} sample and $c_i \,\epsilon\, Y$ is the class label of it and X_i^j refers to the i^{th} instance of the j^{th} view. Views are mutually exclusive, so they have different feature sets such that $\forall_{(p,q)} X^p \cap X^q = \emptyset$. Since views express different representations of the same object, each instance in different views has the same class label $c_i \,\epsilon\, Y$, where $Y = \{c_1, c_2, \ldots, c_n\}$. Firstly, in the *view-based classification*, a set of classifiers are built for each view by using different k input parameters such that $f_j : X^j \rightarrow Y$. After that, in the *multi-view-based classification*, a number of view-based classifiers f_j are combined to determine the final prediction.

Figure 1 shows the general overview of the MVKNN method that can be used for software defect classification. The MVKNN method consists of three main steps: data preparation, view-based classification, and multi-view-based classification.

Step 1 – *Data preparation*: Raw data obtained from a software repository may require preparation before yielding results, so it has been prepared for view-based classification. To increase data quality and to acquire more accurate outcomes, various preprocessing techniques may be applied such as completing missing data or removing duplicate instances. Since the dataset consists of several views, data preprocessing should be performed for all views. It can be noted here that each view has a different feature set. In other words, each view is represented as disjoint features of the same object, so distinct features and class label of an object exist in each view.

Step 2 – *View-based classification*: This classification aims at creating a model learning from each view using an adaptive method in which each instance is classified with a different number of neighbors, rather than utilizing only a single value of k. For each view, weak KNN classifiers

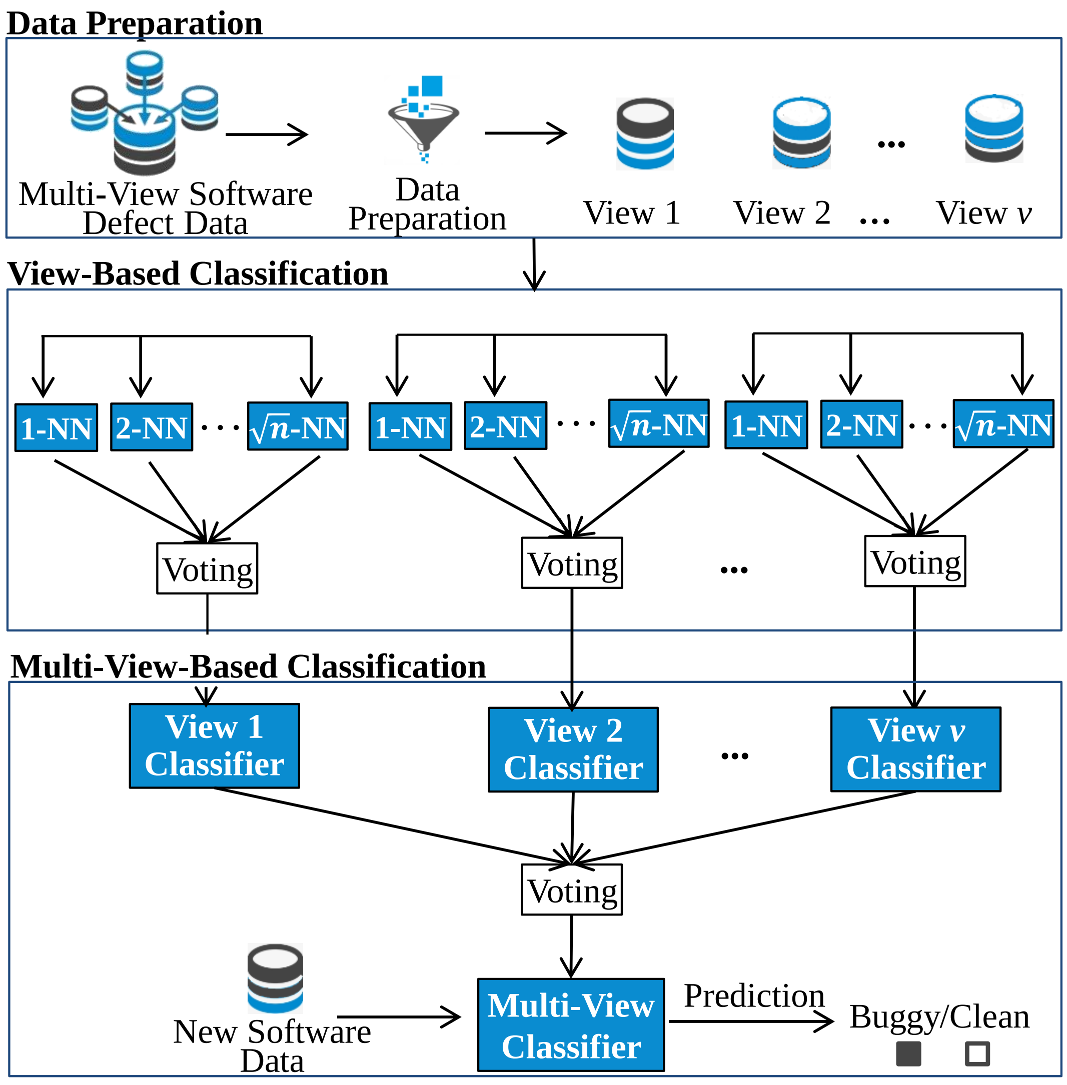


Figure 1: The general overview of the MVKNN method for software defect prediction

are constructed with different parameters (k), where k is ranged from 1 to the square root of the number of samples ($\sqrt{n}$). The classifiers (1-NN, 2-NN, 3-NN, ..., $\sqrt{n}$-NN) are combined to form a strong ensemble model for a specific view. In the end, the class label of a particular view is determined by using a voting strategy that selects the class having the highest number of votes, where n refers to the number of samples in the dataset.

Step 3 – *Multi-view-based classification*: In this step, a number of view-based learners are combined to form a general model for classification. A majority voting strategy is utilized to combine outcomes from each view and the final class label of a new instance is specified as *buggy* or *clean*.

The MVKNN method can be used for software detect prediction to provide many advantages as follows:

– A single-view software data is dependent on a single view-point, whereas multi-view software data usually contains complementary information since it typically includes many view-points. In multi-view learning, the lack of information of one view can be complemented by the sufficiency of other views. Thanks to this essential property of MVKNN, it eliminates the weaknesses of single-view software defect prediction.
– Compared to the traditional KNN algorithm which is substantially developed for single-view data, MVKNN is expected to yield more robust outcomes in the presence of noisy software defect data. Because noise in one view can be reduced by a voting mechanism among multiple views.
– Numerous software metrics can be extracted from software projects and so data can be high dimensional. The MVKNN method has the ability to handle a large number of features since it considers high-dimensional data as a union of multiple low-dimensional subspaces, called views. Software defect data can contain many metrics from different perspectives such as Halstead's measures and McCabe's measures. Therefore, considering high dimensional data containing a group of different feature sets, we can separate the data into appropriate views, each corresponding to a disjoint feature set.

In spite of numerous benefits, the MKKNN method considers correlations at the view level; however, it does not take into account implicit correlations between features in multiple views. In addition, MVKNN is computationally more expensive than KNN since it separately learns from each view dataset and runs the base learner many times to jointly learn from multiple k parameters rather than a single k parameter.

4. Experimental studies

In this section, software defect datasets are described, and several experiments conducted with the MVKNN method are presented. The MVKNN and KNN methods were compared for software defect prediction. The obtained results were validated utilizing statistical tests to ensure that the differences between the methods in the datasets were significant. For this purpose, we applied the Wilcoxon test which is a well-known non-parametric statistical test.

The MVKNN algorithm was implemented utilizing the WEKA machine learning library [34] and *C#* programming language. The implementation of the MVKNN method is available at the website https://github.com/elifeoztu rk/MVKNN. As an evaluation method, the 10-fold cross-validation technique was used, in which the dataset is divided into 10 parts and then each part is used once as a test set while the remaining parts form the training set. In this study, four evaluation metrics (Accuracy, Precision, Recall, and *F1 Score*) were used to evaluate the classification performances. *Accuracy* is the most widely-used performance measure that calculates the ratio between the number of correctly predicted instances and all instances. It is calculated as follows: $Accuracy = (TN + TP)/(TP + FP + TN + FN)$, where TN (true negatives) and TP (true positives) are correctly predicted as real labels. In other words, if the real label is "positive" and the predicted label is "positive" or the actual label is "negative" and the predicted label is "negative", it is TP or TN, respectively. Unlike correct estimates, FP and FN point out that instances did not correctly classified as actual labels. *Precision* shows the ratio of correctly predicted positive instances to the total predicted positive instances. Precision is calculated as follows: *Precision* $= TP/(TP + FP)$. *Recall* is the number of correct predictions divided by the number of all predictions in the actual class. It is calculated as follows: *Recall* $= TP/(TP + FN)$. *F1 Score* is the harmonic mean of the precision and recall. This evaluation measure is particularly preferred when datasets have uneven class distribution. *F1 Score* is calculated as follows: $F1$ *Score* $= 2 \times$ (*Recall* $\times$ *Precision*)/(*Recall* + *Precision*). *Precision*, *Recall*, and *F1 Score* are useful metrics for evaluating "learning from imbalanced datasets".

4.1. Dataset description

In this work, we conducted experiments on 50 bug datasets from three different repositories available in the software engineering area: Tera-PROMISE Open Source Software Projects (OSSP), NASA MDP (Metrics Data Program), and Softlab that included 40, 5, and 5 datasets, respectively. Table A2 lists the main characteristics of the datasets, including their names, the groups to which the datasets belong, the number of samples in the datasets, and defect percentages (%). To be able to test MVKNN on imbalanced data, we especially used the NASA MDP datasets where defect percentages ranged between 7% and 23%. Furthermore, these datasets have been widely used in many machine learning studies [10, 14, 15]. MVKNN is designed for general purposes; therefore, it can be further applied to different datasets with different software engineering metrics when the research community has presented new ones.

More details about datasets are described below, and supplemental information and tables are included in Appendix A.

- *OSSP Datasets* [35]: The datasets in this group consists of 20 independent object-oriented source code metrics and one dependent defect variable that indicates buggy or not.
- *NASA MDP Datasets* [36]: The datasets (named cm1, jm1, kc1, kc2, pc1) in this group were obtained from NASA software projects. These datasets include 21 static code features that were extracted from a software product based on the McCabe metric, Basic Halstead measures, and Derived Halstead measures.
- *Softlab Datasets* [37]: The datasets were denoted by a Software Research Laboratory and collected from a Turkish white-goods manufacturer. The ar1 and ar6 datasets were collected from an embedded controller for white goods, while the other (ar5, ar4, ar3) datasets were obtained from a refrigerator, dishwasher, and washing machine, respectively. The datasets contain 29 static code attributes.

Table A3 presents the fundamental characteristics of datasets, containing the number of classes (i.e., buggy or clean), the number of views, and the number of features that belong to each view.

Tables A4, A5 and A6 show all the software metrics and their categories in each dataset group for NASA, OSSP and SOFTLAB datasets, respectively. The datasets have different views designed based on the previous studies [38–40]. The NASA MDP datasets have three views as given in [38]: McCabe, Basic Halstead, and Derived Halstead features with 4, 9, and 8 software metrics, respectively. The OSSP datasets contain object-oriented measures that indicate the characteristics of inheritance, coupling, cohesion, complexity, and line features of software programs [39]. The Softlab datasets consist of four views [40], including Halstead, McCabe, LOC, and Miscellaneous metrics. Halstead metrics show the program complexity obtained by analyzing the source code. McCabe metrics quantify the control flows in a program. LOC metrics refer to the measures related to lines of code, such as the number of executable lines and the number of comment lines. Finally, the rest software metrics are additionally included in the "Miscellaneous" group.

4.2. Experimental results

Table 1 shows the comparison of the KNN and MVKNN algorithms on the NASA MDP and Softlab datasets in terms of accuracy. According to the results, our MVKNN algorithm achieved 86.59% and 88.09% accuracy values on average for the NASA MDP and Softlab datasets, respectively. However, KNN reached only 86.46% and 86.60% accuracy values on average. It is clear that MVKNN outperformed KNN on four datasets (jm1, kc1, kc2, pc1) from the NASA MDP group, while only on one dataset (cm1) both algorithms have the same classification accuracy (90.16%). The results obtained from the Softlab datasets show that our MVKNN algorithm demonstrated better or equal accuracy on all datasets compared to the KNN algorithm.

According to the results, it is possible to say that quality assurance teams can effectively allocate limited resources for validating software products since the constructed defect prediction models provide satisfactory results (>86%) on average when identifying bug-prone software artifacts. This result indicates that the models can correctly predict defect-prone software modules with a rate of 86% before defects are discovered; thus, the predictive models can be used to prioritize software quality assurance efforts. The code areas that potentially contain defects can be predicted to help developers allocate their testing efforts by first checking potentially buggy code. As the size of software projects becomes larger, the constructed defect prediction models play a more critical role to support developers. Furthermore, they speed up time to market as well as with more robust software products.

Figure 2 shows the comparison of KNN and MVKNN on the OSSP datasets in terms of accuracy. The results show that our MVKNN algorithm has equal to or higher accuracy than the KNN algorithm on 34 out of 40 datasets. Therefore, our MVKNN algorithm is better than KNN on 85% of the datasets. In particular, the biggest accuracy differences between the methods were observed on the "berek" and "velocity 1.6" datasets, where our method increased the accuracy by over 6.98% and 6.37%, respectively. For example, our method (86.05%) achieved better performance than the existing method (79.07%) in the "berek" dataset in terms of accuracy. In the "velocity 1.6" dataset, accuracy of our method (72.05%) is higher than the accuracy of the existing method (65.68%). These results show that our method usually gives more accurate outputs for software defect prediction by using a different perspective.

Some datasets were obtained from different versions of a software project, such as Jedit 4.0, Jedit 4.1, Jedit 4.2, and Jedit 4.3. In this case, within-project defect prediction (WPDP) can be used to identify defect-prone modules in a forthcoming version of a software project. However, some datasets were obtained from a single version of a software project, such as the "arc" dataset. In this case, cross-project defect prediction (CPDP) can be applied since the target project may be a new project or does not have enough labeled modules.

In addition to accuracy, we evaluated the performances of the methods using the *Precision*, *Recall*, and *F1 Score* metrics. As can be seen in Table B1, when considering the NASA MDP datasets, MVKNN achieved equal or higher accuracy than KNN. In the Softlab datasets, it is clearly seen that the MVKNN model has better results than the KNN model on average. In addition, Table B2 shows the *Precision*, *Recall*, and *F1 Score* results for the OSSP datasets. According to the results, the MVKNN algorithm achieved the values of 0.79, 0.83, and 0.81 for the precision, recall, and *F1 Score* metrics, respectively; whereas, KNN only obtained the values of 0.77, 0.81, and 0.79 on average.

Though MVKNN achieved usually higher accuracy than KNN, all the results (Table 1 and Figure 2) were also validated using statistical tests. We utilized a well-known non-parametric statistical test: Wilcoxon Test, also known as Wilcoxon signed-rank test. Since it is used to analyze matched-pair data, it can be considered a rank-based alternative to the two-sample t-test [41]. Wilcoxon test does not rely on the assumption of data complying with any distribution. It considers the sign and magnitude of the distribution of cumulative observations. In

Table 1: Comparison of the KNN and MVKNN algorithms on the NASA MDP and Softlab datasets

NASA MDP			Softlab		
Dataset	KNN	MVKNN	Dataset	KNN	MVKNN
cm1	**90.16**	**90.16**	ar1	**92.56**	**92.56**
jm1	80.98	**81.08**	ar3	90.08	**90.48**
kc1	84.97	**85.30**	ar4	83.88	**85.98**
kc2	83.14	**83.33**	ar5	79.86	**83.33**
pc1	93.03	**93.06**	ar6	**85.15**	**85.15**
Avg.	86.46	**86.59**	**Avg.**	86.60	**88.09**

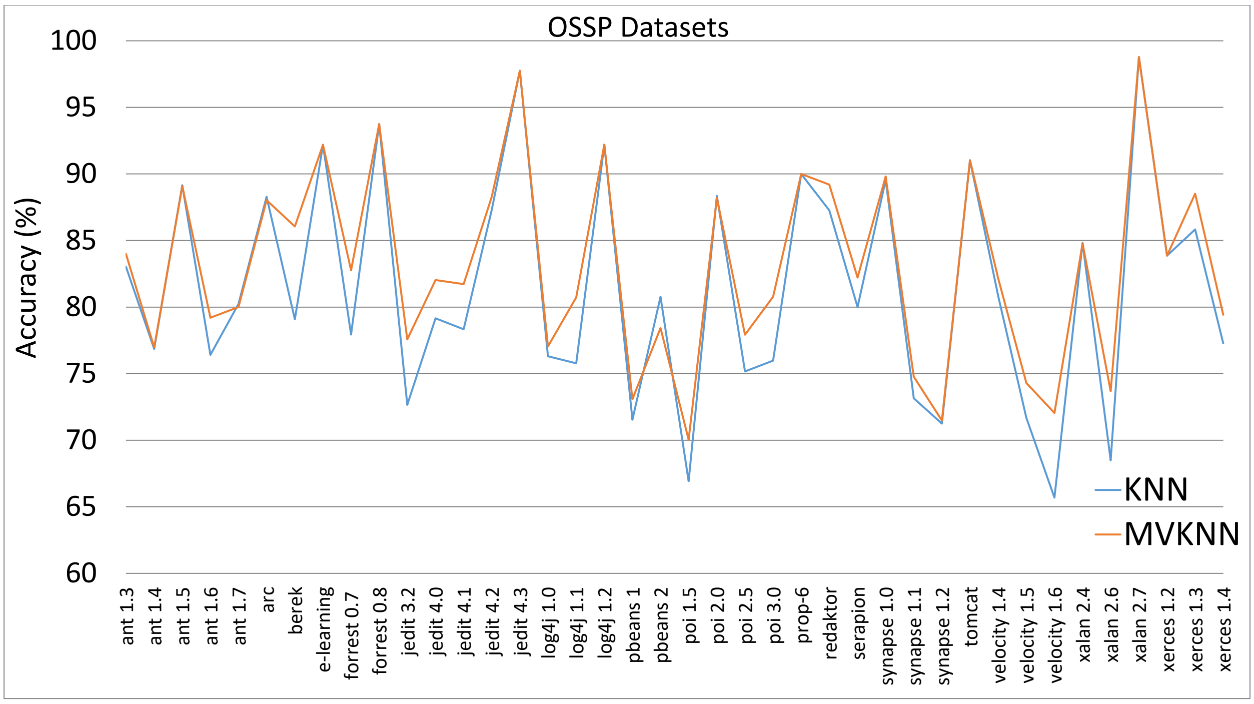


Figure 2: Comparison of the KNN and MVKNN algorithms on the OSSP datasets

this statistical test, the null hypothesis (H0) indicates that there is no difference or relationship between methods. The alternative hypothesis (H1) states that there is a significant difference or relationship between methods. A significance level (α) is usually specified as 0.05. The p-value is used to determine the presence of statistical significance since it shows the level of evidence of the difference. If the obtained p-value is lower than the threshold level ($p < 0.05$), the null hypothesis (H0) is rejected, which means that the difference is significant. Since the p-value obtained from the Wilcoxon test is 0.0000027 and it is smaller than the significance level, H0 is rejected. Therefore, it implies that the obtained results are statistically significant. As can be seen in Figure 3, MVKNN showed the median accuracy of 83.6%, while KNN had the median accuracy of 82.0%.

In order to show performance comparisons between KNN and MVKNN, supplemental tables and figures are presented in Appendix B. Figure B1 displays individual view-based performances. When each view in the datasets is examined separately, it can be seen that MVKNN has good performance on the view-based classification, but it does not always better than KNN. However, the multi-view-based classification results of MVKNN are either greater than or equal to the KNN accuracy values for all datasets.

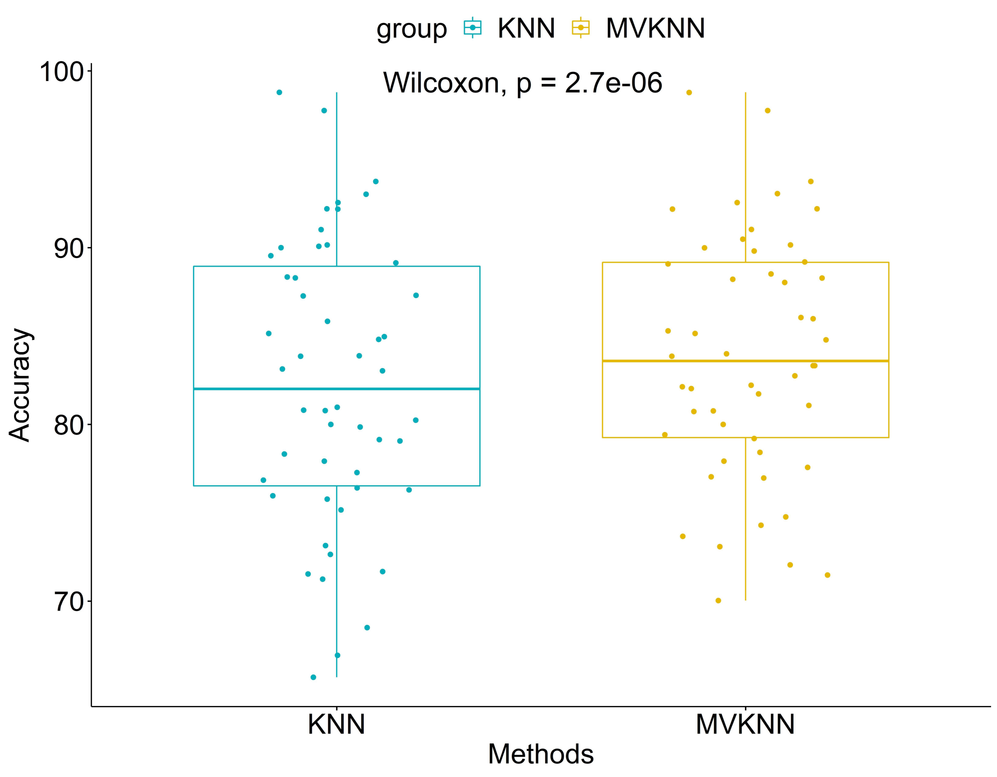


Figure 3: The spread of the accuracy scores for each algorithm

This is because of the complementary power of MVKNN.

In the traditional KNN algorithm, a single and fixed k value is used for classification. However, in a real dataset, data points may be distributed irregularly, or some of them can be noisy. To consider the density variations in the data, different k parameters can be used to benefit from both more neighbors in dense regions and less in sparse regions. Liu et al. pointed out that a fixed k value is not suitable for many test samples in a given training set [42]. For this reason, in our study, rather than just using a single and fixed value of k, the algorithm is run with various k values. MVKNN learns from k values, starting from 1 to $\sqrt{n}$, where n is the number of instances. The maximum k value was selected as $\sqrt{n}$ on the basis of many studies [43–45]. Choosing the maximum value of k as $\sqrt{n}$ is an appropriate decision since the probability of overfitting significantly increases if k is selected as too small or too large. Park and Lee [44] reported that setting k as the square root of the data size is a good empirical ground rule. Lall and Sharma [43] also proved this statement theoretically using a generalized cross-validation (GCV) score function. Mitra et al. [45] also stated that $k = \sqrt{n}$ is usually suitable for test samples. If the k value is small, then the results can be susceptible to noisy instances; otherwise, if the k value is large, the neighborhood may cover many instances from other classes. Therefore, the square root is a reasonable choice for the searching neighborhood. Figure B2 displays accuracy values obtained for different k values. It compares KNN and MVKNN performance on the k values starting from 1 to the square root of the number of samples for each dataset. For all the datasets, MVKNN starts with greater accuracy than KNN, and at the maximum k value, it is either greater than or equal to the KNN accuracy.

Table B3 separately shows the performances of the KNN and MVKNN algorithms for each view in the OSSP datasets. It is clearly observed that MVKNN has equal to or higher accuracy than KNN for 34 out of 40 datasets. It can be seen that our MVKNN method (83.10%) outperformed the traditional KNN method (81.37%)

in classification on average. According to the results given in Table B3, the proposed method (MVKNN) is more successful than the traditional KNN method in terms of accuracy. Thus, multi-view learning can achieve more accurate results than single-view learning in defect prediction since it benefits from different perspectives of data.

4.3. Validity

This section discusses the validity of the research, the threats, and countermeasures of the context under common guidelines given in [46].

i. **Construct validity**
Threats to construct validity are concerned with establishing correct measures for the concepts addressed in empirical analysis. The selection of performance measures is the basic limitation. In our study, the most commonly used performance measure (accuracy) was selected to overcome the threat of measure selection. In other words, the predictive validities of the constructed models were assessed by using the accuracy measure. According to the results, the proposed approach achieved 86.59%, 88.09%, and 83.10% accuracy values on average for the NASA MDP, Softlab, and OSSP datasets, respectively. Thereby, predictive validity was proven, since all average accuracy results are higher than the acceptable level (>80%).

ii. **Internal validity**
Internal validity is related to uncontrolled factors that can cause a difference in experimental measurements. To reduce this threat, we used the k-fold cross-validation technique in which the validation procedure is repeated k times until each of the k data subsets has served as a test set. In addition, we ran all the experiments in the same environment. Another internal threat is related to data collection. We tested our approach on public and most widely-used software engineering datasets in the literature [11, 13–15]. The information on data collection and its validity can be found in [35–37].

iii. **External validity**
External validity concerns the generalizability of a conclusion or experimental finding reached on the sample group under experimental conditions in various environments. In this study, our approach was tested on a total of 50 datasets from three different data repositories to reduce the threat of this kind of validity. In addition, since MVKNN is designed for general-purpose, it can be applied to various domains from transportation to medicine.

iv. **Conclusion validity**
Conclusion validity is concerned with the relationship between the treatment in an experiment and the actual outcome we observed. It is considered as the evaluation of statistical power, significance testing, and effect size. For this purpose, we used the Wilcoxon statistical test to ensure the differences in KNN and MVKNN performances are statistically significant. Since the p-value obtained from the statistical test (0.0000027) is smaller than the significance level (<0.05), it is possible to say that the results are statistically significant.

5. Conclusion and future work

The standard software defect classification studies work on single-view data. They do not utilize different feature sets, called views. However, a software defect prediction problem can involve data with multiple views in which the feature space includes multiple feature vectors. Therefore, in this study, the multi-view k-nearest neighbors (MVKNN) algorithm is used for software defect classification. Here, the software defect metrics are grouped under several views according to their feature extractors. MVKNN consists of two parts. First, base classifiers are constructed to learn from each view. Second, classifiers are combined to create a strong multi-view model.

In this study, several experiments were conducted on 50 bug datasets from different repositories to show the capability of the MVKNN method. It can be concluded from the results that the MVKNN algorithm usually achieved better performance compared to the KNN algorithm.

As future work, the MVKNN method can be used for other software engineering problems such as software cost estimation, software effort prediction, readability analysis, refactoring, software clone detection, vulnerability prediction, and software design pattern mining.

References

[1] R. Ozakinci and A. Tarhan, "Early software defect prediction: A systematic map and review," *The Journal of Systems and Software*, Vol. 144, Oct. 2018, pp. 216–239.

[2] K. Bashir, T. Li, and M. Yahaya, "A novel feature selection method based on maximum likelihood logistic regression for imbalanced learning in software defect prediction," *The International Arab Journal of Information Technology*, Vol. 17, No. 5, Sep. 2020, pp. 721–730.

[3] J. Zhao, X. Xie, X. Xu, and S. Sun, "Multi-view learning overview: Recent progress and new challenges," *Information Fusion*, Vol. 38, No. 1, Nov. 2017, pp. 43–54.

[4] F. Liu, T. Zhang, C. Zheng, Y. Cheng, X. Liu, M. Qi, J. Kong, and J. Wang, "An intelligent multi-view active learning method based on a double-branch network," *Entropy*, Vol. 22, No. 8, Aug. 2020.

[5] Y. Chen, D. Li, X. Zhang, J. Jin, and Y. Shen, "Computer aided diagnosis of thyroid nodules based on the devised small-datasets multi-view ensemble learning," *Medical Image Analysis*, Vol. 67, No. 8, Jan. 2021.

[6] Y. Song, Y. Wang, X. Ye, D. Wang, Y. Yin, and Y. Wang, "Multi-view ensemble learning based on distance-to-model and adaptive clustering for imbalanced credit risk assessment in p2p lending," *Information Sciences*, Vol. 525, Jul. 2020, pp. 182–204.

[7] S. Cheng, F. Lu, P. Peng, and S. Wu, "Multi-task and multi-view learning based on particle swarm optimization for short-term traffic forecasting," *Knowledge-Based Systems*, Vol. 180, Sep. 2019, pp. 116–132.

[8] Y. He, Y. Tian, and D. Liu, "Multi-view transfer learning with privileged learning framework," *Neurocomputing*, Vol. 335, Mar. 2019, pp. 131–142.

[9] J. Li, L. Wu, G. Wen, and Z. Li, "Exclusive feature selection and multi-view learning for alzheimer's disease," *Journal of Visual Communication and Image Representation*, Vol. 64, Oct. 2019.

[10] I.H. Laradji, M. Alshayeb, and L. Ghouti, "Software defect prediction using ensemble learning on selected features," *Information and Software Technology*, Vol. 58, Feb. 2015, pp. 388–402.

[11] S. Agarwal and D. Tomar, "A feature selection based model for software defect prediction," *International Journal of Advanced Science and Technology*, Vol. 65, 2014, pp. 39–58.

[12] H. Wang, T.M. Khoshgoftaar, and N. Seliya, "How many software metrics should be selected for defect prediction?" in *Proceedings of the Twenty-Fourth International Florida Artificial Intelligence Research Society Conference*, R.C. Murray and P.M. McCarthy, Eds. Palm Beach, Florida, USA: AAAI Press, May 2011.

[13] W. Wen, B. Zhang, X. Gu, and X. Ju, "An empirical study on combining source selection and transfer learning for cross-project defect prediction," in *2019 IEEE 1st International Workshop on Intelligent Bug Fixing (IBF)*. Hangzhou, China: IEEE, 2019, pp. 29–38.

[14] A. Iqbal, S. Aftab, I. Ullah, M.S. Bashir, and M.A. Saeed, "A feature selection based ensemble classification framework for software defect prediction," *International Journal of Modern Education and Computer Science*, Vol. 11, No. 9, 2019, pp. 54–64.

[15] A. Arshad, S. Riaz, L. Jiao, and A. Murthy, "The empirical study of semi-supervised deep fuzzy c-mean clustering for software fault prediction," *IEEE Access*, Vol. 6, 2018, pp. 47 047–47 061.

[16] M.M. Mirończuk, J. Protasiewicz, and W. Pedrycz, "Empirical evaluation of feature projection algorithms for multi-view text classification," *Expert Systems with Applications*, Vol. 130, 2019, pp. 97–112.

[17] C. Zhang, J. Cheng, and Q. Tian, "Multi-view image classification with visual, semantic and view consistency," *IEEE Transactions on Image Processing*, Vol. 29, 2020, pp. 617–627.

[18] Z. Zhu, P. Luo, X. Wang, and X. Tang, "Multi--view perceptron: a deep model for learning face identity and view representations," in *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014*, Z. Ghahramani, M. Welling, C. Cortes, N.D. Lawrence, and K.Q. Weinberger, Eds. Montreal, Quebec, Canada: Citeseer, Dec. 2014, pp. 217–225.

[19] S.R. Shahamiri and S.S.B. Salim, "A multi-views multi-learners approach towards dysarthric speech recognition using multi-nets artificial neu-

ral networks," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, Vol. 22, No. 5, Sep. 2014, pp. 1053–1063.

[20] A. Saeidi, J. Hage, R. Khadka, and S. Jansen, "Applications of multi-view learning approaches for software comprehension," *The Art, Science, and Engineering of Programming*, Vol. 3, No. 3, 2019.

[21] E.O. Kiyak, A.B. Cengiz, K.U. Birant, and D. Birant, "Comparison of image-based and text-based source code classification using deep learning," *SN Computer Science*, Vol. 1, No. 5, 2020, pp. 1–13.

[22] A.V. Phan and M.L. Nguyen, "Convolutional neural networks on assembly code for predicting software defects," in *2017 21st Asia Pacific Symposium on Intelligent and Evolutionary Systems (IES)*. Hanoi, Vietnam: IEEE, Nov. 2017, pp. 37–42.

[23] J. Chen, Y. Yang, K. Hu, Q. Xuan, Y. Liu, and C. Yang, "Multiview transfer learning for software defect prediction," *IEEE Access*, Vol. 7, Jan. 2019, pp. 8901–8916.

[24] D. Ulumi and D. Siahaan, "Weighted k-NN using grey relational analysis for cross-project defect prediction," *Journal of Physics: Conference Series*, Vol. 1230, Jul. 2019, p. 012062.

[25] R. Sathyaraj and S. Prabu, "A hybrid approach to improve the quality of software fault prediction using naïve bayes and k-nn classification algorithm with ensemble method," *International Journal of Intelligent Systems Technologies and Applications*, Vol. 17, No. 4, Oct. 2018, pp. 483–496.

[26] L. He, Q.B. Song, and J.Y. SHEN, "Boosting-based k-NN learning for software defect prediction," *Pattern Recognition and Artificial Intelligence*, Vol. 25, No. 5, 2012, pp. 792–802.

[27] R. Goyal, P. Chandra, and Y. Singh, "Suitability of k-NN regression in the development of interaction based software fault prediction models," *IERI Procedia*, Vol. 6, No. 1, 2014, pp. 15–21.

[28] S.K. Srivastava and S.K. Singh, "Multi-label classification of twitter data using modified ML-kNN," in *Advances in Data and Information Sciences*, Lecture Notes in Networks and Systems, K. M., T. M., T. S., and S. V., Eds., Vol. 39. Singapore: Springer, Jun. 2019, pp. 31–41.

[29] P. Villar, R. Montes, A.M. Sánchez, and F. Herrera, "Fuzzy-citation-k-NN: A fuzzy nearest neighbor approach for multi-instance classification," in *2016 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. Vancouver, BC, Canada: IEEE, Jul. 2016, pp. 946–952.

[30] Y. Xia, Y. Peng, X. Zhang, and H.Y. Bae, "DEMST-KNN: A novel classification framework to solve imbalanced multi-class problem," in *Artificial Intelligence Trends in Intelligent Systems*, Advances in Intelligent Systems and Computing, R. Silhavy, R. Senkerik, Z.K. Oplatková, Z. Prokopova, and P. Silhavy, Eds., Vol. 573. Cham, Germany: Springer, Apr. 2017, pp. 291–301.

[31] S. Gupta, S. Rana, B. Saha, D. Phung, and S. Venkatesh, "A new transfer learning framework with application to model-agnostic multi-task learning," *Knowledge and Information Systems*, Vol. 49, No. 3, Feb. 2016, pp. 933–973.

[32] E.O. Kiyak, D. Birant, and K.U. Birant, "An improved version of multi-view k-nearest neighbors (MVKNN) for multiple view learning," *Turkish Journal of Electrical Engineering and Computer Sciences*, Vol. 29, No. 3, 2021, pp. 1401–1428.

[33] S. Li, E.J. Harner, and D.A. Adjeroh, "Random KNN feature selection – A fast and stable alternative to random forests," *BMC bioinformatics*, Vol. 12, No. 1, 2011, pp. 1–11.

[34] I.H. Witten, E. Frank, M.A. Hall, and C.J. Pal, *Data Mining: Practical Machine Learning Tools and Techniques*, 4th ed., The Morgan Kaufmann Series in Data Management Systems. Cambridge, MA, USA: Elsevier Science, 2016.

[35] "Tera-promise data," accessed: 10.05.2020. [Online]. https://github.com/klainfo/DefectData/tree/master/inst/extdata/terapromise

[36] "NASA MDP data," accessed: 07.05.2020. [Online]. https://github.com/klainfo/NASADefectDataset/tree/master/OriginalData/MDP

[37] B. Turhan, T. Menzies, A.B. Bener, and J.D. Stefano, "On the relative value of cross-company and within-company data for defect prediction," *Empirical Software Engineering*, Vol. 14, No. 5, Jan. 2009, pp. 540–578.

[38] E. Borandag, A. Ozcift, D. Kilinc, and F. Yucalar, "Majority vote feature selection algorithm in software fault prediction," *Computer Science and Information Systems*, Vol. 16, No. 2, 2019, pp. 515–539.

[39] Z. Yao, J. Song, Y. Liu, T. Zhang, and J. Wang, "Research on cross-version software defect prediction based on evolutionary information," *IOP Conference Series: Materials Science and Engineering*, Vol. 563, Aug. 2019, p. 052092.

[40] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," *IEEE transactions on software engineering*, Vol. 33, No. 1, Dec. 2006, pp. 2–13.

[41] R.F. Woolson, *Wilcoxon Signed-Rank Test.* Wiley Encyclopedia of Clinical Trials, 2008, pp. 1–3.

[42] H. Liu, S. Zhang, J. Zhao, X. Zhao, and Y. Mo, "A new classification algorithm using mutual nearest neighbors," in *2010 Ninth International Conference on Grid and Cloud Computing.* Nanjing, China: IEEE, Nov. 2010, pp. 52–57.

[43] U. Lall and A. Sharma, "A nearest neighbor bootstrap for resampling hydrologic time series," *Water Resources Research*, Vol. 32, No. 3, Mar. 1996, pp. 679–693.

[44] J. Park and D.H. Lee, "Parallelly running k-nearest neighbor classification over semantically secure encrypted data in outsourced environments," *IEEE Access*, Vol. 8, 2020, pp. 64 617–64 633.

[45] P. Mitra, C. Murthy, and S. Pal, "Unsupervised feature selection using feature similarity," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 24, No. 3, 2002, pp. 301–312.

[46] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, Vol. 14, No. 2, 2009, pp. 131–164.

Appendix A. Description of datasets

Table A2. The main characteristics of the datasets

Group	Dataset	Release	Number of Instances	Defect (%)
Open Source Software Projects (OSSP)	ant	1.3	125	16.00
		1.4	178	22.47
		1.5	293	10.92
		1.6	351	26.21
		1.7	745	22.28
	arc	–	234	11.54
	berek	–	43	37.21
	e-learning	–	64	7.81
	forrest	0.7	29	17.24
		0.8	32	6.25
	jedit	3.2	272	3.31
		4.0	306	24.51
		4.1	312	25.32
		4.2	367	13.08
		4.3	492	2.24
	log4j	1.0	135	25.19
		1.1	109	33.95
		1.2	205	92.20
	pbeans	1.0	26	76.92
		2.0	51	19.61
	poi	1.5	237	59.49
		2.0	314	11.78
		2.5	385	64.41
		3.0	442	63.57
	prop	6	661	9.98
	redactor	–	176	15.34
	serapion	–	45	20.00
	synapse	1.0	157	10.19
		1.1	222	27.03
		1.2	256	33.59
	tomcat	–	858	8.97

Table A2 continued

Group	Dataset	Release	Number of Instances	Defect (%)
OSSP	velocity	1.4	196	75.00
		1.5	214	66.36
		1.6	229	34.06
	xalan	2.4	723	15.21
		2.6	885	46.44
		2.7	909	98.79
	xerces	1.2	440	16.14
		1.3	453	15.23
		1.4	588	74.32
Softlab	ar1	–	121	7.44
	ar3	–	63	12.70
	ar4	–	107	18.69
	ar5	–	36	22.22
	ar6	–	101	14.85
NASA MDP	cm1	–	498	9.83
	jm1	–	10885	19.00
	kc1	–	2109	15.45
	kc2	–	522	20.49
	pc1	–	1109	6.94

Table A3: The number of classes, views, and features of each dataset group

Dataset Group	#Classes	#Views	#Features				
NASA MDP	2	3	McCabe 4	Basic Halstead 9		Derived Halstead 8	
OSSP	2	5	Coupling 5	Complexity 3	Cohesion 3	Inheritance 3	Scale 6
SOFTLAB	2	4	Halstead 12	McCabe 3	LOC 5	Miscellaneous 9	

Table A4. Categories of software metrics in the NASA datasets

NASA MDP		
View	Symbol	Metric Full Name
MCCABE	v(g)	Cyclomatic complexity
	ev(g)	Essential complexity
	Iv(g)	Design complexity
DERIVED HALSTEAD	N	Total operators + operands
	V	Volume
	L	Program length
	D	Difficulty
	I	Intelligence
	E	Effort to write code
	B	Effort estimate

Table A4 continued

NASA MDP		
View	Symbol	Metric Full Name
	T	Time estimator
BASIC HALSTEAD	IOCode	Line count
	IOComment	Comment count
	IOBlank	Blank line count
	IOCodeAnd Comment	# of code and comment lines
	Uniq_Op	# of unique operators
	UniqOpnd	#of unique operands
	Total_Op	#of total operators
	Total_Opnd	# of total operands
	branch_count	# of branch counts

Table A5: Categories of software metrics in the OSSP datasets

	OSSP	
View	Symbol	Metric Full Name
COUPLING	ca	Afferent couplings
	cbm	Coupling between methods
	cbo	Coupling between object classes
	ce	Efferent couplings
	ic	Inheritance coupling
COHESION	lcom	Lack of cohesion in methods
	lcom3	Lack of cohesion in methods
	cam	Cohesion among methods of class
COMPLEXITY	amc	Average method complexity
	avg_cc	Average McCabe
	max_cc	Maximum McCabe
INHERITANCE	dit	Depth of inheritance
	moa	Measure of aggregation
	mfa	Measure of function abstraction
SCALE	loc	Lines of code
	noc	Number of children
	rfc	Response for a class
	npm	Number of public methods
	wmc	Weighted methods per class
	dam	Data access metric

Table A6: Categories of software metrics in the Softlab datasets

SOFTLAB		
View	Symbol	Metric Full Name
MCCABE	v(g)	Cyclomatic complexity
	iv(G)	Cyclomatic density
	Iv(G)	Design complexity
LOC	loc_total	Total lines of code
	loc_blank	number of blank lines
	loc_comments	number of comment lines
	loc_code_and_comment	number of code and comment lines
	loc_executable	number of lines of executable code
HALSTEAD	N1	number of operators
	N2	number of operands
	μ_1	number of unique operators
	μ_2	number of unique operands
	N	program length
	V	volume (program size)
	L	program level
	D	difficulty level
	I	content c
	E	effort to implement
	B	estimated number of bugs
	T	implementation time
MISCELLANEOUS	branch_count	number of branch counts
	call_pairs	number of calls to other functions
	condition_count	number of conditionals in a given module
	decision_count	number of decision points
	decision_density	Condition count / Decision count
	design_density	iv(G) / v(G)
	multiple condition count	number of multiple conditions
	normalized_cyclomatic_complexity	v(G) / number of lines
	formal parameters	Identifiers used in a method

Appendix B. Experimental Results

Table B1: Comparison of the KNN and MVKNN algorithms on the NASA MDP and Softlab datasets in terms of precision, recall, and *F1 Score*

NASA MDP						
	Precision		Recall		*F1 Score*	
Dataset	KNN	MVKNN	KNN	MVKNN	KNN	MVKNN
cm1	0.81	0.81	0.90	0.90	0.85	0.85
jm1	0.77	0.77	0.81	0.81	0.79	0.79
kc1	0.81	0.82	0.85	0.85	0.83	0.83
kc2	0.81	0.82	0.83	0.83	0.82	0.82
pc1	0.87	0.87	0.93	0.93	0.90	0.90
Avg.	0.81	**0.82**	**0.86**	**0.86**	**0.84**	**0.84**
Softlab						
	Precision		Recall		*F1 Score*	
Dataset	KNN	MVKNN	KNN	MVKNN	KNN	MVKNN
ar1	0.86	0.86	0.93	0.93	0.89	0.89
ar3	0.89	0.89	0.90	0.90	0.89	0.89
ar4	0.82	0.85	0.84	0.86	0.83	0.85
ar5	0.76	0.85	0.80	0.83	0.78	0.84
ar6	0.73	0.73	0.85	0.85	0.79	0.79
Avg.	0.81	**0.84**	0.86	**0.87**	0.84	**0.85**

Table B2. Comparison of the KNN and MVKNN algorithms on the OSSP datasets in terms of precision, recall, and *F1 Score*

OSSP							
		Precision		Recall		*F1 Score*	
Dataset	Release	KNN	MVKNN	KNN	MVKNN	KNN	MVKNN
ant	1.3	0.73	0.70	0.83	0.84	0.78	0.76
	1.4	0.62	0.60	0.77	0.77	0.69	0.67
	1.5	0.81	0.79	0.89	0.89	0.85	0.84
	1.6	0.74	0.78	0.76	0.79	0.75	0.78
	1.7	0.78	0.78	0.8	0.8	0.79	0.79
arc	–	0.81	0.78	0.88	0.88	0.84	0.83
berek	–	0.80	0.86	0.79	0.86	0.79	0.86
e-learning	–	0.85	0.85	0.92	0.92	0.88	0.88
forrest	0.7	0.71	0.85	0.78	0.83	0.74	0.84
	0.8	0.88	0.88	0.94	0.94	0.91	0.91
jedit	3.2	0.72	0.77	0.73	0.78	0.72	0.77
	4.0	0.78	0.82	0.79	0.82	0.78	0.82
	4.1	0.77	0.83	0.78	0.82	0.77	0.82
	4.2	0.85	0.88	0.87	0.88	0.86	0.88
	4.3	0.96	0.96	0.98	0.98	0.97	0.97

Table B2 continued

OSSP							
		Precision		Recall		*F1 Score*	
Dataset	Release	KNN	MVKNN	KNN	MVKNN	KNN	MVKNN
log4j	1.0	0.73	0.75	0.76	0.77	0.74	0.76
	1.1	0.76	0.82	0.76	0.81	0.76	0.81
	1.2	0.85	0.85	0.92	0.92	0.88	0.88
pbeans	1.0	0.63	0.58	0.71	0.73	0.67	0.65
	2.0	0.69	0.64	0.8	0.78	0.74	0.70
poi	1.5	0.67	0.7	0.67	0.7	0.67	0.70
	2.0	0.8	0.78	0.88	0.88	0.84	0.83
	2.5	0.75	0.78	0.75	0.78	0.75	0.78
	3.0	0.77	0.81	0.76	0.81	0.76	0.81
prop 6	–	0.81	0.81	0.9	0.9	0.85	0.85
redactor	–	0.85	0.88	0.87	0.89	0.86	0.88
serapion	–	0.74	0.85	0.8	0.82	0.77	0.83
synapse	1.0	0.81	0.81	0.9	0.9	0.85	0.85
	1.1	0.68	0.73	0.73	0.75	0.70	0.74
	1.2	0.7	0.7	0.71	0.71	0.70	0.70
tomcat	–	0.83	0.83	0.91	0.91	0.87	0.87
velocity	1.4	0.8	0.84	0.81	0.82	0.80	0.83
	1.5	0.7	0.77	0.72	0.74	0.71	0.75
	1.6	0.63	0.72	0.66	0.72	0.64	0.72
xalan	2.4	0.77	0.72	0.85	0.85	0.81	0.78
	2.6	0.69	0.75	0.68	0.74	0.68	0.74
	2.7	0.98	0.98	0.99	0.99	0.98	0.98
xerces	1.2	0.7	0.87	0.84	0.84	0.76	0.85
	1.3	0.81	0.89	0.86	0.89	0.83	0.89
	1.4	0.74	0.78	0.77	0.79	0.75	0.78
Avg.		0.77	**0.79**	0.81	**0.83**	0.79	**0.81**

Table B3: Comparison of single-view and multi-view accuracy values of the KNN and MVKNN algorithms on the OSSP datasets

ID	Dataset Name	KNN						MVKNN					
		view1	view2	view3	view4	view5	All views	view1	view2	view3	view4	view5	All views
1	ant 1.3	80.80	84.00	84.00	84.00	82.40	83.04	80.80	83.20	83.20	83.20	82.40	**84.00**
2	ant 1.4	77.53	76.40	76.40	76.97	76.97	76.85	75.84	74.72	74.16	72.47	74.16	**76.97**
3	ant 1.5	88.74	89.08	88.74	89.08	90.10	**89.15**	89.08	89.08	88.40	88.4	91.13	89.08
4	ant 1.6	78.06	76.92	73.79	73.22	80.06	76.41	77.49	76.35	75.21	75.78	78.92	**79.20**
5	ant 1.7	80.67	80.94	79.60	78.52	81.48	**80.24**	80.81	80.94	79.46	78.93	80.27	80.00
6	arc	88.46	88.46	87.18	88.46	88.89	**88.29**	88.03	88.03	88.03	87.61	87.61	88.03
7	berek	81.40	83.72	76.74	72.09	81.40	79.07	72.09	90.70	86.05	72.09	88.37	**86.05**
8	e-learning	92.19	92.19	92.19	92.19	92.19	**92.19**	90.62	92.19	92.19	92.19	90.62	**92.19**
9	forrest 0.7	86.21	75.86	75.86	82.76	68.97	77.93	82.76	75.86	75.86	89.66	72.41	**82.76**
10	forrest 0.8	93.75	93.75	93.75	93.75	93.75	**93.75**	93.75	90.62	90.62	93.75	93.75	**93.75**
11	jedit 3.2	70.59	66.54	73.90	75.74	76.47	72.65	68.38	65.44	73.53	75.00	76.10	**77.57**
12	jedit 4.0	78.10	79.74	76.80	79.41	81.70	79.15	74.51	80.07	75.49	80.39	81.37	**82.03**
13	jedit 4.1	81.41	77.24	77.56	76.92	78.53	78.33	80.45	78.21	78.85	78.21	80.13	**81.73**
14	jedit 4.2	87.74	86.65	87.19	86.92	88.01	87.30	88.56	86.92	88.01	87.19	87.47	**88.28**
15	jedit 4.3	97.76	97.76	97.76	97.76	97.76	**97.76**	97.76	97.76	97.76	97.76	97.76	**97.76**
16	log4j 1.0	77.04	71.11	78.52	77.04	77.78	76.30	76.30	72.59	80.00	80.00	80.00	**77.04**
17	log4j 1.1	77.06	74.31	70.64	77.06	79.82	75.78	75.23	71.56	76.15	80.73	79.82	**80.73**
18	log4j 1.2	92.20	92.20	92.20	92.20	92.20	**92.20**	92.20	92.20	92.20	92.20	90.73	**92.20**
19	pbeans 1	73.08	69.23	69.23	76.92	69.23	71.54	76.92	65.38	80.77	76.92	73.08	**73.08**
20	pbeans 2	80.39	78.43	84.31	80.39	80.39	**80.78**	80.39	74.51	82.35	78.43	74.51	78.43
21	poi 1.5	72.15	57.38	65.82	69.2	70.04	66.92	70.04	65.82	64.98	68.78	69.62	**70.04**
22	poi 2.0	88.22	88.85	88.22	88.22	88.22	**88.35**	88.22	88.85	86.94	88.22	88.22	88.22
23	poi 2.5	78.96	67.27	76.36	73.51	79.74	75.17	76.36	71.69	77.92	72.21	81.30	**77.92**
24	poi 3.0	76.70	78.05	75.34	73.98	75.79	75.97	76.24	77.15	76.70	73.98	78.28	**80.77**
25	prop-6	90.00	90.00	90.00	90.00	90.00	**90.00**	89.85	90.00	90.00	90.00	90.00	**90.00**
26	redaktor	88.07	84.09	90.34	86.36	87.50	87.27	85.23	85.80	90.34	87.50	87.50	**89.2**
27	serapion	80.00	80.00	77.78	80.00	82.22	80.00	77.78	71.11	77.78	80.00	86.67	**82.22**
28	synapse 1.0	89.17	89.17	89.81	89.81	89.81	89.55	89.17	89.17	88.54	89.81	88.54	**89.81**
29	synapse 1.1	72.52	70.27	75.68	71.17	76.13	73.15	72.07	72.07	73.87	72.07	77.03	**74.77**
30	synapse 1.2	71.09	71.88	73.44	67.58	72.27	71.25	69.53	70.70	72.27	67.19	70.70	**71.48**
31	tomcat	91.03	91.03	91.03	91.03	91.03	**91.03**	91.14	90.91	90.91	90.79	91.49	**91.03**
32	velocity 1.4	80.10	75.51	84.69	83.67	80.10	80.81	81.63	74.49	84.69	84.69	77.04	**82.14**
33	velocity 1.5	73.83	73.36	70.09	66.82	74.30	71.68	70.56	71.50	72.90	69.63	72.90	**74.30**
34	velocity 1.6	64.19	62.01	71.18	64.19	66.81	65.68	64.19	68.12	69.87	67.25	67.25	**72.05**
35	xalan 2.4	84.79	84.79	84.79	84.79	84.92	**84.82**	84.09	84.09	84.65	84.92	84.65	84.79
36	xalan 2.6	72.43	71.30	65.88	61.13	71.64	68.48	70.17	71.86	69.27	69.49	74.12	**73.67**
37	xalan 2.7	98.79	98.79	98.79	98.79	98.79	**98.79**	98.79	98.79	98.79	98.79	98.79	**98.79**
38	xerces 1.2	83.86	83.86	83.86	83.86	83.86	**83.86**	83.86	83.64	83.86	83.86	83.64	**83.86**
39	xerces 1.3	86.75	84.55	86.75	83.44	87.64	85.83	86.53	84.33	88.52	85.65	87.64	**88.52**
40	xerces 1.4	71.94	77.89	92.18	72.45	71.94	77.28	72.45	77.89	92.52	72.45	71.60	**79.42**
	Average	81.94	80.36	81.71	80.79	82.02	81.37	81.00	80.36	82.34	81.45	82.19	**83.10**

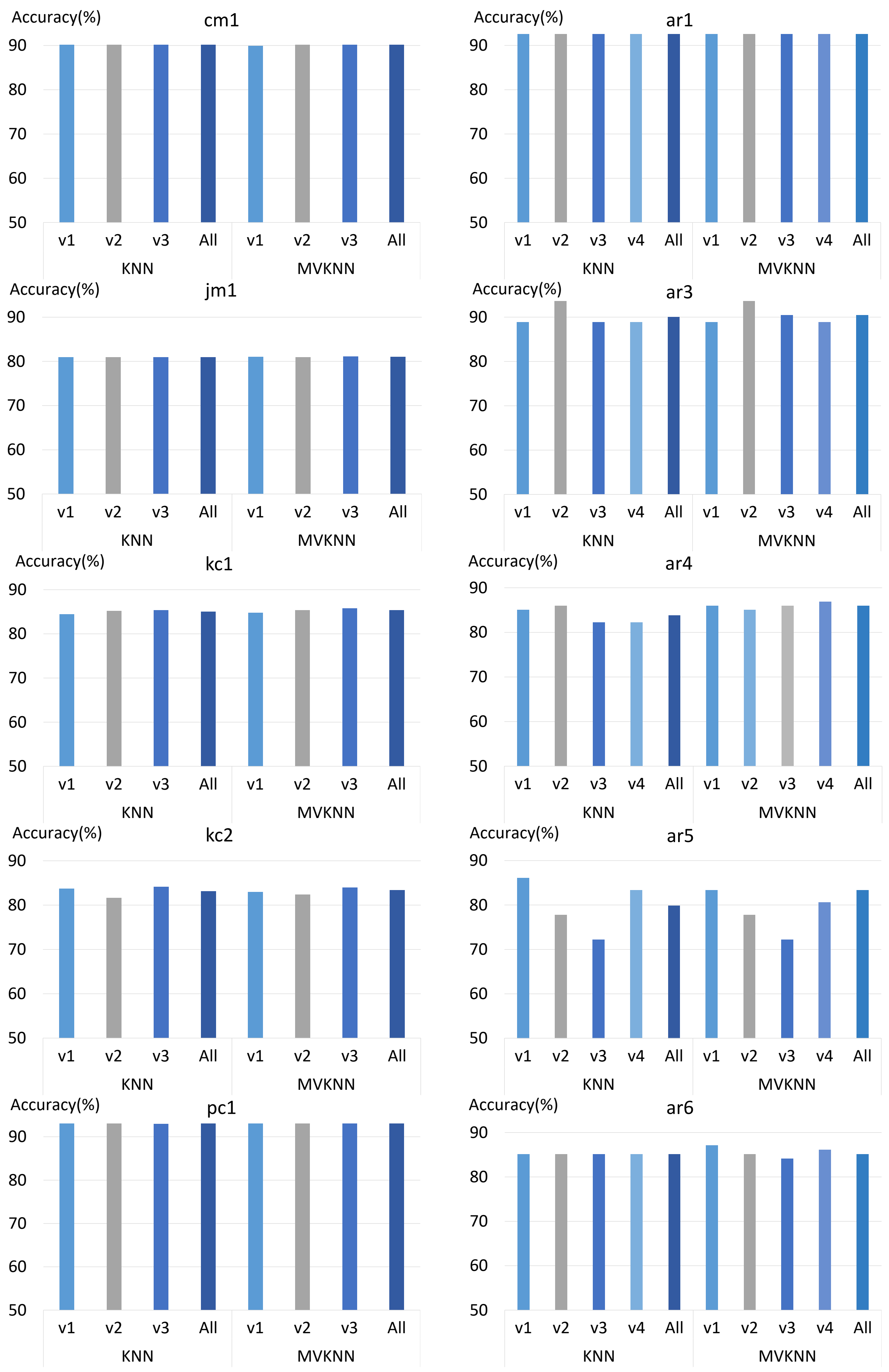


Figure B1: View-based comparison of the KNN and MVKNN algorithms

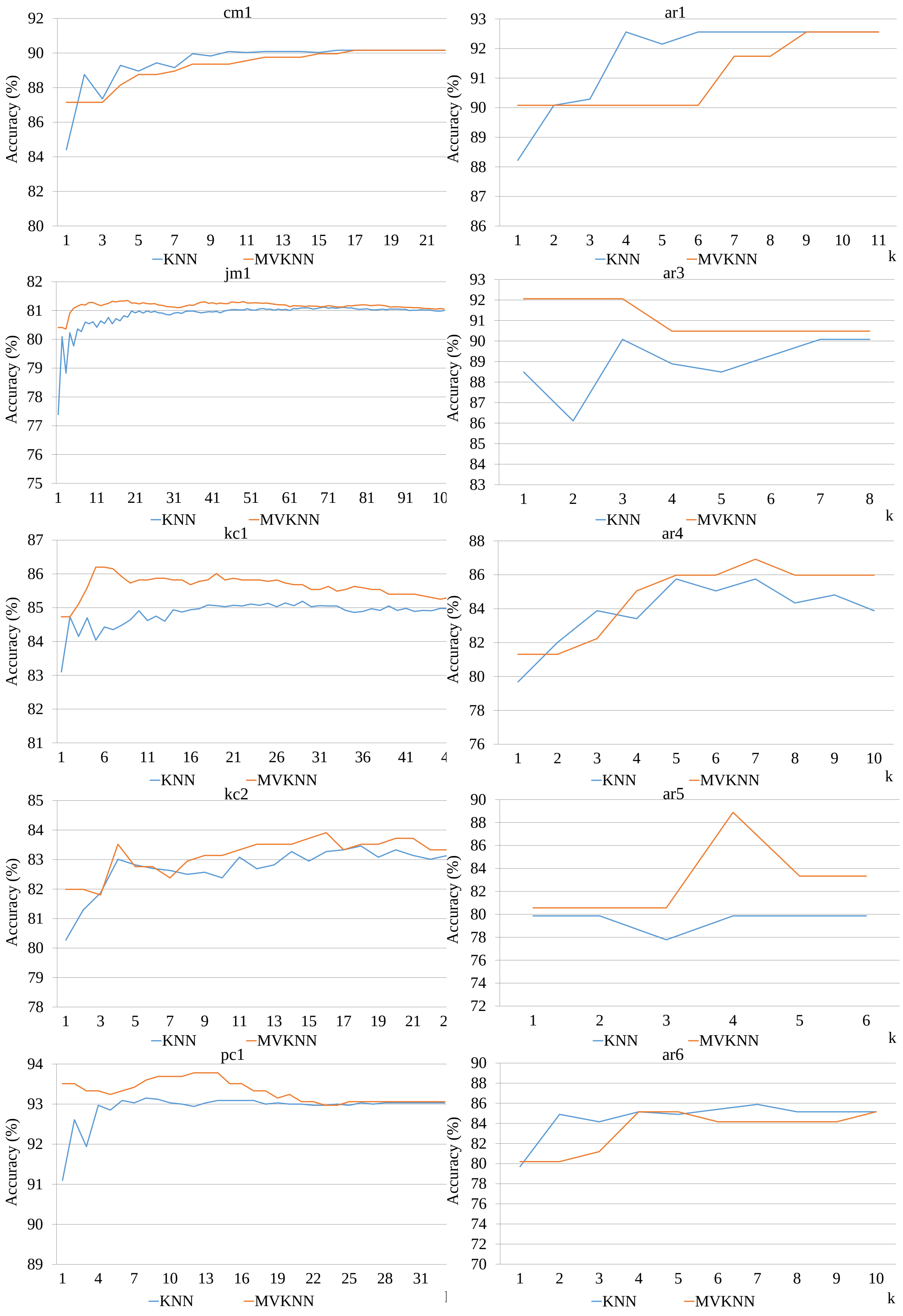


Figure B2: Comparison of single-view and multi-view versions of the KNN algorithm on various k values

e-Informatica Software Engineering Journal, Volume 15, Issue 1, 2021, pages: 185–204, DOI 10.37190/e-Inf210109

Business Model Flexibility and Software-intensive Companies: Opportunities and Challenges

Magnus Wilson*, Krzysztof Wnuk**, Lars Bengtsson***

*Ericsson, Karlskrona, Ericsson, Sweden

**Department of Software Engineering, Blekinge Institute of Technology, Sweden

***Industrial Engineering and Management, LTH, Lund University, Sweden

magnus.wilson@ericsson.com, krw@bth.se, lars.bengtsson@design.lth.se

Abstract

Background: Software plays an essential role in enabling digital transformation via digital services added to traditional products or fully digital business offerings. This calls for a better understanding of the relationships between the dynamic nature of business models and their realization using software engineering practices. Aim: In this paper, we synthesize the implications of digitalization on business model flexibility for software-intensive companies based on an extensive literature survey and a longitudinal case study at Ericsson AB. We analyze how software-intensive companies can better synchronize business model changes with software development processes and organizations. Method: We synthesize six propositions based on the literature review and extensive industrial experience with a large software-intensive company working in the telecommunication domain. Conclusions: Our work is designed to facilitate the cross-disciplinary analysis of business model dynamics and business model flexibility by linking value, transaction, and organizational learning to business model change. We believe that software engineering tools and methods can play a crucial role in enabling more automated synchronization between technology and business model changes.

Keywords: business flexibility, digital business modeling, equivocality, learning organization

1. Introduction

Digitalization brings new opportunities and increased connectivity is the primary fuel for digitalization. Ericsson, as a major player in the telecommunications market, is an actor deeply involved in this process[1]. The advent of the 5G network stands as a prominent example of opportunities and challenges associated with massive connectivity when all value-chain members and partners must rethink or reorganize their positions if necessary. For many companies, 5G will force them to redefine their business offerings and create new business opportunities. However, with this speed of technological changes, the business models can not remain static or re-actively respond to changes.

Digitalization drives significant changes to the process level, organization level, business level of any company and its customers [1]. Digitalization offers a significantly shorter transaction turnaround time. Consequently, the increased transaction speed drives new challenges for the alignment between business and technology changes. Companies that used to sell traditional products enter new markets and ecosystems where digital products and services dominate. The logic of creating these products, monetizing their core value, and maintaining them is significantly different and often counterintuitive at first glance. Thus, software engineers and managers often need to rethink their strategies and operational processes to better align with the nature of the digital business. An example here could be data-driven

[1]https://www.ericsson.com/en/reports-and-papers/networked-society-insights [last visited 23.06.2021].

Submitted: 20 November 2020; Revised: 30 June 2021; Accepted: 03 August 2021; Available online: 05 November 2021

experimentation and feature discovery via A/B testing that significantly changes requirements engineering practice and demands great changes to software architecture [2].

This paper discusses the implications of digitalization for software-intensive companies based on an extensive literature survey and a longitudinal case study at Ericsson. We synthesize six propositions for improved handling of business model change and discuss each proposition's implications on software engineering practices and principles. This paper presents a cross-disciplinary synthesis of digitalization's impact on the alignment between business and technology change (including software engineering methods and tools). We also discuss new ways of handling business model flexibility in software-intensive product development.

Companies are undergoing significant transformations and are struggling with the alignment of business and technology changes [3]. Until recently, companies handled increasing size and complexity by 1) clearly distinguishing between the planning and realization layers for company strategy, product portfolios, and individual products; and 2) handling change mainly in the realization layer and ensuring that the planning layer remains reasonably stable.

Digitalization increases the speed of change in the planning layer, which in many cases, reaches the speed of changes in the realization layer. As a result, negotiation and risk management can no longer only rely on the sales and engineering departments, as the business models shift focus to the ecosystem and collaboration [4, 5], and companies choose operating multi-business-models [6]. Business modeling literature also recognizes the need for efficiently handling change as several authors discuss the dynamic nature of business models and change in the business environment, e.g., [5, 7, 8], just to name a few.

The paper is structured as follows: In Section 2, we present our synthesis based on background and related work. In Section 3, we present how the business environment changes for our industry case and our findings from the longitudinal study. In Section 4, we summarize and discuss our results using the derived value membrane concept and develop one additional proposition. In Section 5, we conclude our paper.

2. Background and related work

The synthesis provided in this section is based on an extensive systematic literature review about efficiency, effectiveness [9], and flexibility of business modeling [10], published in our previous work and updated using the snowballing literature review method. It is also derived from our design science study on capturing changing *business intents* using *context frames* [11]. Our synthesis responds to multiple requests for cross-disciplinary research agenda [12–15]. The focal point of this study is the misalignment between the planning (define) and the realization (execute) of the software business in the fast-changing environment that a software-intensive company operates. A change to either the strategy or the realization has the potential to trigger an escalating misalignment. Formulating and executing a digitalization strategy [16] has the goal of reducing such misalignment by managing the change. The term digital transformation strategy implies a business-centric context when coordinating strategies for products, services, and business models as a whole.

Inspired by Ritter and Littl's focus on broader implications for business-model research, we take the analogy for the business model as a *membrane* between theories [12]. By analyzing uncertainty and equivocality [17] with value within a transaction, as the membrane between two actors in an activity system [13], we propose the business model can also act as the "contextual agent" in what we call the *value membrane* (VaM). This helps identify the cause of the misalignment and minimize gaps between needed change, planned change, and implemented change.

Most scholars focus on detecting or preparing change at one level (strategy, portfolio, or product) or analyzing the organization's broader external aspects, without integrating the activities [9]. Many scholars call for further research on change realization, e.g., [14, 18, 19]. Meier and Bosslau argue that there is almost no attention in research to the dynamic aspects, flexibility, validation, and implementation of business models [20], while Richter et al. emphasize the importance of understanding the degree of flexibility needed to realize change [21]. Seeing business models as activity systems helps organizations (as responsible for the business) adapt to change and generate value [15]. Therefore, our focus is primarily on the dynamics aspects of the business model change and business flexibility and its implications on software engineering.

3. Research method

We utilized the snowballing literature review method to collect relevant articles [22]. The start set was the articles identified in our two previous literature reviews [9, 10]. These two studies used the following search string on the Google Scholar database:

SS1: (business modelling OR business model OR business ecosystem) AND value creation AND strategy,
SS2: ("business modelling" OR "business modeling" OR "business ecosystem") AND "business strategy" AND "value creation" AND ("effectiveness" OR "efficiency" OR "business flexibility" OR modularity OR "variability in realization" OR "governance" OR "multi-business").

Executing SS1 and SS2 (limited to title-abstract-keywords) and screening candidates left us with ten papers in the start set. After 4 snowballing iterations, we included 58 studies [9, 10]. These 58 articles were screened to find new citations after 2018.

3.1. Updates for new papers after 2018

The previously selected 58 papers now became the start set for one snowballing iteration. As we are looking for new evidence, we only analyzed citations since references to these 58 papers were analyzed in our previous work [9, 10].

16833 new iterations were identified since 2018 and screened. From these citations, 60 articles were identified from the title and abstract screening. These 60 articles were carefully read and further evaluated. We excluded 40 papers because they were focusing on business model innovation by creating new business models. Six papers were excluded after the full read since they focused on a general notion of a business model. Fourteen papers were finally accepted and included in the synthesis. Next, we revisited the previously selected 58 papers from the previous literature review [9, 10]. Each of these 58 papers was carefully screened and evaluated focusing on the implications of digitalization on business model change. We included seven papers from the 58 evaluated. The total set of papers used for the synthesis included 24 papers, detailed in Table 1.

3.2. Data analysis and synthesis

The 24 papers included in the data analysis and synthesis were carefully investigated. We focused on analyzing patterns within the identified papers, according to the steps recommended by Cruzes [44]. Two authors read all 24 papers and identified relevant segments of text associated with digitalization and business model change. Next, these segments were discussed in a meeting, and 25 codes were identified using the open coding technique. Next, differences and similarities between these codes were discovered, and codes were merged into higher order statements. We focused on associating the 25 codes with the following categories: digitalization, value transformation, business model change, business flexibility, abstraction layers in business model change. Next, we constructed interpretations in each area and explored the relationships between the five themes (areas). Our high-order factors became the propositions presented in this paper. We provide the list of the most relevant articles from the set of 24 articles included in this work for each proposition. Our **theory (frame of reference)** was that digitalization has an impact on software engineering practices and product offering at Ericsson and also changes the current business models.

Table 1. Selected papers including a short summary of the main contributions in these papers and the most associated propositions

Paper	Authors	Comments	Associated Proposition(s)
P1	Woodard et al. [23]	Digital business strategy and component architectures	3, 6
P2	Chew [24]	Linking servitization and business model design	4
P3	Romero and Molina [5]	Engineering dynamic business models with the help of network organizations and customer communities	1
P4	Meier et al. [20]	Dynamic business models for product-service systems	3, 4
P5	Richter et al. [21]	Flexibility in product-service systems via use-oriented business models	1, 4
P6	Eurich et al. [25]	Business Model innovation process with network thinking and holistic approach	5
P7	Mason and Mouzas [26]	Flexible business models and their architectures	1, 4
P8	Gul [27]	Changes in business models and digital strategy	4, 6
P9	Sjödin et al. [28]	Value creation and value capture in business model innovation	4
P10	Antikainen [29]	Business Model Experimentation	2, 3
P11	Trapp et al. [30]	Business model innovation tools	5
P12	Chritofi et al. [31]	How agility and flexibility is discussed in business research	4
P13	Teece [32]	Business Model and Dynamic Capabilities	2
P14	Linde et al. [33]	Value capture model for digital servitization	2, 5
P15	Hacklin et al. [34]	Migrating value in business model innovation	2, 6
P16	Vendrell-Herrero et al. [35]	Business model experimentation in dynamic contexts	3, 4
P17	Szopinski et al. [36]	Software tools for business model innovation	5
P18	Wirtz [37]	Drivers that trigger business model change	2
P19	Gebauer et al. [38]	Digitalization and servitization	4
P20	Moellers [39]	System dynamics in business model innovation	3
P21	Nailler et al. [40]	Business model evolution and value anticipation	4
P22	Clauss et al. [41]	Strategic agility and business model innovation	2, 5
P23	Schaffer et al. [42]	Dynamic business models	3, 4, 5
P24	Pratama and Iijima [43]	Linking value and business models	2

3.3. The longitudinal study, industrializing services at Ericsson

3.3.1. Case study research design and data collection methods

We report the case study objectives and other design aspects following guidelines suggested by Runeson and Höst [45] The **objective** of the case study was exploratory and focuses on digitalization and the resulting increased flexibility in offering of software-intensive products and services at Ericsson. Digitaliation is a contemporary complex phenomenon; therefore the best approach is to study it in a real world context. We opted for a holistic case study [46] with one **unit of analysis** (service organization). We decided to conduct a longitudinal case study at Ericsson, following the development and growth of the service organization and its impact on software engineering practices. Observations and participation took place for 4 years, giving us the opportunity to explore and understand the implications and impact of digitalization on business models and software engineering practices.

Our **goal** was to explore the impact of service transformation on the business models that Ericsson offers and on software engineering practices utilized to execute these business models. Ericsson had a strong division between the research and development teams. The research organization mostly develops new solutions while the development organizations focus on deployment and customer adaptations for various global regions.

The **theory** associated with this case study assumes that the introduction of digital services impacts the ways of working and handling business operations. Since the offered products are mostly digital, their deployment could be continuous and remote, and their update time is drastically reduced. This has also impact on business models. Back in 2012, the Ericssons' service organization was mainly working in two types of business models:

– Managed Services – running the operator's network for them with large, long-term contracts.
– Service consultancy and Delivery model – focused on project deliveries and learning services.

As part of a corporate strategy, the service organization devised their strategic program "Global Scale – Local Reach", involving 75000+ resources (global, regional, and contractors) in nine regions, working in three segments of the service portfolio (Managed Services, Product Related Services, and Consulting and System Integration). The goal of the program was to improve customer responsiveness, improve productivity, and improve internal evaluation. The part of this transformation included either offering current products as services or creating services on top of the current software-intensive products. In many aspects, Ericsson followed the servitization transformation of the business environment[47].

Data Collection and Analysis Methods. We combined observations, document analysis, and interviews [48]. We also actively participated alongside program managers, the steering group, and requirements analysts. The research team has analyzed the collected empirical data and synthesized it in Section 5.2. Between 2012–2016, we actively worked alongside teams responsible for:

– supporting the program manager and his steering group with a business and enterprise architecture analysis,
– responsible for the business level requirements towards tools and IT development, and
– consultants for the deployment (business processes and training) into the sales and delivery organization (global plus nine regions).

At the beginning of the program (2012–2013), we participated in eleven extensive workshops interviewing practitioners from affected areas: finance; product management (services and software products); key account managers; Ericsson IT (master data, business processes, and system responsible); sales; delivery (project); and support processes (planning, development, and pricing, of services). The 3–4 hours workshops were based on a short introduction to the workshop and the program, followed by practitioners presenting their current business processes and ways of working. Practitioners were then interviewed on current issues, and potential opportunities were discussed under the frame of the new program, providing us with great insights into the scope plus the strategical and operational issues facing the program. The workshops also provided a deeper understanding of uncertainty, equivocality, and rivalry between the different roles and organizations. We were also given continuous access to all program-related information, monthly reports, and steering group protocols. We also conducted two sets of individual, 60+ minutes interviews with a delivery project manager and a solution architect, to identify any misalignment against the program's goals and the actual outcome.

3.4. Validity threats

We adopted the validity guidelines suggested by Runeson [45]. We mitigated the industrial experience bias of the leading author by involving the other two authors as reviewers of the work. We have also followed the thematic analysis approach steps [44]. The selected 24 papers are highly heterogeneous and therefore minimize the bias on specific author or terminology. To minimize the data synthesis bias, two researchers performed the initial read and coding, and these codes were later discussed and merged.

We minimized potential internal validity threats by following the snowballing literature review guidelines [22]. Because of the interdisciplinary nature of this study, the risk remains that some aspects are underrepresented and other aspects are over-represented. In particular, business model innovation or business process modeling seems to be heavily researched in the business management and the computer science community. However, we decided to focus on the interplay between business model change and digitalization and excluded papers that primarily focus on business model innovation realized by the creation of new business models.

Finally, we are aware that a single case study presented in this paper may not offer sufficient external validity. However, we opt for analytical generalization rather than statistical one as suggested by Flyvbjerg [49]. We provide an extensive description of the analyzed case and contrast it with the findings from the literature review.

4. Results

We have synthesized five propositions based on the literature review results and one based on the case study. We applied thematic synthesis to the papers presented in Table 1. The propositions are detailed in the subsections that follow.

4.1. The impact of digital transformation on the nature of negotiating a business deal and equivocality

Negotiating a business deal was traditionally a discussion focused on the functionality, price, and any potential project risks. The surrounding business environment (legislation, platforms and technology, partners and competition, etc.) gave little uncertainty related to the lifespan of the contract and the contractual obligations. Therefore, the negotiations could focus on the scope and usage of the underlying technical (software-based) solution. The business environment, including actors, business processes, and infrastructure, was predominately "stable within reasonable risks" throughout the lifespan of the contract and could be tracked by strategic planning, competitor and market analysis, monitoring standardization, and other regular management initiatives.

For example, the negotiations in the GSM and 3G telecommunication standardization included a well-defined business environment and interfaces between the components. Suppliers could concentrate their risk management to monitor and participate in the standards development while mainly focusing on optimal technology solutions for each component. Negotiating a new business deal was fundamentally about understanding what components, the quantity, and any potential customer-specific features needed to sweeten the deal. This kind of *contractual flexibility* could be implemented by the product and solution engineers under the strict coordination and risk management of sales, product management, and top management.

Software Engineering has developed several concepts to support contractual flexibility, e.g., implementing Software Product Lines (SPL), iterative, lean, and agile software development with daily code deliveries enabling increased customization. The ways of working need to be synchronized with other core business processes like sales and delivery, and hence into the business model. Product Service Systems (PSS) [50, 51], Industrial PSS [20, 52], and service-based business models [53, 54] are examples of how this fusion of engineering and business processes is continuously evolving.

With the digital transformation of the business environment [3, 55], negotiation and risk management can no longer rely on the sales and engineering departments but need to enact business model changes towards ecosystem and collaboration [4], [5]. Romero and Molina advocate collaborative networked organization and customer communities for supporting value co-creation and innovation [5]. These experience-centric networks help for co-creating value not only for the customers (like the previous supplier relationships) but also formulating alliances between the companies offering digital (software) products to its customers. The key enabler for these type of partnerships is openness in not only in shared source code but also Open Innovation initiatives [56]. This also means sharing software development tools and environments and openly discussing future plans and requirements.

Richter et al. suggest focusing on user-oriented business models that capture the necessary flexibility for product-service systems [21]. He includes *agility* as one of the aspects of changeability. Considering software development as a capital investment that should bring value to the customers is critical since software products need maintenance and operational support. This part is often neglected by Agile software development that focuses primarily on delivering new functionality rather than maintaining existing systems. Efficient maintenance improves long-term performance and changes the risk profile to asymmetric by introducing more flexibility early in the process.

Mason and Mouzas introduce the concept of "flexible business models" to capture and realize the necessary flexibility [26]. They include transaction relationships and ownership as the most critical aspects of flexibility. This has implications for software engineering since companies do not need to own the entire codebase and often co-create value in a software ecosystem. Gul describes what new strategies companies should realize in the digital environment, such as software [27].

The negotiating power, coming from knowing what *business flexibility (BF)* can be offered and how this business flexibility is translated into contractual flexibility that can be absorbed by the business model realization. The realization should be done without jeopardizing the underlying effectiveness and efficiency of products and technical solutions (promised contractual characteristics); emerges as a critical competitive advantage. However, with more roles participating in the negotiation [11, Figure 7 p. 1182], uncertainty and equivocality (multiple and conflicting interpretations of a goal, situation, or task) can negatively impact the quality, cost, and lead-time of both the planning and realization phases [17, 57, 58].

Companies undergoing the digitalization transformation should detect if the previously used realization strategy (the combination of the business model, products, and services) still will adhere to the changed contractual terms and conditions. This involves checking if the current business model will accommodate the new terms and conditions and the associated risks to deliver the changed contractual terms. The distance between strategizing, innovating, and planning for Business Model Change (BMCh) is significantly reduced. We argue that such risk management should be done before signing any contract, and therefore propose that,

Proposition 1: A mechanism for early detection of business model change is a critical factor in maintaining a company's negotiating power to ensure business success via improved risk management derived from the business flexibility.

The impact on software engineering. Software engineering should more clearly focus on risk management and negotiation. Risk management has traditionally been assigned to project management activities. Risk management in software engineering was performed assuming that the business model remains stable and the identified risks are most of technical nature [59]. We postulate that more effort should be dedicated to risk management on the requirements level. Some work was already done in the uniREPM model, where risk management is divided into project risk management and requirements risk management [60]. We believe that requirements risk analysis should be extended on the impact on business models and revenue strategies. Software managers should also consider software development as a capital investment that is long-term. This means taking care of software platforms and architectures and minimizing technical debt as much as delivering functionality and responding to ever-changing customer needs. Test automation is also a necessary component for keeping the negotiating power and understanding the limitations of the current codebase.

4.2. The gap between business model planning and execution

Business model experimentation is gaining more importance for software companies as a response to a growing need for business model innovation [61] and digitalization [3]. Experimentation is an approach to achieve effective change to the business, driven by the rationale that in "*highly uncertain environments, strategies are about insight, rapid experimentation, and evolutionary learning as much as the traditional skills of planning and rock-ribbed execution*" [62]. Many companies that offer software products invest in product decision support based on experimentation and A/B testing [63]. Despite the unquestionable potential of online experimentation, they often provide very incremental improvements and are not suitable for radical strategic changes. Business model associated changes are often more radical than incremental. This contributes to a gap between business model planning and execution.

To analyze the gap between planning and execution, we complement Höfflinger's top-down definition of the business model with Rohrbeck et al. bottom-up definition of business modeling, "*to be a creative and inventive activity that involves experimenting with content, structure, and governance of transactions that are designed to create and capture value*" [64].

Rohrbeck et al. focus on experimenting as a "round-trip" process of "translating an idea into execution, test, evaluate, and change until satisfied" (similar to the agile method of developing software products followed up by proper retrospectives). Secondly, they also focus on transactions, connecting the business model to human behavior and value in execution and planning activities. Thirdly, they make a clear distinction between value created and captured, as two (role-dependent) views of a transaction, implying an information representation suitable for maintaining (observe, analyze, decide, change) many relationships to support effective and efficient collaborations (through all the stages of the business model lifecycle, e.g., plan, design, deployment, execution, phase out).

Antikainen et al. [29] suggest the business model experimentation method that supports companies in innovating their B2B business models by benefiting from the shared economy opportunities. The ownership principle is replaced by temporary access and reusing (the two embedded characteristics of software as one piece of software can be reused forever and shared with as many partners as required).

Linde et al. [33] suggest a framework for capturing value while designing, developing or scaling digital services. They highlight agile development and value co-creation (risk and reward sharing) as the two main elements of value and revenue creation for digital services. This moves the main responsibility for a service offering from primarily the software company to the ecosystem of partners that share the risks and benefits.

Teece [32] highlight the role of dynamic capabilities in responding to changing business needs and customer requirements. Software and software engineering capabilities should be considered as such dynamic capabilities that constitute the strength of business agility of digital organizations. As dynamic capabilities are underpinned in organizational routines, selecting the appropriate software development processes and caring about team values and culture becomes important for software organizations. Software companies often experience pivots or other radical changes. In these cases, organizations with high absorptive capability respond and often succeed in this transformation.

Hacklin et al. [34] describe how value is migrated during business model innovation in computer and telecommunication industries where value migrated to value-added service providers from device manufacturers, network providers, and infrastructure companies. This example shows clearly that software as the primary carrier of value not only can penetrate many industries but also disrupt value creation and capture in highly established and often regulated industries. This has deep implications for software engineering principles used by these companies.

Clauss et al. [41] describe how value creation and value capture relate to strategic agility in turbulent business environments. Wirtz [37] out-lines markets, technology, and deregulation as the main drivers for business model change. Software plays a key role in the deregulation of many industries, and the Open Source Movement removes monetary incentives in selling software as source code. Software products and services disrupt many "traditional" industries, such as for example finance or automotive. Therefore, software engineers have to remember that the potential of software innovation stretches greatly beyond the software industry.

Pratama and Iijma [43] describe how to translate the value proposition components from the existing business model to a new business model without losing the content. This approach has important implications for software engineering since the software is almost never fully disposed or destroyed, rather reused or reshaped with the new business idea in mind.

Inspired by Fjeldstad and Snow, we adopt the idea of value as the contingency variable affecting all other elements of the business model [15], and to understand the transaction- and role-dependent *Direction of Value (DoV)*, we build on the value concept proposed in the Value Delivery Metamodel (VDML) [65]. Neither Höfflinger [8], Fjeldstad and Snow [15], nor VDML [65] makes a clear separation between value creation and value capture. Therefore, we postulate that:

Proposition 2: Value translation and value transformation capabilities are essential for business modeling. By exploring value, in an interaction on the individual level as the unit of analysis, we can resolve ambiguities in relation to the different areas of the business model (e.g., product delivery, product development, finance, customer relationships, partner management) stemming from: (1) the *direction of value*; (2) inter-level relationships of source and target for value; and (3) aggregation issues for value creation and value capture (scalability and value slippage).

Impact on Software Engineering. We postulate that business model experimentation should be integrated with data-driven continuous experimentation [66, 67]. For example, The RIGHT model for continuous experimentation is a good start as it has the business model and strategy element in the build-measure-learn process [66]. We believe that this integration should support the transformation into a "data-driven organization at scale" [67], where continuous experimentation is synchronized with business model evolution. We also postulate that software engineers need to consider two aspects when starting the development of new features or products: 1) what is the business viability of these features or products, and 2) how can we co-develop or co-create value.

4.3. Handling business model change

Both radical or incremental business model changes need to be addressed both at the planning and the realization levels [7]. Cavalcante et al. [68] divided BMCh into four types of change: business model creation, extension, revision, and termination. They further argued there is a "pre-stage" of "potential of BMCh" before the actual change occurs, often including analysis, experimentation, and other activities to build insights, learning, and commitment. In software engineering, this phase would include extensive prototyping or building the minimum viable product. Therefore, Cavalcante proposes to develop a detailed guide for analyzing BMCh, both at the level of cognition as well as action, where he sees continuous experimentation and learning as fundamental pillars for effective BMCh, transforming the company into a "*permanent learning laboratory*".

To address change on the planning level, a company needs to understand the As-Is situation (which capabilities exist) and the effects on the To-Be situation (needed abilities). Such insights require understanding how strategy relates to a business model [23], tactics, and residual choices [69], in combination with what strategic agility [70] and level of strategic flexibility [27, 71] the organization has. This could be achieved by business model experimentation as pointed out by Antikainen [29] and highlighted in proposition 2. Flexible business models and their architecture appears to be the central concept here [26]. Dynamic business models and their dependencies in the complex software-intensive systems emerge as an area with increasing importance for further research [42].

To facilitate such insights, we propose to represent a business model by combining the work by Ghezzi's on value networks (VN) and resource management (RM) [72], with Osterwalder's business model canvas (BMC) [73]. Therefore, a company's need for business model change can be derived from having profound knowledge and a sound understanding of the three dimensions: (1) the customer(s) and related relationships; (2) the value proposition (revenue streams, what values to create, how to deliver it to the customer); and (3) the company's assets (products, resources, activities, cost structures, and partner relationships).

Woodard et al. [23] divide digital business strategy into design capital and design moves. The important but often invisible aspects of design capital are technical debt and option value. Both have a fundamental impact on the business agility of software development organizations. An organization holding significant technical debt loses a lot of flexibility in realization and has limited options for creating and delivering value.

Meyer and Boßlau [20] suggest developing both products and services at the same time and therefore capturing more customer value and building long-term relationships with the customers. This helps to integrate business model design and engineering activities. For software services, it appears to be very beneficial due to the possibility of dynamically deploying and updating software services for the customers.

Vendrell-Herrero [35] study the economic value of business model experimentation in many sectors and industries. Experimentation helps to strengthen the network effects and also capture value from various customer needs. Therefore, establishing software-driven experimentation is a way forward for many companies that are becoming software-intensive as it allows for exploring and understanding previously unknown externalities that could in the future become the core value proposition elements.

Moellers [39] utilized system dynamics during different phases of business model innovation. Among the positive results is an improved understanding of how to accommodate a business model from a different context. This is important for software-intensive companies as they often operate in many domains and contexts and thus can reuse the business models between them.

Schaffer et al. [42] highlight understanding complex interactions of the sub-components within dynamic business models and their evolution and important emerging future topics.

To address change on the realization level, i.e., solutions implemented in products, processes, and organizations, literature discuss concepts like business model operationalization (BMO), implying reconfiguration, and tuning of the company's assets depending on the system dynamics [39], business model experimentation [61], [62], collaborative business modeling [5], business model experimentation [29, 35], Dynamic Software Product lines [74], R&D as innovation experiment systems [75], just to name a few. With the advent of the digital business strategy [23], we propose that,

Proposition 3: Software companies possess a unique advantage for detecting and implementing business model change. Using their software development process to integrate their business model innovation with their product innovation, they can efficiently develop "native" product support for managing the linkage of contractual flexibility to the configuration of software products to achieve richer levels of business model experimentation and collaborative business modeling.

Impact on Software Engineering. We postulate that too much effort is dedicated to the creation and extension phases and too little effort on revision and termination. For example, requirements engineering focuses primarily on adding new features rather than reducing the complexity of the product (e.g., feature reduction [76]) or understanding stakeholder inertia and resistance to revolutionary change [77]. We believe that strategic planning of software platforms, e.g., SPL [78] should also include possible revisions or discontinuation of this platform, not forever extensions and growth. Moreover, software products also end their life and get replaced by new products or new businesses [79]. We believe many business model changes should result in ending a product and replacing it with a new one rather than extensively changing or evolving it.

4.4. Increasing business flexibility

Flexibility helps organizations to "*adapt when confronted with new circumstances...and provides the organization with the ability to respond quickly to market forces and uncertainty in the environment.*" [80]. Richter et al. point out that embedding flexibility into system design can optimize stakeholders' incentives, turning incomplete contracts into opportunities [21]. They discuss *changeability* as a term to better understand investments in flexibility related to value, cost, and risk. Changeability is defined by options under internal ("robustness" and "adaptability") respectively external control ("flexibility" and "agility").

In the business and management literature, flexibility is discussed in many different contexts, as related to business models and as ways to manage change, e.g., strategic flexibility [26, 71], resource and organizational flexibility versus dynamic capabilities [81], [82], [83], flexible business models and their architectures [26], dynamic business models in product-service systems [20, 21], linking servitization and business model design [24], and business model flexibility [26, 84].

Chritofi et al. provide a comprehensive literature summary of how agility and flexibility are described in the business literature [31]. They point out several organizational aspects that are relevant for software engineering research and practice, e.g. organizational process alignment, investments in intangible assets, and resource complementarity.

Gul [27] looks at how companies can gain a competitive advantage by executing digital strategies where production and storage are cheaper, deployment is faster, and organizations are collaborative and flexible. Software organizations need to become more collaborative (e.g., work in software ecosystems) and flexible in reusing OSS or previous software components to compete in this new business reality.

Sjödin et al. [28] advocate integrating value--creation and value-capture during value proposition definition, value provision design, and value--in-use delivery. They suggest a process for business model innovation that software-intensive companies can easily apply when designing and experimenting new products with the customers.

Nailler et al. [40] outline six processes by which business models evolve, motivated by the causal mechanism of value anticipation/realization. Gebauer et al. [38] discuss how to increase flexibility by introducing digital servitization.

We define Business Flexibility (BF), as the "*negotiable options in: 1) Relationship; 2) Financial; and 3) the Value proposition between two parties trying to reach an agreement*". These options enable effective negotiation to leverage a company's ability to compromise without breaking the promise in the final contractual agreement. The terms Relationship, Financial, and Value proposition refer to the context of Osterwalder's right side of the BMC [73]. Using the BMC, a company visualizes the strategic decisions and critical business options that characterize the rationale of the business idea and how it strategy-wise will be turned into a successful business (model) realization.

A change (on planning or realization level) is triggered by a gap (misalignment) in expectations and what is delivered. Closing these gaps (transforming a capability into an efficient ability) requires significant investments in time and effort, involving many collaborations. Closing this gap may also require changes in the digital strategy [27], extensive business model experimentation [35], the evolution of the current business model to anticipate more value [40], or a better understanding of the dynamics of current business models [42]. Therefore, we propose the following.

Proposition 4: Software companies have a unique opportunity for implementing business flexibility and efficiently creating value propositions. Software companies should develop software architectures and software functionality to enable a synchronized change in their business model.

Impact on Software Engineering. We believe that the recent development in micro-

-services [85] is a step towards greater flexibility in business model experimentation [29], and a better understanding of the system dynamics [39]. Many software companies offer services instead of products. This means they need to take a large part of the operational cost and also provide frequent updates and new releases. Understanding the product usage data helps to adapt the business models and the offering and therefore optimize the constant operational costs. This helps subscription-based software offerings to stay price competitive. Finally, data-driven experimentation for software products helps to combine value-creation and value-capture during product definition and evolution.

4.5. Supporting dynamic business model change with the help of software tools

Casadesus-Masanell and Ricart argued a clear distinction between strategy and the business model, where the business model "*is a reflection of the firm's realized strategy*" and that the strategy is the plan and process to reach the desired goal via the business model and onto tactics [69]. Strategy refers to the choice of the business model, while tactics refer to the possible realization choices.

Eurich et al. [25] suggest using network thinking as a tool for designing a business model. Dependencies and alternatives are discussed early in this process; this fits very well for software-intensive products as they can be composed of multiple components originating from various sources. Trapp et al. [30] develop a business model innovation identification tool that offers straightforward criteria and indicators to assist practitioners at accelerating BMI in established firms. They tested their tool in four large European corporations.

Bosch suggested a three-layer product model for managing growth and organizes product architecture into a commoditized functionality layer, a differentiating functionality layer, and an innovative and experimental functionality layer [86]. The creation of the business model design alternatives and the analysis of the interdependencies between the business models and the technological capabilities seems to be a promising way forward here [25].

Software tools can provide valuable support in this process by helping to automatically identify criteria and indicators to assist in accelerating business model change [30]. However, the problem remains as most software tools designed to support business modeling focus mainly on business model development rather than evolution [36]. However, most software systems are created once every 10–20 years and later updated, reused, and evolved [87]. This means that the support for this evolution is not covered by most business modeling tools, and the inherited powerful flexibility of software is not considered. Given this long-time perspective understanding strategic agility points [41] and supporting the dynamics aspect of the business models appears to be critical [42].

Proposition 5: Software development tools can provide valuable and mostly automated support for understanding the gap between the capabilities (what software does today) and the planned business model changes and adaptations.

Impact on Software Engineering. Software engineers can use many tools to support collaborative software development and automate many time-consuming tasks. We postulate that a large part of the data generated during the software development process can be used as input for understanding business flexibility and possible business opportunities from the developed software. An example here can be mutation testing that helps to understand the boundaries of software and its limitations beyond the specified or anticipated behavior [88]. Another example could be the data-driven extraction of features from the source code and understanding the offered quality aspects (e.g., security or performance). These aspects can provide valuable indicators for the directions of the business model change, not only for improving engineering activities.

5. Case study: adapting to the digital transformation in the telecommunication industry

For Ericsson AB[2], one critical aspect of achieving the business and technology transformation and managing change has been a long-term focus on industrialization and automation of the product development and the delivery (via process innovation). Digitalization requires additional strategies for handling the fast-paced business environment than driving technology standards. The technology innovation must be in concert with an equally dramatic and accelerating business model innovation. Ericsson's business model has evolved from the resource-centric, standard product-sales model, via several service models, over into different use models, where software-intensive products and services are now sold and delivered as-a-service and on-demand. Today, Ericsson is running multi-business-model operations, and with that, facing additional challenges to keep up with the pace of change. A majority of these challenges can be structured according to Ritter and Lettl's framework [12].

5.1. Business model change at Ericsson

Digitalization shifted the business risks to new dimensions, e.g., business ecosystem (sharing and collaborating in fierce competition), rather than optimizing its assets as a part of a value-delivery chain (e.g., traditionally mitigating risks with long-term business agreements and international standards). Such Business Model Change (BMCh), profoundly impacts the financial steering and control, as much of the investments need to be taken up-front, while the majority of revenues shift to on-demand usage rather than sales of products [20, 21]. The transition from business models based on selling products or hourly-rated services (with a strong focus on add-on sales), into value-based, knowledge-intensive, customer-unique use-models has affected many of Ericsson's dynamic and strategic capabilities and most of the core business processes.

For Ericsson, this also impacted the organizational design, requiring extended focus on organizational learning and incentives, governance, and management structures suited for the inherent dynamics, as well as collaborating with strategic and operational information. It also required enhanced clarity in responsibility and authority for the business model activities.

As a pilot, Ericsson applied the industrializing of the sales and delivery processes in 30+ deliveries to customers in three regions during 2013. These pilot projects delivered contract scoping efficiency and accuracy improvement by 88% . The ordering process was considered simplified, while delivery lead time and project costs were reduced by 12–35%. However, the program complexity and program duration were significantly underestimated (duration exceeded by 150%). We identified three main reasons for the increased complexity:

– the scalability of the piloted solution turned out a bigger issue than anticipated.
– the inherent complexity (flexibility and re-usability) of the services to be industrialized and the services' dependency on the skills and knowledge of the service delivery staff.
– frequent re-organizations – this could be traced back to a substantial business model change together with insufficient support for fast and cross-organizational learning, negatively impacting the transformation program.

The program struggled with two major challenges: 1) to decide what services to industrialize and which should remain "customer-specific", 2) to find the best balance for the new and updated IT tools to minimize disruptions to operations while concurrently updating the business processes.

The technical solution to the first challenge was basically divided into five parts, with a need for completely new tools to be integrated with existing tools and processes.

– Defining the granularity and scope of each service's content (covering all the different products and roles the services are related to).

[2]https://www.ericsson.com/en [last visited 26.06.2021].

- Defining the structure and content of the service catalog and the structure and representation of a service.
- The IT tools needed to plan, develop and deploy a service (so it's available in the product and service catalog ready for marketing, ordering, and sales).
- The IT tools needed in a delivery project to sell, order, plan, and deliver instances of services, plus the benchmark of projects and the outcome of each individual service and it's delivered instances.
- Non-industrialized services were managed manually with little or no automation.

The challenge of the dependency of the industrialized services from the skills and knowledge of the service delivery staff proved to be complex mainly due to the volume of implicit and explicit information in various forms of knowledge representations, and realizations with efficient knowledge management systems.

The decisions between investing in tool support versus investing in business process flexibility turned out to be very challenging for decision-makers and top management. As a consequence, the "traditional" IT update and integration process of new and existing tools to match the evolving business processes was affected by misunderstandings and delays leading to temporary solutions in the sales and delivery organization. Under customer pressure to deliver on signed contracts, this led to decreased trust between organizations, affecting the efficiency of the collaboration.

It also proved difficult to synchronize the business process development (sales and delivery processes to use industrialized services) with the agile Ericsson product development (the new generation of products to be delivered using the updated business processes). We identified the following four root causes of the misalignment:

- temporal effects due to different life cycles of these two core business processes,
- organizational steering, coordination, and incentives,
- expected capabilities that did not deliver on the requested abilities in customer projects, and
- the differences between the old and new product generations, the needed training of the service delivery staff, and their valuable customer experience feedback to the R&D organization.

5.1.1. Temporal effects of organizational learning

The temporal effects of organizational learning created a gap between different organizations (R&D, sales, delivery, and Ericsson IT) were occupied with their life-cycles of change as committed in earlier plans, see [11, Figure 5]. The symptoms of this were observed in areas of communication, coordination, training, and reporting, resulting in uncertainty, equivocality, and sub-optimization at best, and a lack of abilities at worst.

Scaling the solution was affected since planned capabilities needed by different organizations were not translated (in time) into required abilities, i.e., integrated tools and staff adequately trained in relation to the new or changed business processes (so they could perform the tasks demanded by the evolving business model). The scale of the industrialization problem was among the most significant factors since it affected the amount of information and the relationships between the affected organizations involved in the change processes. The rippling change-reaction escalated and started to violate existing goals, commitment, and reporting, leading to more efforts spent on temporary, local solutions to assure customer contracts could be honored.

5.2. Case study results summary and synthesis

Ericsson's traditional, engineering-centered industrialization approach would have benefited by categorizing the strategic program's requirements and associated risks into the five areas (strategic decisions, business logic, business model artifacts, misalignment, and BMa) and highlighting that the program was actually facing a business model change. By addressing the misalignment between the effectiveness ("do the right thing" as a top-down strategic planning process) and the efficiency (as the bottom-up change of existing business models, business processes, organizations, and tools), we believe the scale of the program, as well as the temporal effects, could have been predicted and managed in a better way by proposing a set of different tactics, thereby invoking a higher degree of top management commitment and attention.

This study confirms opportunities and challenges for digitalization reported by scholars, for example, [6, 21, 24, 52]. Our interviews revealed that in practice, the scalability, the complexity of roles and (changing) business intents, and the size of the solution were perceived as the most significant challenges. Given the global, wide scope of the program and frequent organizational changes, establishing a reporting structure for how the different tactics supported each other (and executed by the different parts of the organization) , turned out to be slow and inefficient, causing mistrust and unnecessary tensions. We believe this program would have benefited from a BMCh-centered approach, rather than a engineering-focused servitization approach, by achieving over-arching clarity and consensus between top-management, middle management, and the affected organizations, highlighting it was not just "business as usual".

The case study also highlights the added complexity of BMCh for large software companies that operate with contracts spawning years to complete. This calls for a combination of BMCh and organizational design. What appears to be inevitable is that the business environment will change during the execution of the underlying agreements. Our interview respondents believed that governance mechanisms should facilitate the exploration phase (Knowledge Creation process), transforming tacit knowledge into explicit knowledge fast enough and made it available through the Knowledge Management process.

We believe it requires fast, efficient feedback loops between R&D, sales, and service delivery organizations, illustrating the continuous interaction between knowledge creation and knowledge management processes. Support for these loops should preferably be implemented both in the products as well as in the business processes. We, therefore, propose that,

Proposition 6: The practice of Digital Business Modeling (DBM) should be coined as a fusion between current practices of business modeling and requirement engineering, and become a key practice in facilitating business model innovation through experimentation.

6. Conclusion

Many distinguished scholars have highlighted the cross-disciplinary complexity stemming from the ongoing digitalization and transformation of the business environment [3, 13, 14, 89], just to name a few. This paper highlights three critical aspects of business modeling in the analysis of the misalignment between planning and execution. Firstly, focus on experimenting [64] as a "round-trip" process of "translating an idea into execution, test, evaluate, and change until satisfied" (similar to the agile method of developing software products). Secondly, focus on transactions, thereby connecting the business model to human behavior and value in execution and planning activities. Thirdly, the analysis is direction-sensitive, with minimum two (role-dependent) views of the transaction, implying an information representation suitable for maintaining (observe, analyze, decide, change) many relationships (through all the stages of the business model lifecycle) [11]. FInally, we analyze how software engineering methods and tools can support business model flexibility and promptly realizing business model changes.

This paper is an initial step for such a detailed, cross-disciplinary guide for handing business model change. Synthesizing from the two previous two literature reviews [9, 10], a design science study [11], and the case study presented in this paper, we present six propositions for addressing the challenges of aligning the planning and execution layers for software-intensive product development. We also highlight four critical aspects that software-intensive companies need to address:

- Business model innovation for the business ecosystem, e.g., driven by markets and contextual changes, co-creation of value, collaboration within and between organizations, partners, communities, and customers, new streams of revenue while sharing of risks, revenues, and costs [5, 64].
- Software tools that focus on automation and integration of business and software architecture. These tools should support the shared economy aspect of new business models and the service-driven economy [90–92].
- Organizations prepared for experimentation and collaboration in a digital business world, affecting both the product development as well as the value delivery, e.g., agreement structures, incentives, processes, knowledge management and organizational learning, measurements of effectiveness and efficiency, revenues, cost, decision-making based on multifaceted optimization and transparency [93, 94].
- The level of integration and automation between the four processes of value creation, value capture, knowledge creation, and knowledge management [95, 96]. This is the foundation for an innovative enterprise and should be nurtured as a key competitive advantage.

Acknowledgment

We would like to acknowledge that this work was supported by the KKS foundation through the S.E.R.T. Research Profile project at Blekinge Institute of Technology.

References

[1] P. Parviainen, M. Tihinen, J. Kääriäinen, and S. Teppola, "Tackling the digitalization challenge: how to benefit from digitalization in practice," *International Journal of Information Systems and Project Management*, Vol. 5, No. 1, 2017, pp. 63–77.

[2] S. Gupta, L. Ulanova, S. Bhardwaj, P. Dmitriev, P. Raff, and A. Fabijan, "The anatomy of a large--scale experimentation platform," in *International Conference on Software Architecture (ICSA)*. IEEE, 2018, pp. 1–109.

[3] C. Legner, T. Eymann, T. Hess, C. Matt, T. Böhmann, P. Drews, A. Mädche, N. Urbach, and F. Ahlemann, "Digitalization: Opportunity and challenge for the business and information systems engineering community," *Business and Information Systems Engineering*, Vol. 59, 2017, pp. 301–308.

[4] J. Moore, "The rise of a new corporate form," *Washington Quarterly*, Vol. 21, 1998, pp. 167–181.

[5] D. Romero and A. Molina, "Collaborative networked organisations and customer communities: value co-creation and co-innovation in the networking era," *Production Planning and Control: The Management of Operations*, Vol. 22, No. 5–6, 2011, pp. 447–472.

[6] Y. Snihur and J. Tarzijan, "Managing complexity in a multi-business-model organization," *Long Range Planning*, Vol. 51, No. 1, 2017, pp. 50–63.

[7] S.A. Cavalcante, "Preparing for business model change: The 'pre-stage' finding," *Journal of Management and Governance*, Vol. 18, 2014, pp. 449–469.

[8] N.F. Höflinger, "The business model concept and its antecedents and consequences – Towards a common understanding," *Academy of Management Proceedings: Organization Development and Change*, Vol. 2014:1, 2014.

[9] M. Wilson, K. Wnuk, J. Silvander, and T. Gorschek, "A literature review on the effectiveness and efficiency of business modeling," *e-Informatica Software Engineering Journal*, Vol. 12, No. 1, 2018, pp. 265–302. [Online]. http://www.e-informatyka.pl/attach/e-Informatica_-_Volume_12/eInformatica2018Art11.pdf

[10] M. Wilson and K. Wnuk, "Business modeling and flexibility in software-intensive product development – a systematic literature review," in *Challenges and Opportunities in the Digital Era*. Springer International Publishing, 2018, pp. 292–304.

[11] J. Silvander, M. Wilson, K. Wnuk, and M. Svahnberg, "Supporting continuous changes to business intents," *International Journal of Software Engineering and Knowledge Engineering*, Vol. 27, 2017, pp. 1167–1198.

[12] T. Ritter and C. Lettl, "The wider implications of business-model research," *Long Range Planning*, 2017, pp. 1–8.

[13] C. Zott and R. Amit, "The business model: A theoretically anchored robust construct for strategic analysis," *Strategic Organization*, Vol. 11, No. 4, 2013, pp. 403–411.

[14] D. Veit, E. Clemons, A. Benlian, P. Buxmann, T. Hess, D. Kundisch, J.M. Leimeister, P. Loos, and M. Spann, "Business models: An information systems research agenda," *Business and Information Systems Engineering*, Vol. 6, 2014, pp. 45–53.

[15] O.D. Fjeldstad and C.C. Snow, "Business models and organization design," *Long Range Planning*, Vol. 51, No. 1, 2018, pp. 32–39.

[16] C. Matt, T. Hess, and A. Benlian, "Digital transformation strategies," *Business and Information Systems Engineering*, Vol. 57, 2015, pp. 339–343.

[17] P.E. Eriksson, P.C. Patel, D.R. Sjödin, J. Frishammar, and V. Parida, "Managing interorganizational innovation projects: Mitigating the negative effects of equivocality through knowledge search strategies," *Long Range Planning*, Vol. 49, No. 6, 2016, pp. 691–705.

[18] A. Osterwalder, Y. Pigneur, and C. Tucci, "Clarifying business models: Origins, present, and future of the concept," *Communications of the Association for Information Systems*, Vol. 15, No. 1, 2005, pp. 1–25.

[19] P. Ballon, "Business modelling revisited: the configuration of control and value," *Info*, Vol. 9, No. 5, 2007, pp. 6–19.

[20] H. Meier and M. Boßlau, "Design and engineering of dynamic business models for industrial product-service systems," in *The Philosophers Stone for Sustainability*. Springer, 2013, pp. 179–184.

[21] A. Richter, T. Sadek, and M. Steven, "Flexibility in industrial product-service systems and use-oriented business models," *CIRP Journal of Manufacturing Science and Technology*, Vol. 3, No. 2, 2010, pp. 128–134.

[22] C. Wohlin, "Guidelines for snowballing in systematic literature studies and a replication in software engineering," in *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, 2014, pp. 1–10.

[23] C.J. Woodard, N. Ramasubbu, F.T. Tschang, and V. Sambamurthy, "Design capital and design moves: The logic of digital business strategy," *MIS Quarterly: Management Information Systems*, Vol. 37, No. 2, 2013, pp. 537–564.

[24] E.K. Chew, "Linking a service innovation-based framework to business model design," in *16th Conference on Business Informatics*, Vol. 1. IEEE, 2014, pp. 191–198.

[25] M. Eurich, T. Weiblen, and P. Breitenmoser, "A six-step approach to business model innovation," *International Journal of Entrepreneurship and Innovation Management*, Vol. 18, No. 4, 2014, pp. 330–348.

[26] K. Mason and S. Mouzas, "Flexible business models," *European Journal of Marketing*, Vol. 46, No. 10, 2012, pp. 1340–1367.

[27] M. Gul, "Digital business strategies and competitive superiority," *International Journal of Business Ecosystem*, Vol. 2, No. 1, Feb. 2020, p. 17–31. [Online]. https://www.bussecon.com/ojs/index.php/ijbes/article/view/106

[28] D. Sjödin, V. Parida, M. Jovanovic, and I. Visnjic, "Value creation and value capture alignment in business model innovation: A process view on outcome-based business models," *Journal of Product Innovation Management*, Vol. 37, No. 2, 2020, pp. 158–183.

[29] M. Antikainen, A. Aminoff, and J. Heikkilä, "Business model experimentations in advancing b2b sharing economy research," in *ISPIM Innovation Symposium*. The International Society for Professional Innovation Management (ISPIM), 2018, pp. 1–12.

[30] M. Trapp, K.I. Voigt, and A. Brem, "Business models for corporate innovation management: Introduction of a business model innovation tool for established firms," *International Journal of Innovation Management*, Vol. 22, No. 01, 2018, pp. 18–40.

[31] M. Christofi, V. Pereira, D. Vrontis, S. Tarba, and A. Thrassou, "Agility and flexibility in international business research: A comprehensive review and future research directions," *Journal of World Business*, Vol. 56, No. 3, 2021, p. 101194. [Online]. https://www.sciencedirect.com/science/article/pii/S1090951621000067

[32] D.J. Teece, "Business models and dynamic capabilities," *Long Range Planning*, Vol. 51, No. 1, 2018, pp. 40–49. [Online]. https://www.sciencedirect.com/science/article/pii/S0024630117302868

[33] L. Linde, J. Frishammar, and V. Parida, "Revenue models for digital servitization: A value capture framework for designing, developing, and scaling digital services," *IEEE Transactions on Engineering Management*, Vol. 5, No. 22, 2021, pp. 1–16.

[34] F. Hacklin, J. Björkdahl, and M.W. Wallin, "Strategies for business model innovation: How firms reel in migrating value," *Long Range Planning*, Vol. 51, No. 1, 2018, pp. 82–110. [Online]. https://www.sciencedirect.com/science/article/pii/S0024630117302881

[35] F. Vendrell-Herrero, G. Parry, M. Opazo-Basáez, and F.J. Sanchez-Montesinos, "Does business

model experimentation in dynamic contexts enhance value capture?" *International Journal of Business Environment*, Vol. 10, No. 1, 2018, pp. 14–34.

[36] D. Szopinski, T. Schoormann, T. John, R. Knackstedt, and D. Kundisch, "Software tools for business model innovation: Current state and future challenges," *Electronic Markets*, Vol. 30, No. 3, 2020, pp. 469–494.

[37] B.W. Wirtz, *Adaptation and Modification of Business Models*. Cham: Springer International Publishing, 2020, pp. 227–238.

[38] H. Gebauer, M. Paiola, N. Saccani, and M. Rapaccini, "Digital servitization: Crossing the perspectives of digitization and servitization," *Industrial Marketing Management*, Vol. 93, 2021, pp. 382–388. [Online]. https://www.sciencedirect.com/science/article/pii/S0019850120304855

[39] T. Moellers, L. von der Burg, B. Bansemir, M. Pretzl, and O. Gassmann, "System dynamics for corporate business model innovation," *Electronic Markets*, Vol. 29, No. 3, 2019, pp. 387–406.

[40] C. Nailer and G. Buttriss, "Processes of business model evolution through the mechanism of anticipation and realisation of value," *Industrial Marketing Management*, Vol. 91, 2020, pp. 671–685. [Online]. https://www.sciencedirect.com/science/article/pii/S0019850116302735

[41] T. Clauss, M. Abebe, C. Tangpong, and M. Hock, "Strategic agility, business model innovation, and firm performance: An empirical investigation," *IEEE Transactions on Engineering Management*, 2019, pp. 1–18.

[42] N. Schaffer, M. Pfaff, and H. Krcmar, "Dynamic business models: A comprehensive classification of literature," in *Thirteenth Mediterranean Conference on Information Systems (MCIS 2019)*, 2019.

[43] N. Pratama and J. Iijima, "Value operation: Linking value in new business model creation process," in *23rd Pacific Asia Conference on Information Systems: Secure ICT Platform for the 4th Industrial Revolution, PACIS*, 2019, pp. 12–32.

[44] D.S. Cruzes and T. Dyba, "Recommended steps for thematic synthesis in software engineering," in *2011 International Symposium on Empirical Software Engineering and Measurement*, 2011, pp. 275–284.

[45] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, Vol. 14, 2009, pp. 131–164.

[46] R.K. Yin, *Applications of case study research*. Sage Publications, 2011.

[47] F. Adrodegari and N. Saccani, "A maturity model for the servitization of product-centric companies," *Journal of Manufacturing Technology Management*, 2020.

[48] T.C. Lethbridge, S.E. Sim, and J. Singer, "Studying software engineers: Data collection techniques for software field studies," *Empirical software engineering*, Vol. 10, No. 3, 2005, pp. 311–341.

[49] B. Flyvbjerg, "Five misunderstandings about case-study research," *Qualitative inquiry*, Vol. 12, No. 2, 2006, pp. 219–245.

[50] W. Reim, V. Parida, and D. Örtqvist, "Strategy, business models or tactics – What is product-service systems (PSS) literature talking about?" in *Proceedings of the International Conference on Engineering Design, ICED*, 2013, pp. 309–318.

[51] W. Reim, V. Parida, and D. Örtqvist, "Product-service systems (PSS) business models and tactics – a systematic literature review," *Journal of Cleaner Production*, 2014.

[52] H. Meier, R. Roy, and G. Seliger, "Industrial Product-Service systems – IPS2," *CIRP Annals – Manufacturing Technology*, Vol. 59, No. 2, 2010, pp. 607–627.

[53] D. Kindström, "Towards a service-based business model – Key aspects for future competitive advantage," *European Management Journal*, Vol. 28, No. 6, 2010, pp. 479–490.

[54] A. Zolnowski and T. Böhmann, "Business modeling for services: Current state and research perspectives," in *AMCIS 2011 Proceedings*, 2011, pp. 1–8.

[55] A. Bharadwaj, O.A. El Sawey, P.A. Pavlou, N. Venkatraman, O.a. El Sawy, P.A. Pavlou, and N. Venkatraman, "Digital business strategy: Toward a next generation of insights," *MIS Quarterly*, Vol. 37, No. 2, 2013, pp. 471–482.

[56] H. Munir, K. Wnuk, and P. Runeson, "Open innovation in software engineering: a systematic mapping study," *Empirical Software Engineering*, Vol. 21, No. 2, 2016, pp. 684–723.

[57] X. Koufteros, M. Vonderembse, and J. Jayaram, "Internal and external integration for product development: The contingency effects of uncertainty, equivocality, and platform strategy," *Decision sciences*, Vol. 36, No. 1, 2005, pp. 97–133.

[58] A. Chang and C.C. Tien, "Quantifying uncertainty and equivocality in engineering projects," *Construction Management and Economics*, Vol. 24, 2006, pp. 171–184.

[59] B.W. Boehm, "Software risk management: principles and practices," *IEEE Software*, Vol. 8, No. 1, 1991, pp. 32–41.

[60] M. Svahnberg, T. Gorschek, T.T.L. Nguyen, and M. Nguyen, "Uni-repm: validated and improved," *Requirements Engineering*, Vol. 18, No. 1, 2013, pp. 85–103.

[61] H. Chesbrough, "Business Model Innovation: Opportunities and Barriers," *Long Range Planning*, Vol. 43, No. 2–3, 2010, pp. 354–363.

[62] R.G. McGrath, "Business models: A discovery driven approach," *Long Range Planning*, Vol. 43, No. 2–3, apr 2010, pp. 247–261.

[63] R. Kohavi, D. Tang, and Y. Xu, *Trustworthy online controlled experiments: A practical guide to a/b testing*. Cambridge University Press, 2020.

[64] R. Rohrbeck, L. Konnertz, and S. Knab, "Collaborative business modelling for systemic and sustainability innovations," *International Journal of Technology Management*, Vol. 63, No. 1/2, 2013, p. 4.

[65] OMG, *Value Delivery Modeling Language Specification, v1.0*. Object Management Group, OMG.org, 2015, [Online; accessed 2018-08-08] http://www.omg.org/spec/VDML/About-VDML/.

[66] F. Fagerholm, A. Sanchez Guinea, H. Mäenpää, and J. Münch, "The right model for continuous experimentation," *Journal of Systems and Software*, Vol. 123, 2017, pp. 292–305.

[67] A. Fabijan, P. Dmitriev, H.H. Olsson, and J. Bosch, "The evolution of continuous experimentation in software product development: From data to a data-driven organization at scale," in *39th International Conference on Software Engineering (ICSE)*, 2017, pp. 770–780.

[68] S. Cavalcante, P. Kesting, and J. Ulhøi, "Business model dynamics and innovation: (re)establishing the missing linkages," *Management Decision*, Vol. 49, 2011, pp. 1327–1342.

[69] R. Casadesus-Masanell and J.E. Ricart, "From Strategy to Business Models and onto Tactics," *Long Range Planning*, Vol. 43, No. 2–3, 2010, pp. 195–215.

[70] Y.L. Doz and M. Kosonen, "Embedding strategic agility: A leadership agenda for accelerating business model renewal," *Long Range Planning*, Vol. 43, No. 2–3, 2010, pp. 370–382.

[71] S. Schneider and P.A.T. Spieth, "Business model innovation and strategic flexibility: insights from an experimental research design," *International Journal of Innovation Management*, Vol. 18, No. 6, 2014, pp. 1–22.

[72] A. Ghezzi, "Revisiting business strategy under discontinuity," *Management Decision*, Vol. 51, No. 7, 2013, pp. 1326–1358.

[73] A. Osterwalder and Y. Pigneur, *Business model generation: A handbook for visionaries, game changers, and challengers*. Hoboken, NJ: Wiley, 2010.

[74] R. Capilla, J. Bosch, P. Trinidad, A. Ruiz-Cortés, and M. Hinchey, "An overview of dynamic software product line architectures and techniques: Observations from research and industry," *Journal of Systems and Software*, Vol. 91, 2014, pp. 3–23.

[75] H. Holmström Olsson, J. Bosch, and H. Alahyari, "Towards R&D as innovation experiment systems: A framework for moving beyond agile software development," in *Proceedings of the IASTED International Conference on Software Engineering*, 2013, pp. 798–805.

[76] S. Marciuska, C. Gencel, X. Wang, and P. Abrahamsson, "Feature usage diagram for feature reduction," in *International Conference on Agile Software Development*. Springer, 2013, pp. 223–237.

[77] K. Wnuk, R.B. Svensson, and D. Callele, "The effect of stakeholder inertia on product line requirements," in *Second IEEE International Workshop on Requirements Engineering for Systems, Services, and Systems-of-Systems (RESS)*. IEEE, 2012, pp. 34–37.

[78] K. Pohl, G. Böckle, and F.J. van Der Linden, *Software product line engineering: foundations, principles and techniques*. Springer Science and Business Media, 2005.

[79] S. Jansen, K.M. Popp, and P. Buxmann, "The sun also sets: Ending the life of a software product," in *Software Business*, B. Regnell, I. van de Weerd, and O. De Troyer, Eds. Berlin, Heidelberg: Springer, 2011, pp. 154–167.

[80] H.C. Lucas and M. Olson, "The impact of information technology on organizational flexibility," *Journal of Organizational Computing*, Vol. 4, 1994, pp. 155–176.

[81] J. Barney, "Firm resources and sustained competitive advantage," *Journal of management*, Vol. 17, No. 1, 1991, pp. 99–120.

[82] R. Sanchez and J.T. Mahoney, "Modularity, flexibility, and knowledge management in product and organization design," *Strategic Management Journal*, Vol. 17, 1996, pp. 63–76.

[83] D.J. Teece, G. Pisano, and A. Shuen, "Dynamic capabilities and strategic management," *Strategic Management Journal*, Vol. 18, 1997, pp. 509–533.

[84] K.J.K. Mason and S. Leek, "Learning to Build a Supply Network: An Exploration of Dynamic

Business Models," *Journal of Management Studies*, Vol. 45, No. 4, 2008, pp. 774–799.

[85] D. Namiot and M. Sneps-Sneppe, "On micro-services architecture," *International Journal of Open Information Technologies*, Vol. 2, No. 9, 2014, pp. 24–27.

[86] J. Bosch, "Achieving simplicity with the three--layer product model," *Computer*, Vol. 46, No. 11, Nov. 2013, pp. 34–39.

[87] J.C. Munson, "Software lives too long," *IEEE Software*, Vol. 15, No. 4, Jul. 1998, pp. 18, 20.

[88] Y. Jia and M. Harman, "An analysis and survey of the development of mutation testing," *IEEE Transactions on Software Engineering*, Vol. 37, No. 5, 2010, pp. 649–678.

[89] D. Romero and F. Vernadat, "Enterprise information systems state of the art: Past, present and future trends," *Computers in Industry*, Vol. 79, 2016, pp. 3–13.

[90] D. Olausson and C. Berggren, "Managing uncertain, complex product development in high-tech firms: In search of controlled flexibility," *R&D Management*, Vol. 15, 2010, p. 383–399.

[91] R. Capilla, J. Bosch, and K.C. Kang, *Systems and Software Variability Management Concepts, Tools and Experiences*. Berlin, Heidelberg: Springer, 2013.

[92] T. Magnusson and N. Lakemond, "Evolving schemes of interpretation: investigating the dual role of architectures in new product development," *R&D Management*, Vol. 47, 2017, pp. 36–46.

[93] F. Laloux and K. Wilber, *Reinventing Organizations*. Laloux, Frederic, 2014. [Online]. http://www.reinventingorganizations.com/pay-what-feels-right.html

[94] D. Kahneman, *Thinking, Fast and Slow*. Farrar, Straus and Giroux, 2011.

[95] D.P. Lepak, K.G. Smith, and M.S. Taylor, "Introduction to special topic forum value creation and value capture: A multilevel perspective," *Academy of Management Review*, Vol. 32, 2007, pp. 180–194.

[96] C. Curado, "Organisational learning and organisational design," *The Learning Organization*, Vol. 13, No. 1, 2006, pp. 25–48.

e-Informatica Software Engineering Journal (eISEJ) is an international, open access, no authorship fees, blind peer-reviewed journal that concerns theoretical and practical issues pertaining development of software systems. Our aim is to focus on experimentation and machine learning in software engineering.

The journal is published under the auspices of the Software Engineering Section of the Committee on Informatics of the Polish Academy of Sciences and Wrocław University of Science and Technology.

Aims and Scope:

The purpose of **e-Informatica Software Engineering Journal** is to publish original and significant results in all areas of software engineering research.

The scope of **e-Informatica Software Engineering Journal** includes methodologies, practices, architectures, technologies and tools used in processes along the software development lifecycle, but particular stress is laid on empirical evaluation.

e-Informatica Software Engineering Journal is published online and in hard copy form. The on-line version is from the beginning published as a gratis, no authorship fees, open access journal, which means it is available at no charge to the public. The printed version of the journal is the primary (reference) one.

Topics of interest include, but are not restricted to:

— Software requirements engineering and modeling
— Software architectures and design
— Software components and reuse
— Software testing, analysis and verification
— Agile software development methodologies and practices
— Model driven development
— Software quality
— Software measurement and metrics
— Reverse engineering and software maintenance
— Empirical and experimental studies in software engineering (incl. replications)
— Evidence based software engineering
— Systematic reviews and mapping studies
— Meta-analyses
— Object-oriented software development
— Aspect-oriented software development
— Software tools, containers, frameworks and development environments
— Formal methods in software engineering.
— Internet software systems development
— Dependability of software systems
— Human-computer interaction
— AI and knowledge based software engineering
— Data mining in software engineering
— Prediction models in software engineering
— Mining software repositories
— Search-based software engineering
— Multiobjective evolutionary algorithms
— Tools for software researchers or practitioners
— Project management
— Software products and process improvement and measurement programs
— Process maturity models

Important information: Papers can be rejected administratively without undergoing review for a variety reasons, such as being out of scope, being badly presented to such an extent as to prevent review, missing some fundamental components of research such as the articulation of a research problem, a clear statement of the contribution and research methods via a **structured abstract** or the evaluation of the proposed solution (empirical evaluation is strongly suggested).

Funding acknowledgements: Authors are requested to identify who provided financial support for the conduct of the research and/or preparation of the article and to briefly describe the role of the sponsor(s), if any, in study design; in the collection, analysis and interpretation of data; in the writing of the paper. If the funding source(s) had no such involvement then this should be stated as well.

The submissions will be accepted for publication on the base of positive reviews done by international Editorial Board (`https://www.e-informatyka.pl/index.php/einformatica/editorial-board/`) and external reviewers. English is the only accepted publication language. To submit an article please enter our online paper submission site (`https://mc.manuscriptcentral.com/e-InformaticaSEJ`).

Subsequent issues of the journal will appear continuously according to the reviewed and accepted submissions.

http://www.e-informatyka.pl/

e-Informatica

ISSN 1897-7979