e-Informatica

software engineering journal

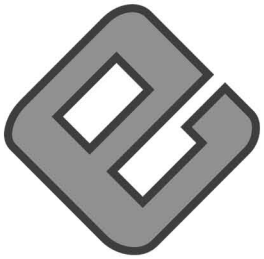
2016 | volume 10 | issue 1

e-Informatica

e-Informatica

software engineering journal

2016 volume 10 issue 1



e-Informatica

Wrocław University
of Science and Technology

e-Informatica Software Engineering Journal
www.e-informatyka.pl, DOI: 10.5277/e-informatica

Editorial Office Manager: Wojciech Thomas

Proofreader: Anna Tyszkiewicz

Typeset by Wojciech Myszka with the LaTeX 2_ε 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

Printed by beta-druk, www.betadruk.pl

Editorial Board

Contents

Editorial 7
ABC-CAG: Covering Array Generator for Pair-wise Testing Using Artificial Bee Colony Algorithm
Priti Bansal, Sangeeta Sabharwal 9
Reducing the Number of Higher-Order Mutants with the Aid of Data Flow
Ahmed S. Ghiduk 31
Automatic SUMO to UML translation
Bogumiła Hnatkowska 51
Highly Automated Agile Testing Process: An Industrial Case Study
Jarosław Berłowski, Patryk Chruściel, Marcin Kasprzyk, Iwona Konaniec, Marian Jureczko . 69
Software Startups – A Research Agenda
Michael Unterkalmsteiner, Pekka Abrahamsson, XiaoFeng Wang, Anh Nguyen-Duc, Syed Shah, Sohaib Shahid Bajwa, Guido H. Baltes, Kieran Conboy, Eoin Cullina, Denis Dennehy, Henry Edison, Carlos Fernandez-Sanchez, Juan Garbajosa, Tony Gorschek, Eriks Klotins, Laura Hokkanen, Fabio Kon, Ilaria Lunesu, Michele Marchesi, Lorraine Morgan, Markku Oivo, Christoph Selig, Pertti Seppänen, Roger Sweetman, Pasi Tyrväinen, Christina Ungerer, Agustin Yagüe 89

Editorial

Following the mission of e-Informatica Software Engineering Journal, we would like to present the 10th volume containing papers referring to testing, domain modelling and startup software companies.

The first one by Bansal et al. [1] concentrates on generating test cases to uncover faults caused by the interaction of input parameters. An artificial intelligence algorithm based on a bee colony was elaborated. It reduces the exponential growth of the number of test cases. The conducted experiments have shown that the proposed approach gives better or similar results in comparison to the existing state-of-the-art algorithms.

A similar problem of overcoming the exponential explosion in the number of higher-order mutants is considered in the second paper by Ghiduk [3]. The basic idea is to utilize a data-flow analysis to select points to seed mutation through the program under test. A set of experiments showed that the proposed technique is more effective than the earlier techniques in generating higher-order mutants without affecting the efficiency of mutation testing.

In the third paper by Hnatkowska [4], a programming tool extracting some knowledge from SUMO-like ontologies and transforming it into the UML class diagram is presented. The usage of the tool in the context of software modelling, especially in domain model construction, is considered.

The problem of testing is considered again in the fourth paper. A highly automated agile testing process is presented by Berłowski et al. [2]. The authors use their industrial experience in a medium size software project developed using Scrum. The main result of the paper is a set of recommendations related to the testing process taking into account the employed principles of agility, specifically: continuous integration, responding to change, test automation and test-driven development. Additionally, an efficient testing environment that combines some testing frameworks with custom-developed simulators is presented.

The last, fifth paper, written by a group of 28 authors from 7 countries, [5] is very special. Software engineering as scientific discipline suggests or recommends a set of rules, good practices, and methodologies for rational and efficient software development. How to apply these suggestions and recommendations in practice, especially in forming software companies? How to establish software startups? These are examples of the main questions stated in the paper. There are no final answers to these questions, but there is a systematic and rational review of some problems that should be considered and solved on the way to a software startup. Software startups are quite distinct from traditional mature software companies, but also from micro-, small-, and medium-sized enterprises, introducing new challenges relevant for software engineering research. The considerations take into account important human aspects and constraints imposed by the modern economy in the societies of today.

As Editors of the volume, we would like to thank all of the authors as well as reviewers, for their efforts. e-Infomatica Software Engineering Journal is now indexed, among others, by the Web of Science™ Core Collection (Emerging Sources Citation Index), Scopus, DBLP, DOAJ, and Google Scholar. We look forward to receiving high quality contributions from researchers and practitioners in software engineering for the next issue of the journal.

Editors
Zbigniew Huzar
Lech Madeyski

References

[1] Priti Bansal, Sangeeta Sabharwal, Nitish Mittal, and Sarthak Arora. ABC-CAG: Covering Array Generator for Pair-wise Testing Using Artificial Bee Colony Algorithm. *e-Informatica Software Engineering Journal*, 10(1):9–29, 2016. doi:10.5277/e-Inf160101.

[2] Jarosław Berłowski, Patryk Chruściel, Marcin Kasprzyk, Iwona Konaniec, and Marian Ju-

reczko. Highly Automated Agile Testing Process: An Industrial Case Study. *e-Informatica Software Engineering Journal*, 10(1):69–87, 2016. doi:10.5277/e-Inf160104.

[3] Ahmed S. Ghiduk. Reducing the Number of Higher-order Mutants with the Aid of Data Flow. *e-Informatica Software Engineering Journal*, 10(1):31–49, 2016. doi:10.5277/e-Inf160102.

[4] Bogumiła Hnatkowska. Automatic SUMO to UML translation. *e-Informatica Software Engineering Journal*, 10(1):51–67, 2016. doi:10.5277/e-Inf160103.

[5] Michael Unterkalmsteiner, Pekka Abrahamsson, XiaoFeng Wang, Anh Nguyen-Duc, Syed Shah, Sohaib Shahid Bajwa, Guido H. Baltes, Kieran Conboy, Eoin Cullina, Denis Dennehy, Henry Edison, Carlos Fernandez-Sanchez, Juan Garbajosa, Tony Gorschek, Eriks Klotins, Laura Hokkanen, Fabio Kon, Ilaria Lunesu, Michele Marchesi, Lorraine Morgan, Markku Oivo, Christoph Selig, Pertti Seppänen, Roger Sweetman, Pasi Tyrväinen, Christina Ungerer, and Agustin Yagüe. Software Startups – A Research Agenda. *e-Informatica Software Engineering Journal*, 10(1):89–124, 2016. doi:10.5277/e-Inf160105.

e-Informatica Software Engineering Journal, Volume 10, Issue 1, 2016, pages: 9–29, DOI 10.5277/e-Inf160101

ABC-CAG: Covering Array Generator for Pair-wise Testing Using Artificial Bee Colony Algorithm

Priti Bansal[a], Sangeeta Sabharwal[a], Nitish Mittal[a], Sarthak Arora[b]
[a] *Netaji Subhas Institute of Technology, University of Delhi*
[b] *School of Computer Science and Engineering, Vellore Institute of Technology, Tamil Nadu*
bansalpriti79@gmail.com, ssab63@gmail.com, nitishmittal94@gmail.com, sarthak10193@gmail.com

Abstract

Testing is an indispensable part of the software development life cycle. It is performed to improve the performance, quality and reliability of the software. Various types of testing such as functional testing and structural testing are performed on software to uncover the faults caused by an incorrect code, interaction of input parameters, etc. One of the major factors in deciding the quality of testing is the design of relevant test cases which is crucial for the success of testing. In this paper we concentrate on generating test cases to uncover faults caused by the interaction of input parameters. It is advisable to perform thorough testing but the number of test cases grows exponentially with the increase in the number of input parameters, which makes exhaustive testing of interaction of input parameters imprudent. An alternative to exhaustive testing is combinatorial interaction testing (CIT) which requires that every t-way interaction of input parameters be covered by at least one test case. Here, we present a novel strategy ABC-CAG (Artificial Bee Colony-Covering Array Generator) based on the Artificial Bee Colony (ABC) algorithm to generate covering an array and a mixed covering array for pair-wise testing. The proposed ABC-CAG strategy is implemented in a tool and experiments are conducted on various benchmark problems to evaluate the efficacy of the proposed approach. Experimental results show that ABC-CAG generates better/comparable results as compared to the existing state-of-the-art algorithms.

Keywords: combinatorial interaction testing, pair-wise testing, covering array, artificial bee colony

1. Introduction

Testing plays a critical role in the software development life cycle (SDLC). It helps to improve the performance of the software system and ensure the delivery of quality and a reliable system. Often more than 50% of the entire software development resources are allocated to testing [1]. As the complexity of the software system increases so does the cost of testing, therefore testing the software effectively within a reasonable time and budget continues to be a challenge for the software testing community. One of the major factors in determining the quality of testing is the design of relevant test cases. Various types of testing techniques such as white-box testing and black-box testing are performed to detect faults in the software system. In black-box testing, test cases are generated from the specification of the system under test (SUT), whereas in the case of white-box testing, they are determined from the internal structure. However, in both cases the primary focus of the software testing community is to design a set of optimal test cases to uncover maximum faults in the software system within a reasonable time.

In a complex software system it has been found that the interaction of input parameters may cause interaction errors and to uncover these interaction errors, it is necessary to generate test cases that test all possible combinations of in-

put parameters. A software system with m input parameters, each having n values, will require a total of nm test cases to exhaustively test all the possible interactions among input parameters. Furthermore, the number of test cases increases exponentially with the increase in the number of parameters, which makes exhaustive testing impractical for large software systems. In the existing literature it has been reported that nearly 100% of the failures are triggered by interactions among 6 parameters. This is the main motivation behind Combinatorial Interaction Testing (CIT) which selects values of input parameters and combines them to generate test cases so as to test all t-way interactions of input parameters. CIT is a black-box testing technique which only requires information about the input parameters of the software system and their values. Empirical studies [2–5] show that a test set covering all possible 2-way combination of input parameter values is effective for software systems and can find a large percentage of the existing faults. Kuhn et al. [6] examined fault reports for many software systems and concluded that more than 70% of the faults are triggered by 2-way interaction of input parameters. Testing all 2-way interactions of input parameters values is known as pair-wise testing. The effectiveness of pair-wise testing in detecting a comparable number of faults early was a key factor driving the research work presented in this paper.

Covering Arrays (CAs) are combinatorial objects that are used to represent test sets for a system where the cardinality of values taken by each input parameter is the same. However, in a system with varying cardinalities of input parameter values, Mixed Covering Arrays (MCAs) are employed, which are a generalization of CAs that are used to represent test sets. The rows of CA/MCA correspond to test cases. As design of test cases is a crucial factor in determining the quality of software testing, the aim of CIT is to generate an optimal CA/MCA that provides 100% coverage of t-way interactions of input parameters. Lei and Tai [7] have shown that the problem of constructing optimal CA for pair-wise testing is NP complete. Many greedy [2, 3, 8–22] and meta-heuristic based optimizations algorithms/tools [23–34] have been developed by the researchers in the past with the aim of generating a near optimal CA/MCA. Both greedy and meta-heuristic techniques have merits and demerits. Greedy techniques are efficient as compared to their meta-heuristic counterparts in terms of CA/MCA generation time whereas meta-heuristic techniques generate optimal CA/MCA as compared to their greedy counterparts. The impressive results of existing meta-heuristic optimization algorithms to generate optimal CA/MCA motivated us to explore yet another optimization algorithm, namely the Artificial Bee Colony algorithm (ABC). It is proposed by Karaboga [35] and has been used to find an optimum solution for many optimization problems [36].

In this paper, we propose ABC-CAG (Artificial Bee Colony-Covering Array Generator) strategy that implements the ABC algorithm to generate optimal CA/MCA for pair-wise testing. The main contribution of this paper is to propose a strategy that integrates greedy approach and meta-heuristic approach thereby exploiting the strength of both techniques, to generate CA/MCA.

The remainder of this paper is organized as follows. In Section 2, we briefly describe combinatorial objects: CA and MCA. Section 3 discusses the existing state-of-the-art algorithms for constructing CA/MCA for pair-wise testing. In Section 4, we present a brief overview of ABC. Section 5 describes the proposed ABC-CAG strategy to generate CA for pair-wise testing. Section 6 describes the implementation of the proposed approach and presents empirical results to show the effectiveness of the proposed approach. Section 7 discusses threats to validity. Section 8 concludes the paper and future plans are discussed.

2. Background

This section discusses the necessary background related to combinatorial objects.

2.1. Covering Array

A covering array [37] denoted by $\mathrm{CA}_\lambda(N; t, k, v)$, is an $N \times k$ two dimensional array on v symbols such that every $N \times t$ sub-array contains all ordered subsets from v symbols of size t at least λ times. If $\lambda = 1$, it means that every t-tuple needs to be covered only once and we can use the notation $\mathrm{CA}(N; t, k, v)$. Here, k represents the number of values of each parameter and t is the strength of testing. An optimal CA contains the minimum number of rows to satisfy the properties of the entire CA. The minimum number of rows is known as a covering array number and is denoted by $\mathrm{CA}\,N(t, k, v)$. A CA of size $N \times k$ represents a test set with N test cases for a system with k input parameters each having an equal number of possible values.

2.2. Mixed Covering Array

A mixed covering array [38] denoted by $\mathrm{MCA}(N; t, k, (v_1, v_2, \ldots, v_k))$, is an $N \times k$ two dimensional array, where $v_1, v_2, \ldots, v_k$ is a cardinality vector which indicates the values for every column. An MCA has the following two properties: i) Each column i $(1 \leq i \leq k)$ contains only elements from a set S_i with $|S_i| = v_i$ and ii) The rows of each $N \times t$ sub-array cover all t-tuples of values from the t columns at least once. The minimum N for which there exists an MCA is called a mixed covering array number and is denoted by $\mathrm{MCA}\,N(t, k, (v_1, v_2, \ldots, v_k))$. A shorthand notation can be used to represent MCAs by combining equal entries in $v_i : 1 \leq i \leq k$. An $\mathrm{MCA}(N; t, k, (v_1, v_2, \ldots, v_k))$ can be represented as $\mathrm{MCA}(N; t, k, (w_1^{q_1}, w_2^{q_2}, \ldots, w_s^{q_s}))$, where $k = \sum_{i=1}^{s} q_i$ and $w_j | 1 \leq j \leq s \subseteq \{v_1, v_2, \ldots, v_k\}$. Each element $w_j^{q_i}$ in the set $\{w_1^{q_1}, w_2^{q_2}, \ldots, w_s^{q_s}\}$ means that q_i parameters can take w_j values each. A MCA of size $N \times k$ represents a test set with N test cases for a system with k components each with a varying domain size.

3. Related Work

Over the past two decades mathematicians and researchers in computer science have proposed various algorithms and tools to generate CA/MCA. Mathematicians use algebraic methods to generate CA [39–42]. These methods are extremely fast, however, they are mostly designed to generate CAs only. Researchers in the field of software testing have designed greedy algorithms to construct optimal CAs and MCAs. Greedy algorithms use two approaches to construct CA/MCA: one-test-at-a-time and one-parameter-at-a-time. In one-test-at-a-time [2, 3, 8–21], CA is constructed by generating one test at a time until all the uncovered combinations are covered. Each subsequent test is generated in such a way that it can cover the maximum possible number of uncovered combinations. In the case of one-parameter-at-a-time approach, such as ACTS [22], a pair-wise test set is constructed by generating a pair-wise test set for the first two parameters and then extending it to generate a pair-wise test set for three parameters and continues to do so for each additional parameter.

Recently, meta-heuristic techniques such as simulated annealing (SA), particle swarm optimization (PSO), tabu search (TS), ant colony optimization (ACO), hill climbing (HC), genetic algorithm (GA) and cuckoo search (CS) have been used by researchers to generate optimal CA/MCA. Meta-heuristic search techniques start from a pre-existing CA/MCA or a population of CA/MCA and apply a series of transformations on them until until a CA/MCA that covers all the uncovered combinations is found. Greedy algorithms generate CA/MCA faster as compared to meta-heuristic techniques, however, meta-heuristic techniques usually generate smaller CA/MCA [38]. Table 1 gives a summary of existing tools/algorithms for constructing optimal CA/MCA.

Table 1. List of existing tools/algorithms to construct CA/MCA for pair-wise testing

S. No.	Tool/Algorithm	Maximum strength support(t)	Technique employed	Test generation strategy	Constraint handling
1	Test cover [39]	4	algebraic	one parameter at a time	✓
2	TConfig [40]	2			✗
3	CTS [41]	4			✓
4	Algebraic method [42]	2			✗
5	AETG [2]	2	greedy	one test at a time	✓
6	TCG [8]	2			✓
7	ITCH [9]	6			✓
8	TVG [10]	6			✓
9	AllPairs [11]	2			✗
10	PICT [12]	6			✓
11	Jenny [13]	8			✓
12	Density [14]	3			✗
13	DA-RO [15]	3			✗
14	DA-FO [15]	3			✗
15	TSG [16]	3			✗
16	G2Way [17]	2			✗
17	GTWay [18]	12			✗
18	MT2Way [19]	2			✗
19	EPS2Way [20]	2			✗
20	CASCADE [21]	6			✓
21	ACTS (IPOG) [22]	6	greedy	one parameter at a time	✗
22	Paraorder [14]	3			✗
23	GA [23]	3	genetic algorithm	test based generation	✗
24	ACA [23]	3	ant colony optimization		✗
25	PSO [24]	2	particle swarm optimization		✗
26	TSA [25, 26]	6	tabu search		✗
27	SA [27]	6	simulated annealing		✗
28	PPSTG [28]	6	particle swarm optimization		✗
29	CASA [29]	3	simulated annealing		✓
30	GAPTS [30]	2	genetic algorithm		✗
31	PWiseGen [31]	2	genetic algorithm		✗
32	GA [32]	2	genetic algorithm		✗
33	CS [33]	6	cuckoo search		✗
34	FSAPSO [34]	4	adaptive particle swarm optimization		✗
35	PSO [24]	2	meta-heuristic	particle swarm optimization	parameter based generation	✗

4. Artificial Bee Colony (ABC) Algorithm

The application of Artificial Intelligence (AI) based algorithms to solve various optimization problems in the field of software testing is an emerging area of research. AI based algorithms can be classified into different groups depending upon the criteria being considered such as population based, iterative based, stochastic based, deterministic, etc. [43]. Population based algorithms commence with a population of individuals (initial solutions) and evolve the population to generate a better solution by performing various recombination and mutation operations. Population based AI algorithms are further categorized as evolutionary algorithms (EA) and swarm intelligence (SI) based algorithms. The well-known EAs are Genetic Algorithm (GA), Genetic Programming (GP), Evolutionary Programming (EP), Evolution Strategy (ES) and Differential Evolution (DE). SI based algorithms are inspired by the collective behaviour of social insect colonies and other animal societies [44]. Various algorithms have been developed by researchers by modelling the behaviour of different swarms of social insects such as ants and bees, flocks of birds or schools of fishes. The well known SI based algorithms are Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO) and algorithms based on the specific intelligent behaviour of honey bee swarms such as honey bee mating optimization (HBMO) [45], ABC [36], Bee Colony optimization (BCO) [46] and Bee Swarm optimization(BSO) [47].

ABC is a swarm-based algorithm which simulates the intelligent foraging behavior of a honey bee swarm. Many researchers have compared the performance of ABC with other optimization algorithms such as GA, PSO, ACO and DE by evaluating their performance on various numerical functions which consist of unimodal and multimodal distributions [48]. In [49], Mala et al. proposed parallel ABC to generate optimal test suites for white box testing of software systems with path coverage, state coverage and branch coverage as test adequacy criteria and compared the performance of parallel ABC with sequential ABC, GA and random testing. The results of comparison showed that ABC is more effective than other optimization algorithms. Optimization algorithms are characterized by a trade-off between two mechanisms, namely exploration and exploitation and it is desirable to have a suitable balance between the two. the exploration process refers to the ability of the algorithm to look out for a global optimum whereas; exploitation process refers to the ability of applying the knowledge of previous solutions to look for better solutions (local search). In the ABC algorithm, an artificial bee colony is divided into three groups: employed bees, onlooker bees and scouts. Exploitation is done by means of employed bees and onlooker bees, and exploration is done by means of scouts. The number of employed bees or onlooker bees is equal to the number of individuals (solutions) in the population. In ABC the position of a food source represents a possible solution to the optimization problem and the nectar amount of the food source corresponds to the fitness (quality) of the food source. The various steps of ABC are as follows:

Step 1: Generation of initial population – ABC starts by generating an initial population of SN possible solutions to the given optimization problem randomly. Each solution $x_i \{i = 1, \ldots, SN\}$ is a D-dimensional vector, where D is the number of optimization parameters.

Step 2: Employed Bees Phase – Each employed bee selects a solution $x_i \{i = 1, \ldots, SN\}$ and tries to produce a new solution $v_i \{i = 1, \ldots, SN\}$ by updating the selected solution x_i using Equation (1). It then applies a greedy selection between the old solution and the newly generated solution and selects the one which has higher fitness (nectar amount of the food source).

$$v_{ij} = x_{ij} + \phi_{ij}(x_{ij} - x_{kj}) \tag{1}$$

Here,

x_{ij} (or v_{ij}) denotes the j^{th} dimension of x_i (or v_i),

$j \in \{1, 2, \ldots, D\}$ is a randomly selected dimension,

x_k is a randomly selected neighbour of $x_i | k \in \{1, 2, \ldots, SN\}$,

```
generate initial population
iteration=1
while (solution not found and iteration ≤ maximum number of iterations)
   employed bees phase
   onlooker bees phase
   scout phase
   memorize the best solution achieved so far
   iteration = iteration + 1
end while
```

Figure 1. ABC algorithm

SN is the number of food sources (solutions) in the population.
Although k is determined randomly, it has to be different from i. ϕ_{ij} is a random number between $[-1, 1]$. Subsequently, once all employed bees have performed the search operation, a probability is assigned to each solution $x_i | 1 \leq i \leq SN$ which is calculated using Equation (2).

$$P(x_i) = \frac{\text{fitness}(x_i)}{\sum_{n=1}^{SN} \text{fitness}(x_n)} \tag{2}$$

Step 3: Onlooker Bees Phase – The ensuing phase of onlooker bees selects a solution based on the probability assigned to them and performs modification on the selected solution using Equation (1). Then a greedy selection is applied as done in case of employed bees.
Step 4: Scout Phase – Scouts look out for a solution which has not been improved by employed bees or onlooker bees through a predefined number of cycles called the limit and replaces it with a randomly generated solution.
Step 2 – Step 4 are repeated until a solution is found or a maximum number of iterations is reached. For further explanation of the ABC algorithms, readers can refer to [35, 43]. ABC is good at exploration but poor in exploitation as employed bees and onlooker bees only modify a small part of the solution instead of taking the global best, which may lead to the trapping of the ABC in local minima [48]. In order to maintain a good balance between exploration and exploitation, various variants of ABC namely GABC [50], I-ABC [48] and PS-ABC [48] were proposed by researchers.
The outline of ABC is shown in Figure 1.

5. ABC-CAG Strategy to Generate CA

In this paper, we propose aa ABC-CAG strategy which applies ABC, a stochastic search based optimization algorithm to solve the problem of constructing an optimal CA/MCA for pair-wise testing. From now onwards CA refers to both CA and MCA unless mentioned explicitly. We start by defining a search space, which in our case consists of the input domain of all input parameters of the SUT. Let us consider a SUT having k input parameters. For each input parameter $IP_j \mid 1 \leq j \leq k$, the possible values of IP_j are represented by an integer number between $[0, v_j)$ where v_j is the maximum number of values that IP_j can have. We start by generating an initial population of PS individuals where an individual in our case is a CA that corresponds to a food source in ABC and represents a possible solution to the given problem. Each covering array $\text{CA}_i \mid i \in \{1, 2, \ldots, PS\}$ in the population is a $N \times k$ dimensional array. Let us consider a web based application where the customer has different options of the operating system, browsers, display, processor and memory which they may use as shown in Table 2.

In order to test the system thoroughly, we need to test the system on all possible configurations, e.g. Android, Safari, 240×320, dual core, 512 MB, etc. which would require a total of $5 \times 3 \times 3 \times 3 \times 4 = 540$ test cases, whereas only 20 test cases will be required to test all pair-wise combinations of features. A possible solution (set of test cases/MCA) to the test pair-wise interactions of features in Table 2 is shown in Figure 2.

Table 2. A web based application

Operating System	Browser	Display	Processor	Memory
Android	Opera mini	128×160	single core	256 MB
iOS	Safari	240×320	multi core	512 MB
Windows	Chrome	800×1280	dual core	1 GB
Blackberry				2 GB
Symbian				

$$\begin{vmatrix} \textit{Blackberry} & \textit{Safari} & \textit{800×1280} & \textit{single core} & \textit{256 MB} \\ \textit{iOS} & \textit{Opera mini} & \textit{800×1280} & \textit{dual core} & \textit{256 MB} \\ . & . & . & . & . \\ . & . & . & . & . \\ \textit{Windows} & \textit{Chrome} & \textit{126×160} & \textit{multi core} & \textit{512 MB} \end{vmatrix}$$

Figure 2. MCA of size 20×5 for pair-wise testing

When generating the initial population of $N \times k$ CAs, N is unknown at the start of the search process. So there are two possibilities. The first one is to use the method suggested by Stardom [51], where we set loose lower bound (LB) and upper bound (UB) on the size of an optimal CA and then apply binary search repeatedly to find the smallest CA. The second one, in case the size of N is known in advance, i.e. best bound achieved in the existing state-of-the-art, we start with the known size and try to optimize it further.

As discussed in Section 4, the fitness of a CA in the population is the measure of its quality. In our case, the fitness of a CA is defined as the total number of distinct 2-way interactions covered by it and is calculated using Equation (3) as:

$$\text{fitness}(\text{CA}_i) = |\{2-\text{way interactions covered by } \text{CA}_i\}| \quad \forall\, i \in \{1, \ldots, PS\} \quad (3)$$

The aim of ABC-CAG is to generate a covering array that maximizes the objective function $f|f : \text{CA}_i \rightarrow I^+$, where I^+ represents a set of positive integers. The objective function f tells us, how good a solution it is for a given problem. In our case f can be calculated as shown in Equation (4):

$$f(\text{CA}_i) = \text{fitness}(\text{CA}_i) \mid i \in \{1, \ldots, PS\} \quad (4)$$

Now, ABC-CAG tries to generate a covering array CA_{max} that maximizes the objective function f as shown in Equation (5):

$$\text{CA}_{\max} = \text{CA}_i \,|f(\text{CA}_i) = \max(f(\text{CA}_i) \quad \forall\, i \in \{1, \ldots, PS\}) \quad (5)$$

Having defined the problem representation and the fitness calculation, the next sub-section describes the various steps of the ABC-CAG strategy.

5.1. Generation of Initial Population

In EA's the role of the impact of the initial population on their performance cannot be ignored as it can affect the convergence speed and quality of the final solution [52]. Many population initialization methods exist to generate initial population. The most common method used to generate the initial population in EA is the random method. ABC-CAG uses the Hamming distance approach proposed by Bansal et al. [32] to generate a good quality initial population of CAs. The motive behind the use of the Hamming distance approach is to generate test cases in a CA in such a way that each new test case covers the maximum number of possible distinct 2-way interactions not covered by the previous test cases. The Hamming distance between two test cases (rows) in a CA is the number of positions in which they differ. Let PS be the population size and N be the size of CA. For each CA in PS, 50% (first N/2) of the test cases are created randomly. Let $tc_1, tc_2, \ldots, tc_i$ represent the test cases in CA generated till now. To create the next test case tc_j where $j = i + 1$,

a candidate test case tc is generated randomly and the Hamming distance of tc from tc_k, for all $k : 1 \leq k \leq i$, denoted by distance(tc), is calculated as:

$$\text{distance}(tc) = \sum_{k=1}^{i} \text{HD}(tc_k, tc) \quad (6)$$

Where, $\text{HD}(tc_k, tc)$ is the Hamming distance between tc_k and tc. An average distance denoted by avg_distance(tc) is calculated as follows.

$$\text{avg\_distance}(tc) = \text{distance}(tc)/(j-1) \quad (7)$$

Candidate test case tc is included in the test set TS only if

$$\text{avg\_distance}(tc) \geq \alpha \times N_{IP} \quad (8)$$

Where, α is a diversity factor whose value ranges from 0.3 to 0.4 and N_{IP} is the number of input parameters. Equation (8) implies that a candidate test case tc is included only if it covers at least 30%-40% of distinct input parameter values as compared to those covered by the existing test cases. The process is repeated until the remaining $N/2$ test cases are generated for the CA. The use of the Hamming distance to create $N/2$ test cases in each CA enhances the quality of initial population as compared to one generated using the random technique.

5.2. The Employed Bees Phase

The number of employed bees is equal to the population size PS. In ABC, each employed bee selects a dimension of a food source randomly and uses the local information to modify the solution using Equation (1). The new solution replaces the old one only if the former is better than the latter. However, in ABC-CAG each employed bee uses a greedy approach to select a test case (dimension) of a CA_i (solution). The impetus behind the greedy selection of test case in a CA_i by an employed bee is to formulate a new CA'_i from the selected CA_i in such a way that CA'_i contains all the test cases of the selected CA_i except its worst test case. The worst test case of CA_i is replaced in CA'_i by a test case generated using the information of a randomly selected neighbouring CA_m in an attempt to increase the overall coverage of 2-way interactions between input parameters. To select the worst test case in CA_i the fitness of each test case in CA_i is calculated by counting the number of distinct pairs covered by each one of them. For instance, suppose we have a CA with nine test cases as shown in Figure 3, for the system shown in Table 2.

We calculate the number of distinct pairs covered by each test case as shown in Table 3 and an employed bee selects the test case that covers the least number of distinct pairs (test case TC8 in Table 3). The value of each input parameter $IP_j\{j = 1, 2, \ldots, k\}$ of the test case covering the least number of distinct pairs is modified based on the values of the respective input parameters in the corresponding test case of the randomly selected neighbouring CA, i.e. CA_m using Equation (9).

$$\text{CA}'_{iqj} = \text{CA}_{iqj} + \phi_{iqj}(\text{CA}_{iqj} - \text{CA}_{mqj}) \quad (9)$$

Here, $i \in \{1, 2, \ldots, PS\}$, $q \in \{1, 2, \ldots, N\}$ represent the index of the worst test case in $\text{CA}_i, j \in \{1, 2, \ldots, k\}$ and $m \in \{1, 2, \ldots, PS\} \mid m \neq i$. Since ϕ_{iqj} is a random number between $[-1, 1]$, it is quite possible that a non-integral value may get generated as a result of calculation performed using Equation (9). To avoid such a condition, whenever a non-integer value is generated for an input parameter, it gets rounded to the nearest integer number. After rounding off the value, if it does not fall in the range $[0, v_j)$ then a value is selected randomly from the input domain of the respective parameter and it replaces the existing value of the selected parameter.

5.3. The Onlooker Bees Phase

Subsequently, the fitness of each CA in the search space is calculated and a probability is assigned to each of them using Equation (2). In ABC-CAG, to select a CA on the basis of the probability assigned to them, a random number is generated in the range $[0, 1]$ and based on the interval in which the random number falls; a covering array CA_i is selected by an onlooker bee. Unlike the traditional ABC which is good at exploration but

iOS	*Chrome*	*240×320*	*dual core*	*1 GB*
Android	*Chrome*	*800×1280*	*multi core*	*256 MB*
Blackberry	*Safari*	*800×1280*	*multi core*	*256 MB*
Blackberry	*Chrome*	*128×160*	*single core*	*2 GB*
Symbian	*Opera mini*	*128×160*	*multi core*	*512 MB*
Windows	*Safari*	*800×1280*	*dual core*	*512 MB*
iOS	*Opera mini*	*800×1280*	*multi core*	*2 GB*
Android	*Chrome*	*128×160*	*multi core*	*512 MB*
Blackberry	*Opera mini*	*240×320*	*single core*	*1 GB*

Figure 3. CA of size 9×5

Table 3. Calculation of distinct pairs covered by each test case of CA

TC1	TC2	TC3	TC4	TC5	TC6	TC7	TC8	TC9
iOS, Chrome	**Android**, **Chrome**	Blackberry, Safari	Blackberry, Chrome	Symbian, Opera mini	Windows, Safari	iOS, Opera mini	**Android**, **Chrome**	Blackberry, Opera mini
iOS, 240×320	Android, 800×1280	Blackberry, 800×1280	Blackberry, 128×160	Symbian, 128×160	Windows, 800×1280	iOS, 800×1280	Android, 128×160	Blackberry, 240×320
iOS, dual core	**Android**, **multi core**	Blackberry, multi core	*Blackberry*, **single core**	Symbian, multi core	Windows, dual core	iOS, multi core	**Android**, **multi core**	*Blackberry*, *single core*
iOS, 1 GB	Android, 256 MB	Blackberry, 256 MB	Blackberry, 2 GB	Symbian, 512 MB	Windows, 512 MB	iOS, 2 GB	Android, 512 MB	Blackberry, 1 GB
Chrome, 240 × 320	Chrome, 800 × 1280	**Safari**, **800×1280**	*Chrome*, *128×160*	Opera mini, 128×160	**Safari**, **800×1280**	Opera mini, 800×1280,	*Chrome*, *128×160*	Opera mini, 240×320
Chrome, dual core	**Chrome**, **multi core**	Safari, multi core	Chrome, single core	*Opera mini*, *multi core*	Safari, dual core	*Opera mini*, *multi core*	**Chrome**, **multi core**	Opera mini, single core
Chrome, 1 GB	Chrome, 256 MB	Safari, 256 MB	Chrome, 2 GB	Opera mini, 512 MB	Safari, 512 MB	Opera mini, 2 GB	Chrome, 512 MB	Opera mini, 1 GB
240×320, dual core	**800×1280**, **multi core**	**800×1280**, **multi core**	128×160, single core	*128×160*, *multi core*	800×1280, dual core	**800×1280**, **multi core**	*128×160*, *multi core*	240×320, single core
240×320, *1 GB*	**800×1280**, **256 MB**	**800×1280**, **256 MB**	128×160, 2 GB	***128×160***, ***512 MB***	800×1280, 512 MB	800×1280, 2 GB	***128×160***, ***512 MB***	*240×320*, *1 GB*
dual core, 1GB	**multi core**, **256 MB**	**multi core**, **256 MB**	single core, 2 GB	*multi core*, *512 MB*	dual core, 512 MB	multi core, 2 GB	*multi core*, *512 MB*	single core, 1 GB
Distinct pairs covered by each test case:								
9	4	6	8	6	9	8	**3**	8

poor at exploitation, ABC-CAG takes advantage of the global best CA denoted by CA^{best} in the population (based on gbest-guided ABC (GABC) developed by Zhu and Kwong [50]) to guide the search of a candidate solution and modifies the selected CA_i. Like an employed bee, an onlooker bee selects the worst test case (dimension) of the selected CA_i and replaces it with a test case that is generated by using the information of the global best CA i.e., CA^{best} and a randomly selected neighbouring CA i.e., CA_m using Equation (10).

$$\text{CA}'_{iqj} = \text{CA}_{iqj} + \phi_{iqj}(\text{CA}_{iqj} - \text{CA}_{mqj}) + \psi_{iqj}(\text{CA}^{\text{best}}_{qj} - \text{CA}_{iqj}) \quad (10)$$

Here, $\text{CA}^{\text{best}}_{qj}$ is the value of j^{th} parameter of q^{th} test case of the global best CA^{best}, ψ_{iqj} is a uniform random number in [0, C], where C is a non-negative constant. The GABC technique drives the new candidate solution CA'_i towards the global best solution, thereby improving its exploitation capabilities.

However, in case the best CA i.e., CA^{best} gets selected per se, based on the generated random number; the ABC-CAG modifies it by replacing its worst test case by a smart test case. A smart test case is constructed by selecting the value for each parameter greedily. For each parameter, the value whose occurrence in the best CA is minimum is selected. The replacement of the worst test case in the best CA by a smart test case is done to make sure that certain new pairs get covered by this replacement. An example to illustrate the selection and modification done by onlooker bees phase is shown in Table 4.

Let us consider a system having configuration $(N; 2, 2^2 3^3)$ as shown in Table 4a.

The total number of distinct 2-way interactions in this system is 67. Let our population size PS be 8 which means that the population consists of 8 CAs and let us assume that the size of each CA array is 7×5 which means that a CA consists of 7 test cases. After an initial population of CAs is generated, each employed bee modifies a CA as discussed in Section 5.2. The fitness and the probability assigned to each CA generated after being modified by the employed bee is shown in Table 4b.

Here, CA_4 is the global best CA. Let a random number 'r' be generated to select a CA by the onlooker bee. There are two cases:
Case 1: When the global best CA is different from the CA selected by the onlooker bee – let r be 0.8. Based on the value of r, covering array CA_7 gets selected by the onlooker bee and lets CA_1 be the randomly selected neighbour of CA_7. Also according to the fitness values, CA_4 is CA^{best}. The test cases of CA1, CA4 and CA7 are shown in Tables 4c–4e.

After calculating the number of distinct pairs covered by each test case of CA_7, it is clear that TC2 covers the least number of distinct pairs i.e., 1. So the value of each parameter in the test case TC2 of CA_7 is modified using Equation (10). For performing calculations, the values of each input parameter have been mapped to integer values (0/1/2) and the new covering array CA'_7 generated after modification is shown in Table 4f.

After modification by the onlooker bee based on the global best CA and the randomly selected neighbouring CA, the fitness of the newly generated covering array CA'_7 increases by 2 and becomes 47.
Case 2: When the global best CA and the CA selected by the onlooker bee is the same – In this case the worst test case of the global best covering array CA^{best} is replaced by a smart test case. However, in our case the global best covering array CA_4 has three test cases: TC1, TC3 and TC4 that cover the least number of distinct pairs. Here, ABC-CAG selects a test case randomly from the three test cases covering the least number of distinct pairs. Let the randomly selected test case be TC3. Now, TC3 will be replaced by a smart test case which in this case is 'b1 b2 b3 c4 b5' as these values have the least number of occurrences in CA_4. The replacement of the worst test case of CA_4 by the smart test case increases its fitness by 2.

The above procedure is repeated for each onlooker bee.

5.4. The Scouts Phase

ABC's exploration strategy is effectuated by scout bees replacing a food source abandoned by an employed bee with a randomly generated food source. To further enhance the exploration capability of ABC, we use a greedy approach to select a CA instead of the primitive approach followed by ABC. In ABC-CAG, a scout replaces the worst CA (least fitness) in the population by a new CA. In ABC, the food sources that cannot be improved through a predetermined threshold called the limit are abandoned. The aforementioned abandoned food sources are thereupon replaced by randomly created new food sources by an artificial scout. ABC-CAG necessitates setting the frequency of the scout operation with discretion: a very high value of frequency will proliferate diversity of the population and avoid getting stuck in local minima but concurrently makes it difficult to converge to a good solution, whereas a lower value of frequency will result in early convergence leading to a suboptimal solution. Hence, it is required to set the frequency of scout denoted by f_{scout} to an optimal value.

Table 4. Example: the selection and modification done by the onlooker bees phase

a) A $(N; 2, 2^2 3^3)$ system

IP1	IP2	IP3	IP4	IP5
a1	a2	a3	a4	a5
b1	b2	b3	b4	b5
		c3	c4	c5

b) The fitness and the probability assigned to each CA

CA	CA_1	CA_2	CA_3	CA_4	CA_5	CA_6	CA_7	CA_8
Fitness	49	47	51	52	45	44	45	47
Probability	0.119	0.114	0.134	0.137	0.118	0.116	0.118	0.114

c) CA_1 (randomly selected neighbouring CA)

TC1					TC2					TC3					TC4					TC5					TC6					TC7				
a1	a2	a3	a4	b5	b1	a2	b3	a4	c5	a1	a2	c3	a4	c5	a1	b2	a3	c4	b5	a1	b2	a3	a4	c5	b1	a2	c3	b4	b5	b1	b2	b3	c4	a5

d) CA_4 (best CA)

TC1					TC2					TC3					TC4					TC5					TC6					TC7				
a1	a2	a3	b4	b5	a1	a2	c3	b4	b5	b1	b2	c3	b4	c5	a1	a2	b3	a4	c5	a1	a2	c3	c4	c5	a1	b2	a3	c4	c5	b1	a2	a3	a4	c5

e) CA_7 (CA selected by onlooker on the basis of probability)

TC1					**TC2**					TC3					TC4					TC5					TC6					TC7				
a1	a2	a3	b4	b5	**a1**	**a2**	**c3**	**b4**	**b5**	b1	b2	c3	b4	c5	a1	a2	b3	a4	c5	a1	a2	c3	c4	c5	a1	b2	a3	c4	c5	b1	a2	a3	a4	c5
Minimum distinct pairs covered by each test case:																																		
2					1					6					5					2					4					4				

f) CA'_7 (CA_7 after modifications)

TC1					**TC2**					TC3					TC4					TC5					TC6					TC7				
a1	a2	a3	b4	b5	**b1**	**a2**	**b3**	**c4**	**c5**	b1	b2	c3	b4	c5	a1	a2	b3	a4	c5	a1	a2	c3	c4	c5	a1	b2	a3	c4	c5	b1	a2	a3	a4	c5

ABC-CAG replaces the worst CA by a randomly generated CA after every f_{scout} generation. For example, in the example given in Section 5.3, a scout will replace the worst covering array i.e., CA_6 by a randomly created new CA.

All the three phases, namely the employed bees phase, the onlooker bees phase and the scout phase are perpetuated until a solution is found or the maximum number of generations is reached.

6. Evaluation

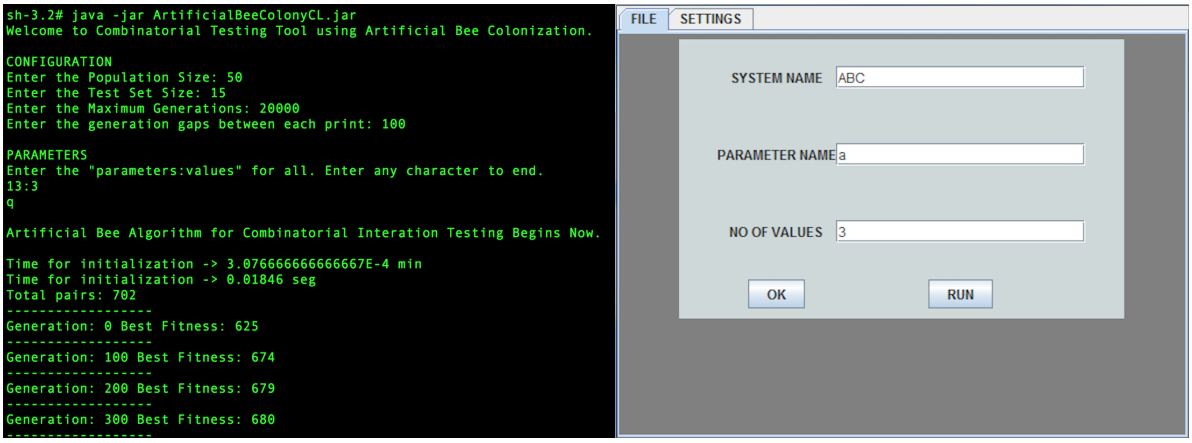
We start our evaluation by presenting three research questions in Section 6.1. Then we outline our implementation and experimental design in Section 6.2. The results and analysis are shown in Section 6.3.

6.1. Research Questions

Many greedy and meta-heuristic techniques have been proposed by researchers in the past with the aim of generating an optimal CA. The ultimate goal of the research work presented in this paper is to develop a strategy that generates optimal CA as compared to existing state-of-the-art algorithms/tools. Our first objective is therefore to measure the effectiveness of the proposed approach:

RQ1: (Comparison of ABC-CAG with existing techniques) How effective is ABC-CAG in generating an optimal CA with respect to the existing state-of-the art algorithms/tools?

In addition to evaluating the effectiveness of ABC-CAG, it is important to check whether ABC-CAG is comparable in terms of runtime



a) Command Line Interface (CLI) b) Graphical User Interface (GUI)

Figure 4. ABC-CAG interfaces

with the state-of-the art algorithms. Our next research question is:

RQ2: (Efficiency of ABC-CAG) How efficient is ABC-CAG in generating an optimal CA?

ABC is a meta-heuristic search algorithm and all search algorithms are randomized in nature. Randomized algorithms are used to solve problems where it is not possible to solve the problem in a deterministic way within a reasonable time and they are associated with a certain degree of randomness as part of their logic. Due to the randomized nature of search algorithms, running them on the same problem instance multiple times produces different results. It is therefore important to analyse their results and compare them against simpler alternatives. This motivates our next research question:

RQ3: (Effectiveness of ABC-CAG) How effective is ABC-CAG when applied to the problem of generating an optimal CA for pair-wise testing as against existing meta-heuristic techniques?

6.2. Experimental Design

To answer the research questions asked in Section 6.1, we have implemented ABC-CAG using Java. Two types of external interfaces have been provided: Command Line Interface (CLI) and Graphical User Interface (GUI) which are shown in Figure 4a and Figure 4b, respectively.

ABC-CAG takes population size PS and maximum number of iterations NI as input. As discussed in Section 5, in ABC-CAG there is an option of whether we want to start from a known N or we want to start with a large random array whose size is calculated as suggested by Stardom [51]. If we start with a known N, then the user has to supply the value of N as input to ABC-CAG.

To answer RQ1 and RQ2, three sets of experiments were conducted. In the first experiment, a Traffic collision avoidance system (TCAS) benchmark, which has been used by many researchers [25, 26, 53] in the past to compare CA generation strategies, is taken to evaluate the performance of ABC-CAG with respect to the existing state-of-the-art algorithms. TCAS has 12 control parameters with 2 parameters having 10 values, 1 having 4 values, 2 having 3 values and 7 having 2 values each. It can be represented by an MCA instance MCA$(N; 2, 12, 10^2 4^1 3^2 2^7)$.

In the second experiment, we took a case study of various features of a printer that are available while printing a document as shown in Figure 5. The printer case study is a practical example that models and illustrates the concept of combinatorial testing.

It is clear from Figure 5 that there are 7 features that a user can set during printing. However, the feature 'Resolution' is only for information and a user cannot change its value. So, we consider only 6 features and regard them as input parameters. Out of these 6 input parameters, 3 parameters

have 2 values each, 2 parameters have 3 values each whereas the remaining 1 parameter has 4 values as shown in Table 5. This problem can be represented by an MCA instance $MCA(N; 2; 5; 2^3 3^2 4^1)$.

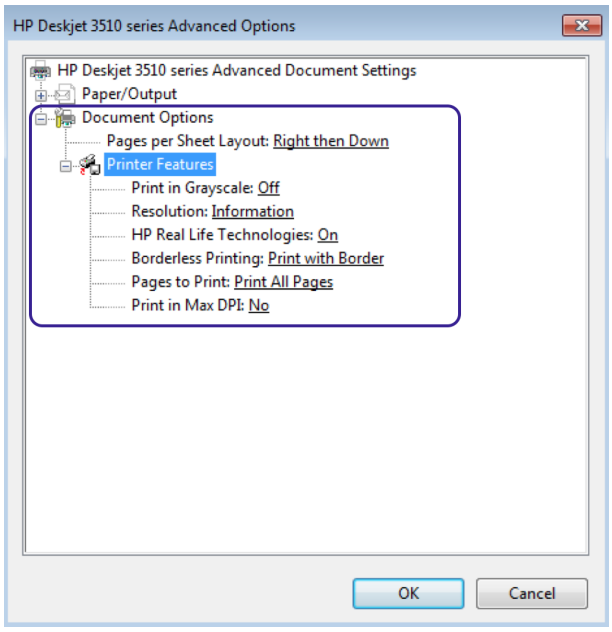


Figure 5. Various features of a printer

To ensure fair comparison and to evaluate the effectiveness of ABC-CAG, in the third experiment we took a dataset that consists of 20 benchmark problems selected carefully from the existing literature [25–28,38,54] on pair-wise testing as shown in Table 6. Each benchmark problem in the dataset is either an instance of CA or MCA. For example the problem 3^3 in Table 6 represents a CA instance $CA(N; 2, 3, 3^3)$ which means that the system has 3 input parameters each having 3 possible values and the strength of testing is 2. Similarly the problem $5^1 3^8 2^2$ in Table 6 represents an MCA instance $MCA(N; 2, 11, 5^1 3^8 2^2)$ which means that the system has 11 input parameters with 1 parameter having 5 values, 8 parameters having 3 values and 2 parameters having 2 values each and the strength of testing is 2.

As CA size is absolute and is independent of the system configuration, to answer RQ1, we compared CA sizes based on the data available in the existing literature [21, 25–28, 33, 34, 38, 54]. However, CA generation time is dependent on the system configuration, so to answer RQ2 and to ensure a fair comparison, we limited our comparison based on CA generation time to only those algorithms whose implementations are publicly available, which includes greedy algorithms based tools, namely AllPairs, Jenny, TVG, ACTS(IPOG), and meta-heuristic algorithms based tools namely CASA, PWiseGen [55]. The CA generation time was obtained by executing the dataset of Table 5 on these tools under Windows using an INTEL Pentium Dual Core 1.73 GHZ processor with 1.00 GB of memory.

To answer RQ3, we need to perform a statistical test to compare ABC-CAG with meta-heuristic techniques. We compared ABC-CAG with two meta-heuristic tools namely PWiseGen and CASA.

PWiseGen starts with a known array size N and tries to generate a CA that covers 100% pair-wise combinations of input parameter values. Therefore, to compare ABC-CAG and PWiseGen we ran each problem on both of them 30 times and noted down the fitness of the CAs/MCAs obtained during these runs. The value of N was kept the same for a problem during these 30 runs. To compare ABC-CAG and CASA, we ran each problem on ABC-CAG and CASA 30 times and noted down the size of generated CA/MCA.

6.3. Results and Analysis

Here we present the results of experiments conducted to answer RQ1, RQ2 and RQ3.

6.3.1. Comparison of ABC-CAG with Existing Techniques (RQ1)

Results: The result of experiments performed to compare ABC-CAG with the existing techniques for TCAS, the printer case study and the dataset shown in Table 6 are shown in Table 7, Table 8 and Table 9 respectively. Entries marked '–' mean that the results are not available. Since ABC-CAG is a randomized algorithm and it gives different results when run multiple times on the same problem instance, therefore we report the best as well as average CA size obtained over multiple runs.

Table 5. Various features of a printer

HP Real Life Technologies	Borderless Printing	Print in max DPI	Pages per sheet layout	Print in grayscale	Pages to print
On	Print with border	No	Left then down	Off	Print all pages
Off	Print borderless	Yes	Down then left	High quality grayscale	Print even pages only
			Right then down	Black ink only	Print odd pages only
			Down then right		

Table 6. Dataset

Sno.	Benchmark Problems	k (number of input parameters)	Total number of pairs
1	3^3	3	27
2	3^4	4	54
3	3^{13}	13	702
4	5^{10}	10	1125
5	10^{20}	20	19000
6	2^{100}	100	19800
7	$4^5 3^4$	9	454
8	$5^1 3^8 2^2$	11	492
9	$7^2 6^2 4^2 3^2 2^2$	10	854
10	$8^2 7^2 6^2 5^2$	8	1178
11	$6^4 4^5 2^7$	16	1556
12	$5^1 4^4 3^{11} 2^5$	21	1944
13	$6^1 5^1 4^6 3^8 2^3$	19	1992
14	$6^2 4^9 2^9$	20	2052
15	$6^5 5^5 3^4$	14	2074
16	$7^1 6^1 5^1 4^5 3^8 2^3$	19	2175
17	$6^9 4^3 2^7$	19	3000
18	$6^7 4^8 2^3$	18	3004
19	$4^{15} 3^{17} 2^{29}$	61	14026
20	$4^1 3^{39} 2^{35}$	75	17987

Analysis: It is clear from Tables 7–9 that ABC-CAG generates CA of smaller size as compared to greedy algorithms. When compared to meta-heuristic techniques, ABC-CAG generates comparable results in the case of TCAS and the printer case study, whereas it outperforms GA, ACA, PSO, PPSTG, PWiseGen and CS in the case of benchmark problems given in dataset of Table 6. When compared to CASA, it can easily be seen from Table 9 that in 60% cases the size of CAs generated by ABC-CAG and CASA are the same. In 25% of cases ABC-CAG generates smaller CAs whereas in only 15% of cases CASA outperforms ABC-CAG. In the case of TS, the results are comparable. Overall, ABC-CAG outperforms all greedy and most of the meta-heuristic techniques and generates a smaller size CA for pair-wise testing.

6.3.2. Efficiency of ABC-CAG (RQ2)

Results: The time (in seconds) taken by each of the publicly available tools, namely Jenny, All-Pairs, TVG, ACTS (IPOG), CASA, PWiseGen, and the proposed algorithm ABC-CAG for generating CA for TCAS, the printer case study and

Table 7. Comparison of MCA sizes generated for TCAS

Problem Instance	Pairs	ACTS (IPOG)	PICT	AllPairs	ITCH	Jenny	TConfig	TVG	TSA	CASA best	CASA avg.	PWise-Gen	ABC-CAG best	ABC-CAG avg.
$10^2 4^1 3^2 2^7$	837	100	100	100	120	108	108	100	100	100	100	101	100	100

Table 8. Comparison of MCA sizes generated for printer case study

Problem Instance	Pairs	ACTS (IPOG)	PICT	AllPairs	Jenny	TVG	CASA best	CASA avg.	PWise-Gen	ABC-CAG best	ABC-CAG avg.
$3^2 2^3 4^1$	105	12	14	16	14	13	12	12	12	12	12

Table 9. Comparison of CA/MCA sizes generated for the 20 benchmark problems

Benchmark Problems	AETG	TCG	All-Pairs	PICT	Jenny	CAS-CADE	ACTS (IPOG)	GA	ACA	PSO	TS	PPSTG	CASA best	CASA avg.	PWise-Gen	CS	FSAPO	ABC-CAG best	ABC-CAG avg.
3^3	–	–	9	10	9	–	9	–	–	9	–	–	9	9	9	9	9	9	9
3^4	9	–	9	12	11	9	9	9	9	9	–	9	9	9	9	9	9	9	9
3^{13}	15	20	17	20	18	–	19	18	18	18	–	17	16	16.24	16	20	16	15	15.93
5^{10}	–	–	47	47	45	–	45	–	–	–	–	45	38	39.7	43	–	–	41	41.6
10^{20}	180	218	197	216	193	–	227	227	232	213	–	–	192	193.33	224	–	–	210	211.5
2^{100}	10	16	14	16	16	–	16	14	14	–	–	–	11	11	11	–	–	10	10.43
$4^5 3^4$	–	–	22	26	26	–	24	–	–	–	19	–	19	20	21	–	–	19	19.83
$5^1 3^8 2^2$	20	20	20	20	23	–	19	17	17	17	15	21	15	16.24	16	21	18	15	15.96
$7^2 6^2 4^2 3^2 2^2$	–	–	54	56	57	–	53	–	–	–	–	–	49	49.3	50	–	–	49	49.6
$8^2 7^2 6^2 5^2$	–	–	64	80	76	–	72	–	–	–	64	–	64	65.13	72	–	–	64	64.4
$6^4 4^5 2^7$	–	–	45	55	53	–	44	–	–	–	38	–	41	52.8	47	–	–	39	41.9
$5^1 4^4 3^{11} 2^5$	30	30	27	32	32	–	26	26	27	27	22	–	22	23.7	26	–	–	22	23.9
$6^1 5^1 4^6 3^8 2^3$	34	41	34	38	40	–	36	33	34	35	30	39	30	30.26	33	43	35	30	30.2
$6^2 4^9 2^9$	–	–	38	41	44	–	39	–	–	–	36	–	36	36.4	39	–	–	36	36.16
$6^5 5^5 3^4$	–	–	53	59	56	–	56	–	–	–	50	–	47	49.33	55	–	–	51	52.5
$7^1 6^1 5^1 4^5 3^8 2^3$	45	45	43	46	50	–	43	43	43	43	42	49	42	42	43	51	–	42	42.16
$6^9 4^3 2^7$	–	–	59	67	64	–	61	–	–	–	51	–	52	53.23	61	–	–	51	51.66
$6^7 4^8 2^3$	–	–	53	63	63	–	54	–	–	–	47	–	48	49.86	57	–	–	47	48.1
$4^{15} 3^{17} 2^{29}$	37	33	35	38	39	–	33	38	38	38	30	30	30	30.53	33	–	–	30	30.43
$4^1 3^{39} 2^{35}$	27	–	26	29	31	–	28	29	28	27	22	–	22	22.9	24	–	–	22	22.73

the dataset of Table 6 are shown in Table 10, Table 11 and Table 12, respectively.

Analysis: It is evident from Tables 10–12 that meta-heuristic techniques take a longer time to generate CA as compared to their greedy counterparts. However, the extra time taken by meta-heuristic techniques allows them to generate smaller CA/MCA than greedy algorithms. When we compare the time taken by CASA, PWiseGen and ABC-CAG, it has been observed that out of the three meta-heuristic techniques, CASA takes the minimum time to generate CA whereas the time taken by PWiseGen and ABC-CAG is comparable.

6.3.3. Effectiveness of ABC-CAG (RQ3)

Results: To compare ABC-CAG and PWiseGen, we performed a statistical test namely Welch's t-test on the fitness of CAs/MCAs generated during 30 runs for each benchmark problem in the dataset except the two benchmark problems CA (3^3) and CA (3^4), TCAS and the printer case study, as the fitness of CAs generated during 30 runs of TCAS, the printer case study and the two aforementioned benchmark problems were the same for both techniques. Hence, there is no reason for performing the Welch t-test on these two problems. Each problem was run 30 times as a minimum of 30 data points are required for Welch's t-test [56]. From Table 7 and Table 9, it is evident that ABC-CAG generates smaller CAs than PWiseGen, however by performing Welch's t-test we try to assess whether the difference is significant or not. To do this, we formulate null hypothesis H_0 as:

There is no difference between the average fitness of CAs generated by ABC-CAG and the average fitness of CAs generated by PWiseGen.

When a statistical test is performed, two types of errors are possible: (I) we reject the null hypothe-

Table 10. Comparison of time (in seconds) to generate MCA for TCAS

Problem Instance	ACTS (IPOG)	AllPairs	Jenny	TVG	CASA	PWiseGen	ABC-CAG
$10^2 4^1 3^2 2^7$	0.07	0.13	0.545	0.09	0.726	20.34	0.5

Table 11. Comparison of time (in seconds) to generate MCA for the printer case study

Problem Instance	ACTS (IPOG)	AllPairs	Jenny	TVG	CASA	PWiseGen	ABC-CAG
$3^2 2^3 4^1$	0.01	0.016	0.015	0.10	0.269	0.32	0.15

Table 12. Comparison of time (in seconds) to generate CA/MCA for the 20 benchmark problems

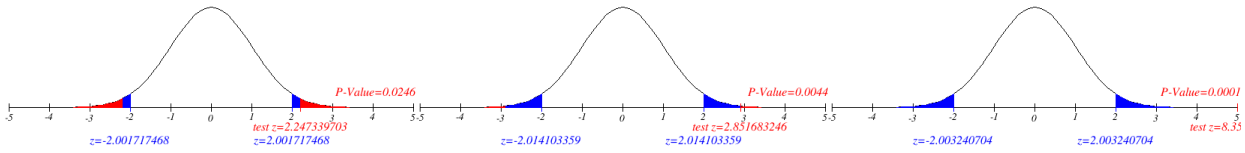
Problem Instance	ACTS (IPOG)	AllPairs	Jenny	CASA	PWiseGen	ABC-CAG
3^3	0.047	0.059	0.034	0.104	0.0546	0.06
3^4	0.002	0.012	0.027	0.151	4.392	0.106
3^{13}	0.014	0.018	0.144	2.23	29.428	70.98
5^{10}	0.003	0.013	0.309	3.608	47.017	123.708
10^{20}	0.53	0.009	2.615	10852.35	675.168	2057.943
2^{100}	0.078	0.071	1.208	4.6985	701.25	1654.17
$4^5 3^4$	0.001	0.012	0.15	0.6785	20.88	37.2
$5^1 3^8 2^2$	0.002	0.013	0.095	0.8145	22.62	47.1
$7^2 6^2 4^2 3^2 2^2$	0.001	0.01	0.141	2.54	135.42	91.2
$8^2 7^2 6^2 5^2$	0.003	0.007	0.183	10.28	50.04	100.5
$6^4 4^5 2^7$	0.006	0.01	0.329	26.36	83.52	133.4
$5^1 4^4 3^{11} 2^5$	0.015	0.016	0.202	3.15	84.12	74.9
$6^1 5^1 4^6 3^8 2^3$	0.016	0.009	0.229	11.2	40.68	78.65
$6^2 4^9 2^9$	0.016	0.009	0.247	2.21	46.38	180.8
$6^5 5^5 3^4$	0.015	0.015	0.249	106.56	87.18	246.87
$7^1 6^1 5^1 4^5 3^8 2^3$	0.016	0.016	0.383	1.44	50.58	66.24
$6^9 4^3 2^7$	0.016	0.015	0.319	7.06	152.82	633.23
$6^7 4^8 2^3$	0.016	0.015	0.295	140.323	135.42	315.34
$4^{15} 3^{17} 2^{29}$	0.031	0.017	0.741	65.8	705.921	1843.56
$4^1 3^{39} 2^{35}$	0.016	0.031	1.96	122.2	752.35	2143.23

sis H_0 when it is true and (II) we accept the null hypothesis H_0 when it is false. These two types of errors are conflicting means minimizing the probability of one increasing the probability of the other. In general there is more emphasis on not committing a type I error. When performing Welch's t-test, the p-value denotes the probability of type I error. The significant level α of a test is the highest p-value that can be accepted to reject H_0. Traditionally, $\alpha = 0.05$ is used during experimentation. We have conducted the Welch t-test at both $\alpha = 0.05$ and $\alpha = 0.01$.The results of the Welch t-test performed to compare average fitness of CAs/MCAs obtained by ABC-CAG and PWiseGen at $\alpha = 0.05$ and at $\alpha = 0.01$ are shown in Table 13. For space reason, we show the graphs depicting the t-values and p-values of only few benchmark problems at $\alpha = 0.05$ and at $\alpha = 0.05$ in Figure 6 and Figure 7, respectively.

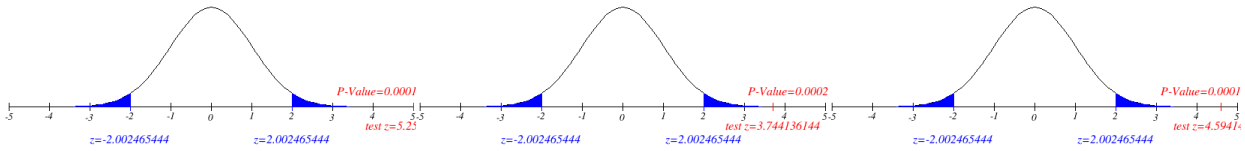
As discussed in Section 6.3.1, the performance of ABC-CAG and CASA is comparable in around 60% cases. So we calculated the standard deviation of the sizes of each CA obtained over 30 runs of TCAS, the printer case study and each of the 20 benchmark problem on both ABC-CAG and CASA to quantify the amount of dispersion or variation of the CA size obtained over these runs on the two meta-heuristic techniques. For

Table 13. Results of the Welch t-test to compare ABC-CAG and PWiseGen

Benchmark Problems	$\alpha = 0.05$				$\alpha = 0.01$			
	p (2-tailed)	t-critical	t-stat	df	p (2-tailed)	t-critical	t-stat	df
3^3	–	–	–	–	–	–	–	–
3^4	–	–	–	–	–	–	–	–
3^{13}	0.028		2.247	58	**0.028**		**2.247**	58
5^{10}	0.006		2.852	45	0.006		2.85	45
10^{20}	2.02×10^{-11}		8.355	56	2.02×10^{-11}		8.35	56
2^{100}	4.4×10^{-05}		4.416	58	4.44×10^{-05}		4.41	58
$4^5 3^4$	2.33×10^{-06}		5.25	57	2.33×10^{-06}		5.25	57
$5^1 3^8 2^2$	4.83×10^{-07}		5.716	54	2.43×10^{-07}		5.71	54
$7^2 6^2 4^2 3^2 2^2$	1.67×10^{-06}		5.33	58	1.67×10^{-06}		5.33	58
$8^2 7^2 6^2 5^2$	1.6×10^{-10}		7.86	54	1.6×10^{-10}		7.86	54
$6^4 4^5 2^7$	5.05×10^{-06}	2.00	5.04	56	5.05×10^{-06}	2.66	5.047	56
$5^1 4^4 3^{11} 2^5$	3.07×10^{-09}		7.24	48	3.07×10^{-09}		7.24	48
$6^1 5^1 4^6 3^8 2^3$	5.9×10^{-04}		3.65	53	5.9×10^{-04}		3.65	53
$6^2 4^9 2^9$	1.79×10^{-05}		4.96	35	1.7×10^{-05}		4.96	35
$6^5 5^5 3^4$	9.7×10^{-09}		6.95	55	4.04×10^{-09}		6.5	56
$7^1 6^1 5^1 4^5 3^8 2^3$	3.49×10^{-08}		6.37	57	3.49×10^{-08}		6.37	57
$6^9 4^3 2^7$	6.9×10^{-08}		6.193	57	6.92×10^{-08}		6.19	57
$6^7 4^8 2^3$	2.3×10^{-07}		5.889	56	2.3×10^{-07}		5.88	56
$4^{15} 3^{17} 2^{29}$	4.2×10^{-04}		3.744	57	0.0004		3.74	57
$4^1 3^{39} 2^{35}$	2.44×10^{-05}		4.59	57	2.44×10^{-05}		4.59	57



a) CA(3^{13}) b) CA(5^{10}) c) CA(10^{20})



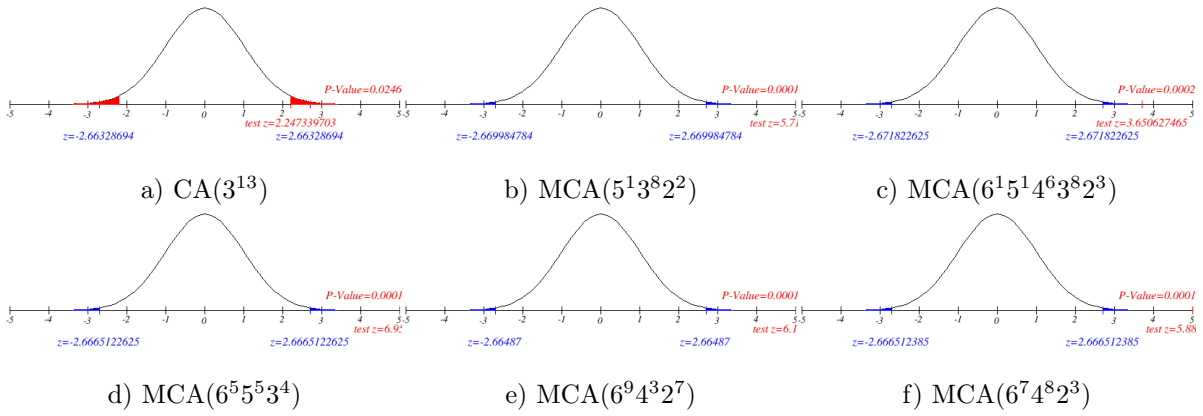
d) MCA($4^5 3^4$) e) MCA($4^{15} 3^{17} 2^{29}$) f) MCA($4^1 3^{39} 2^{35}$)

Figure 6. Results of the Welch t-test of the selected benchmark problems at $\alpha = 0.05$

TCAS and the printer case study, the size of CA generated in each of the 30 runs on ABC-CAG is the same. The same is true for CASA as well. Therefore, we plotted the average standard deviation of only the benchmark problems given in Table 6 when run multiple times on ABC-CAG and CASA as shown in Figure 8.

Analysis: It is evident from Table 13 and Figure 6 that at $\alpha = 0.05$, the value of t-stat $>$ t-critical and $p < \alpha$ for each benchmark problem. Similarly, from Table 13 and Figure 7, it can be seen that at $\alpha = 0.01$, the value of t-stat $>$ t-critical and $p < \alpha$ for each benchmark problem except CA (3^{13}). So, we reject the null hypothesis H_0 and conclude that there is a significant difference between ABC-CAG and PWiseGen.

From Figure 8, it can be seen that for most of the benchmark problems, the average standard deviation of the sizes of CA/MCA obtained over multiple runs of a problem instance on ABC-CAG is smaller than that of CASA. Even in the case of those problems where the best sizes generated by both ABC-CAG and CASA are the same, the standard deviation of ABC-CAG is smaller than



a) CA(3^{13}) b) MCA($5^1 3^8 2^2$) c) MCA($6^1 5^1 4^6 3^8 2^3$)

d) MCA($6^5 5^5 3^4$) e) MCA($6^9 4^3 2^7$) f) MCA($6^7 4^8 2^3$)

Figure 7. Results of the Welch t-test of the Selected Benchmark Problems at $\alpha = 0.01$

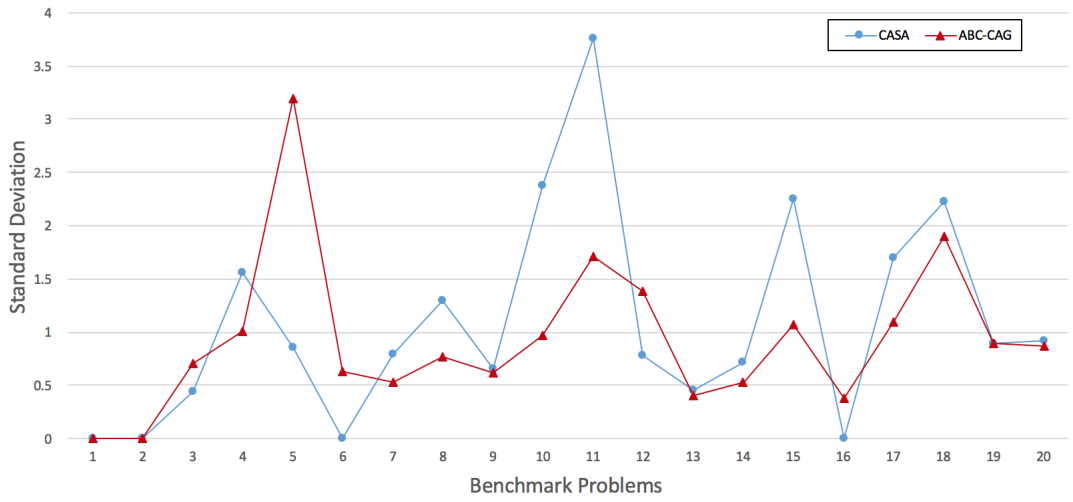


Figure 8. Average standard deviation of sizes of CAs obtained over multiple runs

CASA for 8 problems out of 10. On this basis it can be inferred that ABC-CAG is more stable as compared to CASA.

In summary, we can conclude that ABC-CAG performs better than greedy algorithms and most of the meta-heuristic techniques except TS where the results are comparable, and the outcome of statistical testing proves the validity of the results generated by ABC-CAG and the standard deviation shows that ABC-CAG is more stable as compared to CASA when run multiple times on the same problem instance.

7. Threats to Validity

An important threat to validity is that we use only 30 runs of the stochastic algorithms namely ABC-CAG, CASA and PWiseGen because of time and resource constraints. Though more runs are unlikely to change the qualitative answer to the research questions, they may affect the magnitude of the algorithmic differences.

8. Conclusion and Future Work

In this paper we have presented the Artificial Bee Colony – Covering Array Generator (ABC-CAG) that deals with the problem of constructing optimal covering arrays for pair-wise testing. The key feature of ABC-CAG is the integration of greedy and meta-heuristic algorithms which enable ABC-CAG to exploit the advantages of both techniques. Second, the use of the global best CA during onlooker bees' phase derives the solution

towards global best thereby improving the exploitation capability of onlooker bees. In addition to this, the use of smart test cases to replace the worst test case of the global best CA during the onlooker bees' phase further enhances the performance of ABC-CAG. Experiments conducted on various benchmark problems and a real world problem show that the proposed strategy generates smaller CA as compared to its greedy and meta-heuristic counterparts except TS where the results are comparable.

In future, we plan to construct CA for strength t greater than 2 and incorporate constraint handling feature in ABC-CAG.

References

[1] J.M. Glenford, *The art of software testing*. John Willey & Sons, 2011.

[2] D.M. Cohen, S.R. Dalal, A. Kajla, and G.C. Patton, "The automatic efficient test generator (AETG) system," in *Proceedings of the 5th International Symposium on Software Reliability Engineering*. IEEE, 1994, pp. 303–309.

[3] D.M. Cohen, S.R. Dalal, M.L. Fredman, and G.C. Patton, "The AETG system: An approach to testing based on combinatorial design," *IEEE Transactions on Software Engineering*, Vol. 23, No. 7, 1997, pp. 437–444.

[4] K. Burr and W. Young, "Combinatorial test techniques: Table-based automation, test generation and code coverage," in *Proceedings of the International Conference on Software Testing Analysis & Review*, San Diego, 1998.

[5] S.R. Dalal, A. Jain, N. Karunanithi, J. Leaton, C.M. Lott, G.C. Patton, and B.M. Horowitz, "Model-based testing in practice," in *Proceedings of the 21st international conference on Software engineering*. ACM, 1999, pp. 285–294.

[6] D.R. Kuhn, D.R. Wallace, and A.M. Gallo Jr, "Software fault interactions and implications for software testing," *IEEE Transactions on Software Engineering*, Vol. 30, No. 6, 2004, pp. 418–421.

[7] Y. Lei and K.C. Tai, "In-parameter-order: A test generation strategy for pairwise testing," in *Proceedings of the Third IEEE International High-Assurance Systems Engineering Symposium*. IEEE, 1998, pp. 254–261.

[8] Y.W. Tung and W.S. Aldiwan, "Automating test case generation for the new generation mission software system," in *Proceedings of the IEEE Aerospace Conference*, Vol. 1. IEEE, 2000, pp. 431–437.

[9] A. Hartman, T. Klinger, and L. Raskin, "IBM intelligent test case handler," *Discrete Mathematics*, Vol. 284, 2010, pp. 149–156.

[10] J. Arshem, Test vector generator (TVG), (2010). [Online]. https://sourceforge.net/projects/tvg/

[11] AllPairs, (2009). [Online]. http://sourceforge.net/projects/allpairs/

[12] J. Czerwonka, "Pairwise testing in the real world: Practical extensions to test-case scenarios," in *Proceedings of the 24th Pacific Northwest Software Quality Conference*, 2006, pp. 419–430.

[13] B. Jenkins, jenny: a pairwise testing tool, (2005). [Online]. http://burtleburtle.net/bob/math/jenny.html

[14] Z. Wang, B. Xu, and C. Nie, "Greedy heuristic algorithms to generate variable strength combinatorial test suite," in *The Eighth International Conference on Quality Software. QSIC'08*. IEEE, 2008, pp. 155–160.

[15] Z. Wang and H. He, "Generating variable strength covering array for combinatorial software testing with greedy strategy," *Journal of Software*, Vol. 8, No. 12, 2013, pp. 3173–3181.

[16] S.A. Abdullah, Z.H. Soh, and K.Z. Zamli, "Variable-strength interaction for t-way test generation strategy," *International Journal of Advances in Soft Computing & Its Applications*, Vol. 5, No. 3, 2013.

[17] M.F. Klaib, K.Z. Zamli, N.A.M. Isa, M.I. Younis, and R. Abdullah, "G2Way a backtracking strategy for pairwise test data generation," in *15th Asia-Pacific Software Engineering Conference, APSEC'08*. IEEE, 2008, pp. 463–470.

[18] K.Z. Zamli, M.F. Klaib, M.I. Younis, N.A.M. Isa, and R. Abdullah, "Design and implementation of a t-way test data generation strategy with automated execution tool support," *Information Sciences*, Vol. 181, No. 9, 2011, pp. 1741–1758.

[19] K.F. Rabbi, A.H. Beg, and T. Herawan, "MT2Way: A novel strategy for pair-wise test data generation," in *Computational Intelligence and Intelligent Systems*. Springer, 2012, pp. 180–191.

[20] K. Rabbi, S. Khatun, C.Y. Yaakub, and M. Klaib, "EPS2Way: an efficient pairwise test data generation strategy," *International Journal of New Computer Architectures and their Applications (IJNCAA)*, Vol. 1, No. 4, 2011, pp. 1099–1109.

[21] Z. Zhang, J. Yan, Y. Zhao, and J. Zhang, "Generating combinatorial test suite using combinato-

rial optimization," *Journal of Systems and Software*, Vol. 98, 2014, pp. 191–207.

[22] R. Kuhn, Advanced combinatorial testing system (ACTS), National Institute of Standards and Technology, (2011). [Online]. http://csrc.nist.gov/groups/SNS/acts/documents/comparison-report.html#acts

[23] T. Shiba, T. Tsuchiya, and T. Kikuno, "Using artificial life techniques to generate test cases for combinatorial testing," in *Proceedings of the 28th Annual International Computer Software and Applications Conference*. IEEE, 2004, pp. 72–77.

[24] X. Chen, Q. Gu, J. Qi, and D. Chen, "Applying particle swarm optimization to pairwise testing," in *IEEEProceedings of the 34th Annual Computer Software and Applications Conference*. IEEE, 2010, pp. 107–116.

[25] L. Gonzalez-Hernandez, N. Rangel-Valdez, and J. Torres-Jimenez, "Construction of mixed covering arrays of variable strength using a tabu search approach," in *Combinatorial Optimization and Applications*. Springer, 2010, pp. 51–64.

[26] L. Gonzalez-Hernandez, N. Rangel-Valdez, and J. Torres-Jimenez, "Construction of mixed covering arrays of strengths 2 through 6 using a tabu search approach," *Discrete Mathematics, Algorithms and Applications*, Vol. 4, No. 03, 2012, p. 1250033.

[27] H. Avila-George, J. Torres-Jimenez, V. Hernández, and L. Gonzalez-Hernandez, "Simulated annealing for constructing mixed covering arrays," in *Distributed Computing and Artificial Intelligence*. Springer, 2012, pp. 657–664.

[28] B.S. Ahmed, K.Z. Zamli, and C. Lim, "The development of a particle swarm based optimization strategy for pairwise testing," *Journal of Artificial Intelligence*, Vol. 4, No. 2, 2011, pp. 156–165.

[29] B.J. Garvin, M.B. Cohen, and M.B. Dwyer, "Evaluating improvements to a meta-heuristic search for constrained interaction testing," *Empirical Software Engineering*, Vol. 16, No. 1, 2011, pp. 61–102.

[30] J.D. McCaffrey, "Generation of pairwise test sets using a genetic algorithm," in *Proceedings of the 33rd Annual IEEE International Computer Software and Applications Conference*, Vol. 1. IEEE, 2009, pp. 626–631.

[31] P. Flores and Y. Cheon, "PWiseGen: Generating test cases for pairwise testing using genetic algorithms," in *Proceedings of the International Conference on Computer Science and Automation Engineering*, Vol. 2. IEEE, 2011, pp. 747–752.

[32] P. Bansal, S. Sabharwal, S. Malik, V. Arora, and V. Kumar, "An approach to test set generation for pair-wise testing using genetic algorithms," in *Search Based Software Engineering*. Springer, 2013, pp. 294–299.

[33] B.S. Ahmed, T.S. Abdulsamad, and M.Y. Potrus, "Achievement of minimized combinatorial test suite for configuration-aware software functional testing using the cuckoo search algorithm," *Information and Software Technology*, Vol. 66, 2015, pp. 13–29.

[34] T. Mahmoud and B.S. Ahmed, "An efficient strategy for covering array construction with fuzzy logic-based adaptive swarm optimization for software testing use," *Expert Systems with Applications*, Vol. 42, No. 22, 2015, pp. 8753–8765.

[35] D. Karaboga, "An idea based on honey bee swarm for numerical optimization," Erciyes University, Engineering Faculty, Computer Engineering Department, Tech. Rep. TR-06, 2005.

[36] D. Karaboga and B. Basturk, "Artificial bee colony (ABC) optimization algorithm for solving constrained optimization problems," in *Foundations of Fuzzy Logic and Soft Computing*. Springer, 2007, pp. 789–798.

[37] A.S. Hedayat, N.J.A. Sloane, and J. Stufken, *Orthogonal arrays*. Springer Science & Business Media, 2012.

[38] M.B. Cohen, P.B. Gibbons, W.B. Mugridge, and C.J. Colbourn, "Constructing test suites for interaction testing," in *Proceedings of the 25th International Conference on Software Engineering*. IEEE, 2003, pp. 38–48.

[39] G. Sherwood, Testcover.com, (2006). [Online]. http://testcover.com/

[40] A.W. Williams, "Determination of test configurations for pair-wise interaction coverage," in *Testing of Communicating Systems*. Springer, 2000, pp. 59–74.

[41] A. Hartman, "Software and hardware testing using combinatorial covering suites," in *Graph theory, combinatorics and algorithms*. Springer, 2005, pp. 237–266.

[42] N. Kobayashi, T. Tsuchiya, and T. Kikuno, "A new method for constructing pair-wise covering designs for software testing," *Information Processing Letters*, Vol. 81, No. 2, 2002, pp. 85–91.

[43] D. Karaboga and B. Basturk, "A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm," *Journal of global optimization*, Vol. 39, No. 3, 2007, pp. 459–471.

[44] E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm intelligence: from natural to artificial systems*. Oxford University Press, 1999.

[45] O.B. Haddad, A. Afshar, and M.A. Mariño, "Honey-bees mating optimization (HBMO) algorithm: a new heuristic approach for water resources optimization," *Water Resources Management*, Vol. 20, No. 5, 2006, pp. 661–680.

[46] D. Teodorović and M. Dell'Orco, "Bee colony optimization – a cooperative learning approach to complex transportation problems," in *Advanced OR and AI Methods in Transportation: Proceedings of 16th Mini–EURO Conference and 10th Meeting of EWGT*. Poznań: Publishing House of the Polish Operational and System Research, 2005, pp. 51–60.

[47] H. Drias, S. Sadeg, and S. Yahi, "Cooperative bees swarm for solving the maximum weighted satisfiability problem," in *Computational Intelligence and Bioinspired Systems*. Springer, 2005, pp. 318–325.

[48] G. Li, P. Niu, and X. Xiao, "Development and investigation of efficient artificial bee colony algorithm for numerical function optimization," *Applied soft computing*, Vol. 12, No. 1, 2012, pp. 320–332.

[49] D. Jeya Mala, V. Mohan, and M. Kamalapriya, "Automated software test optimisation framework – an artificial bee colony optimisation-based approach," *IET Software*, Vol. 4, No. 5, 2010, pp. 334–348.

[50] G. Zhu and S. Kwong, "Gbest-guided artificial bee colony algorithm for numerical function optimization," *Applied Mathematics and Computation*, Vol. 217, No. 7, 2010, pp. 3166–3173.

[51] J. Stardom, "Metaheuristics and the search for covering and packing arrays," Ph.D. dissertation, Simon Fraser University, 2001.

[52] B. Kazimipour, X. Li, and A. Qin, "A review of population initialization techniques for evolutionary algorithms," in *IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2014, pp. 2585–2592.

[53] D.R. Kuhn and V. Okun, "Pseudo-exhaustive testing for software," in *Proceedings of the 30th Annual IEEE/NASA Software Engineering Workshop*. IEEE, 2006, pp. 153–158.

[54] Pairwise testing, (2016). [Online]. http://www.pairwise.org/

[55] P. Flores, PWiseGen, (2010). [Online]. https://code.google.com/p/pwisegen/

[56] A. Arcuri and L. Briand, "A practical guide for using statistical tests to assess randomized algorithms in software engineering," in *Proceedings of the 33rd International Conference on Software Engineering*. IEEE, 2011, pp. 1–10.

e-Informatica Software Engineering Journal, Volume 10, Issue 1, 2016, pages: 31–49, DOI 10.5277/e-Inf160102

Reducing the Number of Higher-order Mutants with the Aid of Data Flow

Ahmed S. Ghiduk[a]

[a] *Dept. of IT, College of Computers and Information Technology, Taif University, Saudi Arabia, Department of Mathematics and Computer Science, Faculty of Science, Beni-Suef University, Egypt*

asaghiduk@tu.edu.sa

Abstract

Higher-order mutants are created by injecting two or more mutations into the original program, while first-order mutants are generated by seeding single faults in the original program. Mutant generation is a key stage of mutation testing which is computationally very expensive, especially in the case of higher-order mutants. Although many mutation testing techniques have been developed to construct the first-order mutants, a very small number of techniques have been presented to generate the higher-order mutants because of the exponential growth of the number of higher-order mutants, and the coupling effect between higher-order and first-order mutants. To overcome the exponential explosion in the number of higher-order mutants considered, this paper introduces a new technique for generating a reduced set of higher-order mutants. The proposed technique utilizes a data-flow analysis to decrease the number of mutation points through the program under test and consequently reduce the number of higher-order mutants. In this technique only positions of *defs* and *uses* are considered as locations to seed the mutation. The generated set of higher-order mutants consists of a reduced number of mutants, which reduces the costs of higher-order mutation testing. In addition, the proposed technique can generate the higher-order mutants directly without generating the first-order mutants or by combining two or more first-order mutants. A set of experiments are conducted to evaluate the effectiveness of the proposed technique. The results of the conducted experiments are presented and compared with the results of the related work. These results showed that the proposed technique is more effective than the earlier techniques in generating higher-order mutants without affecting the efficiency of mutation testing.

Keywords: mutation testing, first-order mutants, higher-order mutants, data-flow analysis

1. Introduction

Higher-order mutants (*HOMs*) are complex mutants, which are produced by inserting two or more mutations in the original program [1]. The space of higher-order mutants is wider than the space of first-order mutants (*FOMs*) [2]. Mutation testing has been developed by DeMillo et al. [3] and Hamlet [4] to find test inputs to kill the seeded mutants in the program under test [5]. The motivation of mutation testing is that the injected faults represent errors that programmers often create. Although mutation testing is a very powerful software testing technique, it contains many computationally expensive phases such as mutant generation and mutant execution.

Many mutation testing techniques have been developed to consider the first-order mutants [6]. Higher-order mutation testing techniques are proposed by Jia and Harman [1] and used to study the interactions between defects and their impact on software testing for fault detection.

Although mutation testing is an effective high automation technique to assess the quality of the test data, it has three main limitations. These limitations are large number of mutants, realism, and the equivalent mutant problem [7,8]. A large number of mutants will be generated during the

mutant generation phase of mutation testing even for small programs. For example, a program consists of one statement such as $return\ x + y$; (where x and y are integers) can be mutated into many different mutants: $return\ x-y$; $return\ x*y$; $return\ x/y$; $return\ x + y + +$; $return\ \ - x + y$; $return\ x + -y$; $return\ 0 + y$; $return\ x + 0$; ..., etc. This problem leads to a very high execution cost because the test cases are executed not only on original program but also on each mutant. For example, if a program under test has 200 mutants and 150 test cases, it requires $(1 + 200) * 150 = 30150$ executions with their corresponding results [7]. In addition, because mutants are generated by single and simple syntactic changes, they don't represent realistic faults and 90% of real faults are complex [2]. In fact, several mutation operators can generate equivalent mutants which have the same behavior as the original program and need additional human effort to kill [9]. These limitations are resulting from the used method to generate mutants.

Many techniques have been proposed to reduce the number of mutants. The first approach to reduce the number of mutants is Mutant Sampling approach proposed by Acree [10] and Budd [11]. In addition, Bluemke and Kulesza [12] explored the reduction of computational costs of mutation testing by randomly sampling mutants. This approach randomly selects a small subset of mutants from the entire set. Mutant Sampling is valid with a value higher than 10% of mutants [7]. Agrawal et al. [13] and Mathur [14] proposed an approach to reduce the number of mutation operators which can lead to reduced number of mutants. Offutt et al. [15, 16] used the same idea and called it Selective Mutation. This approach selects a small set of operators that generate a subset of all possible mutants without losing test effectiveness. Husain [17] applied clustering algorithms to select a subset of mutants.

Second-order Mutation Testing [9, 18–20], in particular, and Higher-Order Mutation Testing [1, 2, 21, 22] in general, are the most promising solutions to reduce the number of mutants [7]. The number of generated mutants can be reduced to about 50% by combining two first-order mutants to generate a second-order mutant or by using subsuming higher-order mutants algorithms [7]. Previous work employed different methods for reducing the number of higher-order mutants. Polo et al. [20] proposed three methods: 1) *RandomMix* which couples randomly selected mutation operators, 2) *LastToFirst* which combines *FOMs* in order from the last operator to the first one, and 3) *DifferentOperator* which combines different *FOMs* mutation operators. Madeyski et al. [9] proposed two methods: 1) *JudyDiffOp* which combines different *FOMs* mutation operators, and 2) *NeighPair* which combines *FOMs* which are close to each other. Although, these techniques have the ability to reduce the number of mutants, the number of mutants can still grow exponentially. From the above discussion, the higher-order mutant generation problem needs a lot of effort.

Data-flow testing is essential because it augments control-flow testing. It aims at creating more efficient and targeted test suites. Data flow testing is concerned not only with the definitions and uses of variables, but also with sub-paths from definitions to statements where those definitions are used [23, 24]. A family of data flow criteria [25] have been proposed and successfully applied in many software testing activities [26]. Unfortunately, this family of criteria has never been applied in mutation testing as basis for generating the *HOMs* or reducing the number of these mutants.

The main contributions of this paper are: 1) introducing a data-flow based approach for generating higher-order mutants; In this approach only locations of *def* points and *use* points are considered as locations to seed the mutation. A second-order mutant contains two mutations, the first mutation at the *def* point and the second mutation at the *use* point and the two points belong to the same *def-use* pairs. 2) Using the proposed approach to perform a set of empirical studies to answer the following research questions:

- **RQ1**: How effective is data flow in aiding the generation of higher-order mutants?
- **RQ2**: How effective is the proposed technique in reducing the number of higher-order mutants?

Table 1. An example of mutants

Original Program	Mutants		
	FOM1	*FOM2*	*SOM*

`if (min < max)`
`{`
`max = min + max;`
`min = max − min;`
`}`
	`if (min > max)`
	`{`
	`max = min + max;`
	`min = max − min;`
	`}`
		`if (min < max)`
		`{`
		`max = min − max;`
		`min = max − min;`
		`}`
			`if (min > max)`
			`{`
			`max = min − max;`
			`min = max − min;`
			`}`

The rest of this paper is organized as follows. Section 2 gives some basic concepts and definitions. Section 3 describes the proposed technique for generating a reduced set of higher-order mutants. Section 4 describes the empirical studies performed to evaluate the proposed technique. Section 5 gives a discussion of how the present paper differs from the related ones. Section 6 gives the conclusion and future work.

2. Background

This section introduces some basic concepts that will be used throughout this work.

2.1. Mutation Testing

The input parameters to the mutation testing are: the tested program P, a set of mutation operators, and a set of test inputs, T. Initially, the program under test must be executed with the test set T to show that it is correct and produces the desired outputs. If not, then the program under test contains faults, which should be corrected before resuming the process.

The next stage is generating a set of mutants of the tested program by seeding faults in it. The seeded faults are generated by applying the mutation operators. The transformation that creates a mutant from the original program is known as a mutation operator [1]. A mutant is generated by making one or more small changes (faults) into the original program. *FOMs*, which are created by the injection of unique faults in the tested program, are created by applying mutation operators only once. *HOMs*, which are created by injecting two or more mutations into the original program, are created by applying mutation operators more than once. Table 1 shows two first-order mutants (*FOM1* and *FOM2*) generated by changing the "<" operator in the original program into the ">" operator in *FOM1* and changing the "+" operator in the original program into the "−" operator in *FOM2*. In addition, Table 1 gives a second-order mutant, *SOM*, created by coupling the two first-order mutants *FOM1* and *FOM2*.

In Traditional Mutation Testing (Strong Mutation), each mutant will be executed using a test set T. If the result of executing a mutant is different from the result of executing the original program for any test case in T, then the mutant is *killed* otherwise it is *survived*. The adequacy level of the test set T can be measured by a mutation score [27] that is computed in terms of the number of mutants killed by T as follows.

$$\mathrm{MS}(P,T) = \frac{\text{Number of Killed Mutants}}{\text{Total Number of Mutants} - \text{Number of Equivalent Mutants}} \quad (1)$$

Howden [6] proposed Weak Mutation [28] to optimize the execution of Strong Mutation. Weak Mutation checks the result of a mutant immediately after the mutated component is executed with the resulting execution of the original component to say is the mutant is killed or not.

2.2. Higher-order Mutation Testing

Higher-order mutation (*HOM*) testing is a generalization of traditional mutation testing. Higher-order mutants are constructed by inserting two or more changes into the program under

test or by combining two or more first-order mutants. Higher-order mutants can be classified into six categories based on the way that they are *Coupled* and *Subsuming* [1]. Coupled means: complex errors are coupled to simple errors, and the coupling effect hypothesis states that test input sets that detect simple types of faults are sensitive enough to detect more complex types of faults [3]. A subsuming *HOM* is one in which the first-order constituent mutants partly mask one another. Therefore, a subsuming *HOM* is harder to kill than the first-order mutants from which it is constructed.

2.3. Data-flow Analysis

The structure of the program can be represented by the control-flow graph. A control-flow graph $G = (N, E)$ with a unique entry node n_0 and a unique exit node n_k, consists of a set N of nodes, where each node represents a statement, and a set E of directed edges, where a directed edge $e = (n, m)$ is an ordered pair of two adjacent nodes, called *tail* and *head* of e, respectively [29, 30].

Data-flow analysis identifies all definition-use (*def-use*) pairs for any variable v of the program under test. A *def-use* is the order triple (d, u, v) in which statement d contains a *definition* for variable v and statement u contains a *use* of v that can be reached by d over some paths in the program under test [23, 24]. A variable is defined in a statement when its value is assigned or changed. A variable is used in a statement when its value is utilized in a statement and not changed. A *predicate use* (*p-use*) for a variable indicates the *use* of the variable in a predicate. A *computational use* (*c-use*) indicates the *use* of the variable in a computation.

3. A Proposed Higher-Order Mutant Generation Technique

This section describes the proposed technique for generating a reduced set of higher-order mutants. This technique utilizes the concepts of data-flow analysis of the program to reduce the number of mutation positions through the tested program which will reduce the number of higher-order mutants. The proposed technique is based on data-flow analysis [31] and Muclipse tool [32, 33]. The proposed technique consists of the following main modules.

1. Analysis Module.
2. Mutant Generation Module.
3. Mutant Filtering Module.

These modules are described in more detail below.

3.1. Analysis Module

This module applies the data-flow analysis procedure proposed by F.E. Allen and J. Cocke [31] to find all definition-use pairs (all definition-c-use and all definition-p-use) in the tested program. This module reads the Java source code of the program under test, builds the control-flow graph of the tested program, and identifies all definition-use pairs for each method in this Java program individually. The proposed technique reduces the number of *def-p-use* pairs by combining all *def-p-use* pairs which have the same *def* point (i.e., beginning statement of the edge *p-u*) into one *def-p-use* pair where the *use* point (end statement *u*) does not contain any *uses*. The outputs of this phase are passed to the Mutant Generation Module (step 2).

For the Java example program shown in Table 2, this phase finds all definition-c-uses and all definition-p-uses pairs in the tested program by applying the proposed data-flow analysis procedure. The Analysis Module finds 10 *def-c-use* pairs and 20 *def-p-use* pairs for method Midnum(). Table 3 shows all *def-use* pairs of method Midnum() of the example program given in Table 2. In Table 3, a *def-c-use* (*d, cu, x*) consists of the statement *"d"* which contains a definition for variable *"x"* which is used in a computation statement *"cu"* and a *def-p-use* (*d, p-u, x*) consists of the statement *"d"* which contains a definition for variable *"x"* which is used through the edge *"p-u"* which starts at statement *"p"* and ends at statement *"u"*. Then, the proposed technique reduces the number of *def-p-use* pairs by merging all *def-p-use* pairs which have the same *def* point.

Table 2. Java example program

```
1. package edu.ncsu.csc326.paperHOM_dataflow;
2. public class Mid1 {
3.   private int num1, num2, num3, Mid;
4.   public Mid1(){
5.     }
6.   public void setNum1(int x){
7.         num1 = x;
8.     }
9.   public void setNum2(int x){
10.        num2 = x;
11.    }
12.  public void setNum3(int x){
13.        num3 = x;
14.    }
15.  public int getMid(){
16.        return Mid;
17.    }
18.  public void Midnum()
19.    {
20.        int x, y, z;
21.        int mid;
22.        x = num1;
23.        y = num2;
24.        z = num3;
25.        mid = z;
26.        if(y<z)
27.        {
28.          if(x<y)
29.  {
30.    mid = y;
31.  }
32.  else
33.  {
34.    if(x<z)
35.    {
36.      mid = x;
37.    }
38.   }
39.  }
40.  else
41.  {
42.   if(x>=y)
43.   {
44.     mid = y;
45.   }
46.   else
47.   {
48.   if(x>z)
49.   {
50.     mid = x;
51.   }
52.  }
53. }
54. Mid = mid;
55. }
56. }
```

Therefore, two *def-p-use* pairs such as (*22, 28-39, x*) and (*22, 28-32, x*) will merge to one *def-p-use* (*22, 28, x*). The proposed technique reduces the 20 *def-p-uses* to 10 *def-p-uses*. Table 3 gives the new *def-p-uses* pairs. The list of *def-c-uses* and the reduced *def-p-uses* are passed to the Mutant Generation Module.

3.2. Mutant Generation Module

This module uses the data collected by the Analysis Module to generate the set of higher-order mutants. This module considers only the locations of *def* points and *use* points as locations to seed mutation. This module uses the set of method-level operators proposed by Y. Ma and J. Offutt [34] using the Muclipse tool [32,33] to generate the first-order mutants. Table 4 shows this set of mutation operators. This module requires three inputs to perform its task (i.e., generating a set of higher-order mutants): the first input is the Java source code of the program under test, the second is the set of mutation operators given in Table 4, and the third is the set of mutation locations which is the location of *def* and *use* statements (i.e., set of *defs* ∪ set of *uses*) in the tested program. For the example program given in Table 2, the set of mutation locations is $\{22, 23, 24, 25, 30, 36, 44, 50\} \cup \{36, 50, 30, 44, 25, 54, 28, 34, 42, 48, 26\} = \{22, 23, 24, 25, 26, 28, 30, 34, 36, 42, 44, 48, 50, 54\}$.

For generating first-order mutants, the proposed technique needs a set of mutation operators, a set of mutation locations, and the program to be mutated. For the example program, the set of mutation locations is the set of *defs* locations and *uses* locations = $\{22, 23, 24, 25, 26, 28, 30, 34, 36, 42, 44, 48, 50, 54\}$ and the set of mutation operators is the 16 operators given in Table 4. The following pseudocode presents the proposed *DataFolwBasedFOM* algorithm for generating a reduced list of *FOMs*.

Table 3. All def-uses of method Midnum() of the example program

#	def-c-uses	def-p-uses		Reduced def-p-uses
1	(22,36,x)	(22,28-39,x)	(22,28-32,x)	(22,28,x)
2	(22,50,x)	(22,34-35,x)	(22,34-38,x)	(22,34,x)
3	(23,30,y)	(22,42-43,x)	(22,42-46,x)	(22,42,x)
4	(23,44,y)	(22,48-49,x)	(22,48-52,x)	(22,48,x)
5	(24,25,z)	(23,26-27,y)	(23,26-40,y)	(23,26,y)
6	(25,54,mid)	(23,28-29,y)	(23,28-32,y)	(23,28,y)
7	(30,54,mid)	(23,42-43,y)	(23,42-46,y)	(23,42,y)
8	(36,54,mid)	(24,26-27,z)	(24,26-40,z)	(24,26,z)
9	(44,54,mid)	(24,34-35,z)	(24,34-38,z)	(24,34,z)
10	(50,54,mid)	(24,48-49,z)	(24,48-52,z)	(24,48,z)

Table 4. Set of mutation operators

Category	Mutation Operator	Description
AO	AORB	A binary arithmetic operator is replaced by another one.
	AORU	An unary arithmetic operator is replaced by another one.
	AORS	A short-cut arithmetic operator is replaced by another one.
	AOIS	A short-cut arithmetic operator is inserted into the program.
	AOIU	An unary arithmetic operator is inserted into the program.
	AODS	A short-cut arithmetic operator is deleted from the program.
	AODU	An unary arithmetic operator is deleted from the program.
RO	ROR	A relational operator is replaced by another one.
CO	COR	A binary conditional operator is replaced by another one.
	COI	An unary conditional operator is inserted into the program.
	COD	An unary conditional operator is deleted from the program.
SO	SOR	A shift operator is replaced by another one.
LO	LOR	A binary logical operator is replaced by another one.
	LOI	An unary logical operator is inserted into the program.
	LOD	An unary logical operator is deleted from the program.
AS	ASRS	A short-cut assignment operator is replaced by another one.

```
Algorithm DataFolwBasedFOM(program,
          mutationPoints[], operators[])
LET firstOrderMutants be an empty list
WHILE mutationPoints.size() > 0 DO
  WHILE !(oprators.empty()) DO
    op = oprators.select();
    mp = mutationPoints.select();
    newMutant = program.mutate(op, mp);
    firstOrderMutants.update(newMutant);
  ENDWHILE
ENDWHILE
RETURN firstOrderMutants;
```

The function *Operator.select*() uses different procedures to select an operator such as 1) not selected yet, 2) different operator, and 3) different category and the function *mutationPoints.select*() selects a not selected yet mutation position.

A second-order mutant contains two mutations, the first mutation at the *def* position and the second mutation at the *use* position and the two positions belong to the same *def-use* pairs. Therefore, the set of second-order mutants is the intersection between the set of first-order mutants at *defs* locations with the set of first-order mutants at *uses* locations. To generate second-order mutants, the proposed technique inserts two mutations into the original pro-

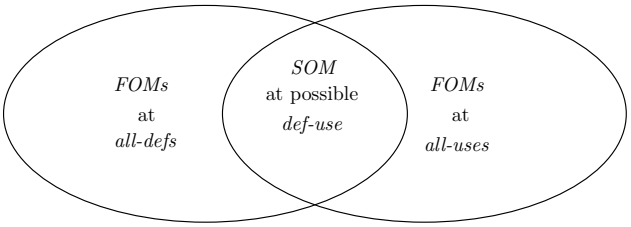


Figure 1. Mathematical representation of *FOM* and *SOM*

gram at the *def* and the *use* locations of the same *def-use* pairs. It can also merge the two first-order mutants at *def* and *use* positions of the same *def-use* pairs to construct a second-order mutant. Figure 1 shows a mathematical representation for the first-order mutants and the second-order mutants.

According to the above description the proposed technique needs only mutation positions, mutation operators, and the program to be mutated to generate higher-order mutants without needing the first-order mutants. The proposed technique selects one of the elements of the set of all *def-use* pairs and seeds this element with two mutation operators: one operator is applied at the *def* location and the second operator is applied at the *use* location. The following pseudocode presents the proposed *DataFolwBasedSOM* algorithm for generating the reduced list of second-order mutants.

```
Algorithm DataFolwBasedSOM(program,
         allDefUsePairs[], operators[])
LET secondOrderMutants be an empty list
WHILE allDefUsePairs.size()>0 DO
  WHILE !(oprators.empty()) DO
    op1 = oprators.select();
    op2 = oprators.select();
    du= allDefUsePairs.select();
    newMutant=program.mutate(op1, op2,
                             du);
    secondOrderMutants.update(newMutant);
  ENDWHILE
ENDWHILE
RETURN secondOrderMutants;
```

The function *Operator.select*() uses different procedures to select two operators such as 1) not selected yet, 2) different operator, and 3) different category and the function *allDefUsePairs.select*() selects a not selected yet *def-use* pairs to be a mutation point.

Table 5 (a) presents an example for second-order mutant of the example program at the *def-c-use* (*22, 36, x*) and Table 5 (b) presents an example for second-order mutant of the example program at the *def-p-use* (*24, 48, z*).

For generating higher-order mutants of even order greater than the second-order, the proposed technique applies the *DataFolwBasedSOM* algorithm more than one time with a change of the input program to the output or mutated program of the previous cycle. To generate higher-order mutants of odd order greater than the second order, the proposed technique applies the *DataFolwBasedSOM* algorithm more than one time with a change of the input program to the output or mutated program of the previous cycle in such a way that in the last cycle the algorithm seeds one mutation operator at *def* or *use* location only.

For example, for generating fourth-order mutants the technique applies the *DataFolwBasedSOM* algorithm two times in such a way that the inputs of the second cycle are the mutated programs (second-order mutants) of the first cycle. To generate third-order mutants, the technique applies the *DataFolwBasedSOM* algorithm two times such that the inputs of the second cycle are the mutated programs of the first cycle (second-order mutants) restricting the function *program.mutate*(*op1, op2, du*) to seed one operator at the *def* location or at the *use* location only. Table 6 gives examples for third-order (*3OMs*) and fourth-order (*4OMs*) mutants of the example program given in Table 2.

Table 5. An example for second-order mutant of the example program

18. public void Midnum()	38. }	18. public void Midnum()	38. }
19. {	39. }	19. {	39. }
20. int x, y, z;	40. else	20. int x, y, z;	40. else
21. int mid;	41. {	21. int mid;	41. {
22. **x = ++num1;**	42. if(x>=y)	22. x = num1;	42. if(x>=y)
23. y = num2;	43. {	23. y = num2;	43. {
24. z = num3;	44. mid = y;	24. **z = num3++;**	44. mid = y;
25. mid = z;	45. }	25. mid = z;	45. }
26. if(y<z)	46. else	26. if(y<z)	46. else
27. {	47. {	27. {	47. {
28. if(x<y)	48. if(x>z)	28. if(x<y)	48. **if(x<z)**
29. {	49. {	29. {	49. {
30. mid = y;	50. mid = x;	30. mid = y;	50. mid = x;
31. }	51. }	31. }	51. }
32. else	52. }	32. else	52. }
33. {	53. }	33. {	53. }
34. if(x<z)	54. Mid = mid;	34. if(x<z)	54. Mid = mid;
35. {	55. }	35. {	55. }
36. **mid *= x;**	56. }	36. mid = x;	56. }
37. }		37. }	
(a) SOM at du-pair (22,36,x)		(b) SOM at du-pair (24,48,z)	

3.3. Mutant Filtering Module

This module eliminates any useless mutants from the generated set of higher-order mutants. This module uses some criteria to divide the mutants into two categories: the first category is the target set of *HOMs* and the second one is the set of useless mutants. These criteria are:

1. Redundant mutants: the repeated mutants which were generated before.
2. First-order mutants: this happens if the mutation location of the *SOM* refers to the same position of the *FOM* (i.e., the same arithmetic operator in the same statement). This happens if the *def* location and *use* location are in the same statement. For example in the following loop:

```
1 i = 0;
2 sum = 0;
3 while(i < 10)
4     sum = sum + i;
```

   In the above code, the *def-use* (*4, 4, sum*) is a *def-use* at statement 4 for variable *sum*. In this *def-use* pairs, the *def* location and the *use* location are the same. Therefore, the proposed algorithm can generate a first-order mutant by changing the addition operator " + " to the division operator " / " and changing the division operator " / " to the addition operator " + ".
3. Equivalent mutants: this module can be supported by a technique for identifying the equivalent mutants to remove it. In our experiments, equivalent mutants are manually identified.

4. Empirical Studies

This section describes the empirical studies performed to evaluate the proposed technique. Two empirical studies were conducted: the first study aims to investigate the efficiency of data flow in aiding the generation of higher-order mutants and reducing their number as well; the second study aims to demonstrate that the proposed mutants do not lead to a substantial loss in the effectiveness of the method.

4.1. Empirical Study #1

4.1.1. Setup of Empirical Study #1

Prototype: Figure 2 gives the architecture of the prototype *HOMG*, which consists of three

Table 6. An example for third and fourth order mutants of the example program

```
18. public void Midnum()
19. {
20.  int x, y, z;
21.  int mid;
22.  x = ++num1;
23.  y = num2;
24.  z = num3;
25.  mid = z;
26.  if(y<z)
27.  {
28.   if(x<y)
29.   {
30.    mid = y++;
31.   }
32.   else
33.   {
34.    if(x<z)
35.    {
36.     mid *= x;
37.    }
38.   }
39.  }
40.  else
41.  {
42.   if(x>=y)
43.   {
44.    mid = y;
45.   }
46.   else
47.   {
48.   if(x>z)
49.   {
50.    mid = x;
51.   }
52.  }
53. }
54.  Mid = mid;
55.  }
56. }
```

(a) 3OM at du-pairs (22,36,x), and (30,54,mid)

```
18. public void Midnum()
19. {
20.  int x, y, z;
21.  int mid;
22.  x = num1;
23.  y = num2;
24.  z = num3++;
25.  mid = z;
26.  if(y<z)
27.  {
28.  if(x<y)
29.  {
30.   mid = ++y;
31.  }
32.  else
33.  {
34.  if(x<z)
35.  {
36.   mid = x;
37.  }
38.   }
39.  }
40.  else
41.  {
42.  if(x>=y)
43.  {
44.   mid = y;
45.  }
46.  else
47.  {
48.  if(x<z)
49.  {
50.   mid = x;
51.  }
52. }
53. }
54. Mid = -mid;
55. }
56. }
```

(b) 4OM at du-pair (24,48,z) and (30,54,mid)

modules: an analysis module, a mutant generation module, and a mutants filtering module. This prototype is based on the proposed technique which is presented in Section 3.

Subject Programs: A set of Java programs was selected from the previous studies for conducting an empirical study to evaluate the proposed technique. The set of subject programs contains some common programs which are often used as benchmarks in many software testing studies. This set of programs is *triangle*, *mid*, *power*, *remainder*, and three synthetic programs with different and complex structures.

Table 7 presents the details of the subject programs: the first column, *Subject Program*, presents a designated title of the program under test; the second column, *Reference*, presents some of the previous studies which used this set of subject programs; and the third column, *Scale*, presents the number of lines of code, classes, and methods in the subject program.

Procedure: the empirical study is conducted as follows.

1. Run *Muclipse* tool on the program to be mutated (original program) to generate *FOMs*. Because *Muclipse* cannot generate second-order mutants mutants the Muclipse tool was run on each first-order mutant to generate all possible second-order mutants. This set of second-order mutants is used for comparing the *LastToFirst* Algorithm, *DifferentOperators* Algorithm, and our proposed Algorithm.
2. Run the analysis module of our technique to find all *def-use* pairs of the original program.
3. Run the mutants generation module according to the *DataFolwBasedSOM* algorithm. Then the useless mutants are removed.

4.1.2. Objectives of Study #1

The study procedure to measure the efficiency of our proposed technique in generating the second-order mutants was applied. This study addresses the following research questions:

- **RQ1**: How effective is data flow in aiding the generation of higher-order mutants?
- **RQ2**: How effective is the proposed technique in finding a reduced set of higher-order mutants?

4.1.3. Results and Discussion of Study #1

To answer the first research question **RQ1**, the *DataFolwBasedFOM* algorithm to find the first-order mutants was applied. According to the

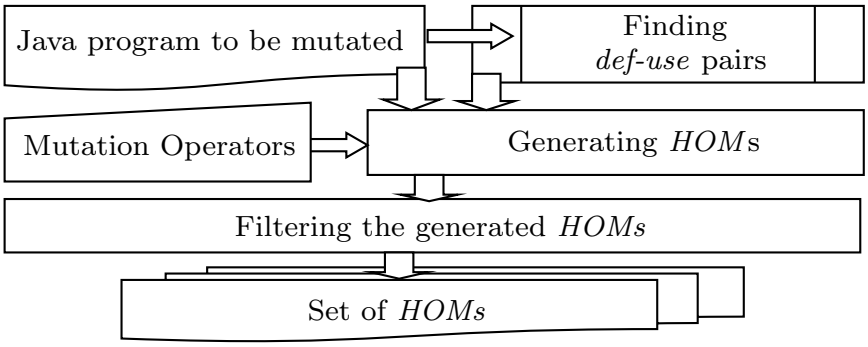


Figure 2. The architecture of the prototype of *HOMG*

Table 7. Subject programs

# Subject Program	Reference	Scale
P#1. Triangle	[2, 20, 26, 35]	73 LOC, 1 C, 6 M
P#2. Mid	[20, 26, 36]	61 LOC, 1 C, 6 M
P#3. Power	[26, 36, 37]	49 LOC, 1 C, 5 M
P#4. Remainder	[26, 36, 37]	60 LOC, 1 C, 5 M
P#5. SyntheticProg1	[26]	65 LOC, 1 C, 5 M
P#6. SyntheticProg2	[26]	60 LOC, 1 C, 5 M
P#7. SyntheticProg3	[26]	62 LOC, 1 C, 5 M

procedure of the empirical study, the *Muclipse* tool was run on the program to be mutated to generate *FOMs*. *Muclipse* generates 1114 mutated versions of the programs to be mutated. Table 8 presents the number of first-order mutants for each subject program, and the frequency of each mutation operator.

The analysis modulefinds a list of *def-c-use* pairs and *def-p-use* pairs for each subject program. The analysis module finds 124 *def-c-use* pairs for all subject programs and 196 *def-p-use* pairs which are reduced to 98 *def-p-use* pairs. The analysis module finds 320 *du-pairs* which are reduced to 222 *du-pairs* for all subject programs. Table 9 presents the number of *du-pairs* for each subject program. The mutant generation module generates 122 mutated versions of the programs to be mutated. This means that the proposed technique reduced 89% of the number of first-order mutants generated by *Muclipse* and presents the efficiency of data flow in aiding the reduction of the number of mutants. Figure 3 shows the number of *FOMs* generated by *Muclipse* and the proposed technique for each subject program.

To answer the second research question *RQ2*, the four techniques were applied: the *Muclipse* tool, the *LastToFirst* Algorithm, the *DifferentOperators* Algorithm, and the proposed *DataFolwBasedSOM* algorithm to generate all possible second-order mutants. Table 10 presents the number of second-order mutants (*SOMs*) generated by each one of these algorithms. Applying the *Muclipse* tool twice gives 186802 *SOMs*, which represents the worst case. The proposed algorithm generated 222 *SOMs*, while the *LastToFirst* algorithm generated 559 *SOMs*, and the *DifferentOperators* algorithm generated 595 *SOMs* for all subject programs. To compare the last three algorithms in reducing the number of higher-order mutants regarding *FOMs*, the reduction value of *FOMs* generated by *Muclipse* ($RR1 = (FOMs - SOMs)/FOMs$) was computed. Table 10 and Figure 4 show the reduction ratio of the 1114 first-order mutants generated by the *Muclipse* tool for each subject program. Our proposed algorithm reduced 88.07% of the 1114 *FOMs*, while the *LastToFirst* algorithm reduced 49.82%, and the *DifferentOperators* algorithm reduced 46.59% of 1114 *FOMs* for all subject programs. The results show that our proposed al-

Table 8. Details of *FOMs* using Muclipse

#	Mutation Operators															Total
	AORB	AOIU	AODU	ROR	COD	SOR	LOI	ASRS	AORS	AOIS	AODS	COR	COI	LOR	LOD	
P#1	4	13	0	40	0	0	30	0	0	118	0	4	10	0	0	219
P#2	0	9	0	10	0	0	19	0	0	76	0	0	5	0	0	119
P#3	16	8	1	5	0	0	7	0	0	48	0	0	3	0	0	88
P#4	20	13	1	25	0	0	24	0	0	96	0	2	7	0	0	188
P#5	32	12	0	25	0	0	20	0	0	80	0	0	5	0	0	174
P#6	16	12	0	20	0	0	22	0	0	88	0	0	5	0	0	163
P#7	36	10	0	15	0	0	20	0	0	78	0	0	4	0	0	163
Total	124	77	2	140	0	0	142	0	0	584	0	6	39	0	0	1114

Table 9. The number of *du-pairs* for each subject program

# Subject program	dcu	dpu	Reduced dpu (Rdpu)	dcu+Rdpu	Mutation Points (mp)
P#1	16	72	36	52	22
P#2	10	20	10	20	14
P#3	14	10	5	19	13
P#4	24	30	15	39	21
P#5	20	20	10	30	18
P#6	22	28	14	36	19
P#7	18	16	8	26	15
Total	124	196	98	222	122

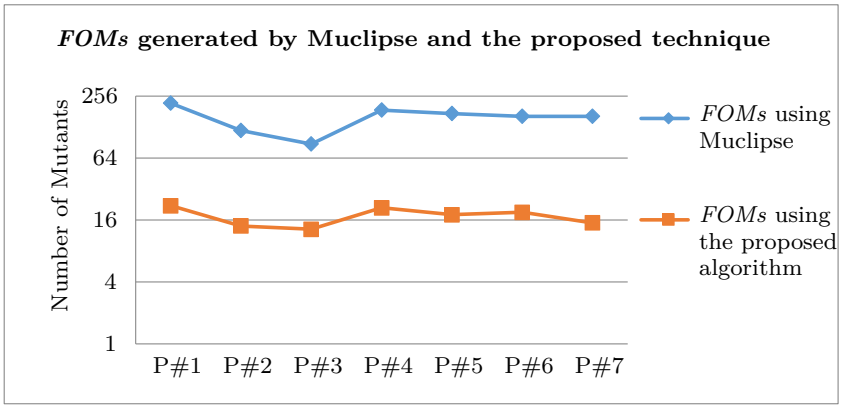


Figure 3. The number of *FOMs* generated by Muclipse and the proposed technique

gorithm outperforms the *LastToFirst* algorithm by 38.25% and the *DifferentOperators* algorithm by 41.48%.

To compare the last three algorithms in reducing all possible number of higher-order mutants, the Authors computed the reduction value of *SOMs* generated by Muclipse ($RR2 = (MSOMs - ASOMs)/MSOMs$) where *MSOMs* is the number of second-order mutants generated by *Muclipse* tool, and *ASOMs* is the number of second-order mutants generated by one of the other three algorithms. Table 12 and Figure 5 show the reduction ratio *RR2* of the 186802 second-order mutants generated by the *Muclipse* tool for all subject programs. Our proposed algorithm reduced 99.88% of all *SOMs* while the *LastToFirst* algorithm reduced 99.70% and the *DifferentOperators* algorithm reduced 99.68% of 186802 *SOMs* for all subject programs. The results show that our proposed algorithm outperforms the *LastToFirst* algorithm by 0.18%, and *DifferentOperators* algorithm by 0.20%. The re-

Table 10. The number of second-order mutants generated by the three algorithms

# Subject program	Muclipse	LastToFirst	DifferentOperators	The proposed algorithm
P#1	47557	110	118	52
P#2	13935	60	76	20
P#3	7634	44	48	19
P#4	35028	94	96	39
P#5	29995	87	87	30
P#6	26304	82	88	36
P#7	26349	82	82	26
Total	186802	559	595	222

Table 11. Reduction % of the number of first-order mutants generated by the three algorithms

# Subject program	LastToFirst	DifferentOperators	The proposed algorithm
P#1	49.77%	46.12%	76.26%
P#2	72.60%	65.30%	90.87%
P#3	79.91%	78.08%	91.32%
P#4	57.08%	56.16%	82.19%
P#5	60.27%	60.27%	86.30%
P#6	62.56%	59.82%	83.56%
P#7	62.56%	62.56%	88.13%
Total	49.82%	46.59%	80.07%

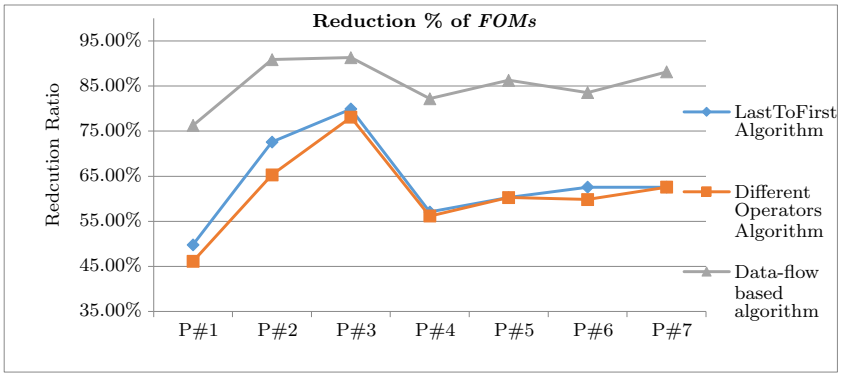


Figure 4. Reduction percentage of *FOMs* using the three algorithms

sults show the efficiency of data flow in aiding the reduction of the number of mutants, and the effectiveness of the proposed technique in finding a reduced set of higher-order mutants.

4.2. Empirical Study #2

4.2.1. Setup of Empirical Study #2

Subject Programs: The Authors selected four programs of the subject programs showed in Table 7 for conducting this empirical study to demonstrate that the proposed mutant generation technique does not lead to a substantial loss in the effectiveness of the mutation testing method. These programs are: *triangle*, *mid*, *power*, and *remainder*.

Procedure: the empirical study is conducted as follows.

1. The Authors randomly selected a set of the generated *FOMs* and *SOMs*. Table 13 shows the number and ratio of the selected *FOMs* and *HOMs*.
2. The Authors manually generated a set of test cases to kill all selected *FOMs*. Then, the

Table 12. Reduction % of the number of second-order mutants generated by the three algorithms

# Subject program	LastToFirst	DifferentOperators	The proposed algorithm
P#1	99.77%	99.75%	99.89%
P#2	99.57%	99.45%	99.86%
P#3	99.42%	99.37%	99.75%
P#4	99.73%	99.73%	99.89%
P#5	99.71%	99.71%	99.90%
P#6	99.69%	99.67%	99.86%
P#7	99.69%	99.69%	99.90%
Total	99.70%	99.68%	99.88%

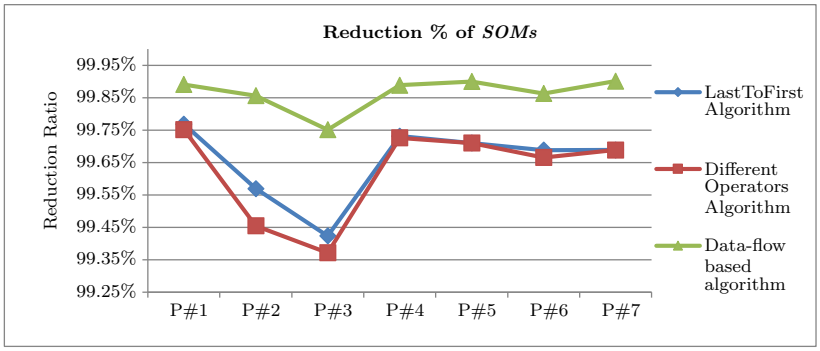


Figure 5. Reduction percentage of *SOMs* using the three algorithms

selected *SOMs* were executed using this set of test cases.

Table 13. The number and ratio of selected *FOMs* and *HOMs*

#Subject program	*FOMs*	*SOMs*
P#1. Triangle	28 (12.8%)	12 (23.1%)
P#2. Mid	36 (30.3%)	10 (50.0%)
P#3. Power	63 (71.6%)	17 (89.5%)
P#4. Remainder	32 (17.0%)	10 (25.6%)
Total(Mean)	159 (25.9%)	49 (37.7%)

4.3. Objectives of Study #2

The Authors applied the study procedure to illustrate that the proposed mutant generation technique does not lead to a substantial loss in the effectiveness of the mutation testing method.

4.3.1. Results and Discussion of Study #2

To illustrate that the proposed mutant generation technique does not lead to a substantial loss in the effectiveness of the mutation testing method, the Authors selected approximately 26% of *FOMs* and 38% of *SOMs* as shown in Table 13. There is a difference between the ratio of *FOMs* and *SOMs* because most of the selected *SOMs* contained one of the selected *FOMs*. The Authors manually generated a set of test cases to kill selected set of *FOMs*. Table 14 shows the number of required test cases to kill the selected set of *FOMs*.

All *FOMs* and *SOMs* were selected using the generated test cases Then, we classified the selected *FOMs* and *SOMs* into killed, not killed, and equivalent (manually investigated) mutants. Table 15 and Table 16 show the classification, number, and ratio of *FOMs* and *SOMs*, respectively.

The mutation score MS(P, T) was computed for each program using Eq. 1. Table 17 shows the mutation score for each program with respect to *FOMs* and *SOMs*. The mutation score shows that there is no significant loss in the efficiency of the generated mutants. The results of empirical study show that the proposed technique generated a smaller number of equivalent mutants.

Table 14. The number of test cases

Subject program	P#1. Triangle	P#2. Mid	P#3. Power	P#4. Remainder	Total
No. of test cases	3	3	2	2	10

Table 15. Classification of the selected *FOMs*

Subject program	#Killed mutants(%)	#Not killed (%)	#Equivalent (%)	Total
P#1. Triangle	26 (93%)	2 (7%)	0 (0%)	28 (100%)
P#2. Mid	32 (89%)	0 (0%)	4 (11%)	36 (100%)
P#3. Power	52 (83%)	11 (17%)	0 (0%)	63 (100%)
P#4. Remainder	29 (91%)	0 (0%)	3 (9%)	32 (100%)
Total(Mean)	139 (89%)	13 (6%)	7 (5%)	159 (100%)

Table 16. Classification of the selected *SOMs*

Subject program	#Killed mutants(%)	#Not killed(%)	#Equivalent(%)	Total
P#1. Triangle	11 (92%)	1 (8%)	0 (0%)	12 (100%)
P#2. Mid	9 (90%)	0 (0%)	1 (10%)	10 (100%)
P#3. Power	14 (82%)	2 (12%)	1 6%)	17 (100%)
P#4. Remainder	10 (100%)	0(0%)	0(0%)	10 (100%)
Total(Mean)	44 (91%)	3 (5%)	2 (4%)	49 (100%)

Table 17. Mutation score of *FOMS* and *SOMs*

Subject Program	*FOM*	*SOM*
P#1. Triangle	92.9%	91.7%
P#2. Mid	100.0%	100.0%
P#3. Power	82.5%	87.5%
P#4. Remainder	100.0%	100.0%
Mean	93.8%	94.8%

4.4. Threats to Validity

– Construct Validity
There are three important questions about the goal of the experiments. First, are the Authors measuring the construct they intended to measure? Although, the Authors intended to find a reduced set of higher-order mutants, some useless mutants (e.g., equivalent mutants and redundancies) can be generated and included in this set. Second, did the Authors translate these constructs correctly into observable measures? Although, the considered the def locations and use locations of the same variable, the mutations for other variables at these use locations are not considered. Third, did the used metrics have suitable discriminatory power? Although, the metric of the reduction is the ratio between the number of generated higher-order mutants to the number of all first-order mutants, it does not consider the subtlety of the generated higher-order mutants.

– External Validity
The main external threat to validity; conditions that limit the ability to generalize the results of our empirical studies to a larger population of subjects programs, is the set of subject programs. Although the set of the subject programs contains some programs which have been used in many previous studies, the Authors cannot claim that these subjects represent a random selection over the population of programs as a whole. Although the set of the subject programs have been used in many previous studies, a single researcher selected these programs which may influence results. Although, the Authors selected the subject programs in a neutral attitude, there is no guarantee that selection process was performed in unbiased way.

– Internal Validity
There are some main internal threats to validity, which are the influences that can af-

fect the dependent variables. First, although the mutation operators were selected in a significant way to prevent the generation of equivalent mutants, the equivalent mutants were not considered through the reduction ratio. Therefore, other empirical studies are required to overcome this problem. Second, although a common algorithm proposed by F. E. Allen and J. Cocke [31] was implemented to find the set of definition-uses, the accuracy of the implementation can influence the number of definition-uses pairs which have a strong effect on the number of generated mutants.

5. Related Work

Mutation testing has been developed by DeMillo et al. [3] and Hamlet [4] to create test data for killing the seeded mutations in the tested program [5]. The researchers classified mutants into two categories: 1) First-order mutants which are created by the injection of a unique fault in the tested program [5]; 2) Higher-order mutants which are produced by inserting two or more faults in the tested program [1]. Jia and Harman [38] provided a comprehensive analysis of trends and results of mutation testing techniques. This section reviews mutation operators design, generation of mutants, reduction of the number of mutants, and data flow analysis in mutation testing.

5.1. Mutation Operators and Mutant Generation

At the beginning of mutation testing, most mutation testing techniques targeted FORTRAN programs. Many mutation operators are presented for most of programming languages such as FORTRAN IV [39, 40], FORTRAN 77 [15, 41], Ada [42, 43], ANSI C [13], and the Java programming language [44, 45]. Alexander et al. [46] presented a set of mutation operators to insert into Java utility libraries. Bradbury et al. [47] presented a set of mutation operators to the concurrent Java programs. Derezińska proposed a set of C# mutation operators [48, 49]. Ferrari et al. [50] suggested a set of mutation operators for Aspect-Oriented programs. Anbalagan and Xie [51, 52] presented a technique for creating mutants for pointcuts and detecting equivalent mutants.

5.2. Mutant Reduction

Considering all mutants makes mutation testing a computationally expensive technique. Therefore, reducing the number of the considered mutants without a significant loss of test effectiveness has become a key research problem. Suppose M is a set of mutants and T is a set of test data. The mutation score of the test set T applied to mutants M is $MS(M, T)$. Therefore, the mutant reduction problem is known as finding a subset of mutants m from M, where $MS(m, T) = MS(M, T)$. Offutt and Untch [53] classified mutant reduction techniques to three techniques. These techniques concentrate only on the *fewer*, the *faster*, or the *smarter* mutants. Jia and Harman [38] divided these techniques into two techniques. One technique reduces the created mutants and the other technique reduces the execution cost. There are four popular techniques to reduce the number of considered mutants: *mutant sampling*, *mutant clustering*, *selective mutation*, and *higher-order mutation*. In mutant sampling a percentage of mutants is randomly selected from the set of all mutants [10–12] and the remaining mutants are discarded [15, 54]. In mutant clustering [17] a subset of mutants is selected using clustering algorithms and the remaining mutants are discarded [55]. The Selective Mutation [56,57] can be achieved by reducing the number of applied mutation operators without a significant loss of test effectiveness [14]. Selective Mutation can be done by omitting two mutation operators [14], four mutation operators [53], or six mutation operators. Wong and Mathur selected mutation operators based on test effectiveness [58, 59]. Offutt et al. [60] classified Mothra mutation operators to three groups: statements, operands, and expressions and omitted operators from each class in turn. Mresa and Bottaci [61] considered mutants which have the

ability to detect equivalent mutants. Jia and Harman [20, 22] suggested reducing the number of first-order mutants by replacing them with a single *HOM*. Langdon et al. have used genetic programming to generate higher-order mutants [62].

5.3. Data Flow Analysis and Mutation Testing

A number of work explored the role of data flow analysis in mutation testing. Girgis and Woodward [63] and Marshall et al. [64] studied applying data flow analysis in weak mutation testing. Offutt and Tewary [65] and Mathur and Wong [54] studied the coverage of mutation based and data flow criteria by each other. Wong and Mathur [66] compared the effectiveness of mutation and data flow testing in fault detection. A comprehensive comparison between mutation and data flow testing techniques based on findings reported in research articles can be found in [67]. This field is in need for a lot of work to study the role of data flow concepts in higher-order mutation testing. From the above discussion, it is clear that our work belongs to the mutant reduction category. In addition, it differs from all pervious mutant reduction work. It is the first work treating mutant reduction by reducing the locations of seeding mutation.

6. Conclusion and Future work

In this paper a new technique for generating a reduced set of higher-order mutants was introduced. The proposed technique uses data-flow concepts for the identification of the higher-order mutants. The generated set of higher-order mutants consists of a reduced number of mutants, which reduces the cost of higher-order mutation testing. In addition, the proposed technique can generate the higher-order mutants directly without generating the first-order mutants or by combining two or more first-order mutants. The results of the conducted experiments showed that the proposed technique outperforms the *LastToFirst* algorithm by 38.25%, and the *DifferentOperators* algorithm by 41.48% reducing the total possible number of higher-order mutants regarding *FOMs*. In addition, the proposed algorithm outperforms the *LastToFirst* algorithm by 0.18%, and the *DifferentOperators* algorithm by 0.20% in reducing all possible number of higher-order mutants. The obtained results showed the efficiency of data flow in aiding the reduction of the number of mutants and the effectiveness of the proposed technique in finding a reduced set of higher-order mutants. In future work, The Authors are planning to perform these studies with real and large subject programs. In addition, the future work will try to answer the questions: "Do data-flow based higher-order mutants create subtle faults?" and "What are the effects of the proposed approach in terms of overcoming realism and equivalent mutant problems of mutation testing?". Besides, the Authors will study the subsuming property of the generated mutants in the future work.

References

[1] Y. Jia and M. Harman, "Higher order mutation testing," *Inf. Softw. Technol.*, Vol. 51, No. 10, Oct. 2009, pp. 1379–1393. [Online]. http://dx.doi.org/10.1016/j.infsof.2009.04.016

[2] W.B. Langdon, M. Harman, and Y. Jia, "Efficient multi-objective higher order mutation testing with genetic programming," *J. Syst. Softw.*, Vol. 83, No. 12, Dec. 2010, pp. 2416–2430. [Online]. http://dx.doi.org/10.1016/j.jss.2010.07.027

[3] R.A. DeMillo, R.J. Lipton, and F.G. Sayward, "Hints on test data selection: Help for the practicing programmer," *Computer*, Vol. 11, No. 4, Apr. 1978, pp. 34–41. [Online]. http://dx.doi.org/10.1109/C-M.1978.218136

[4] R.G. Hamlet, "Testing programs with the aid of a compiler," *IEEE Transactions on Software Engineering*, Vol. SE-3, No. 4, July 1977, pp. 279–290.

[5] K. Ayari, S. Bouktif, and G. Antoniol, "Automatic mutation test input data generation via ant colony," in *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '07. New York, NY, USA: ACM, 2007, pp. 1074–1081. [Online]. http://doi.acm.org/10.1145/1276958.1277172

[6] W.E. Howden, "Weak mutation testing and completeness of test sets," *IEEE Transactions on Software Engineering*, Vol. SE-8, No. 4, July 1982, pp. 371–379.

[7] Q.V. Nguyen and L. Madeyski, *Advanced Computational Methods for Knowledge Engineering: Proceedings of the 2nd International Conference on Computer Science, Applied Mathematics and Applications (ICCSAMA 2014)*. Cham: Springer International Publishing, 2014, ch. Problems of Mutation Testing and Higher Order Mutation Testing, pp. 157–172. [Online]. http://dx.doi.org/10.1007/978-3-319-06569-4_12

[8] M. Kintis, M. Papadakis, and N. Malevris, "Isolating first order equivalent mutants via second order mutation," in *IEEE Fifth International Conference on Software Testing, Verification and Validation (ICST)*, April 2012, pp. 701–710.

[9] L. Madeyski, W. Orzeszyna, R. Torkar, and M. Józala, "Overcoming the equivalent mutant problem: A systematic literature review and a comparative experiment of second order mutation," *IEEE Transactions on Software Engineering*, Vol. 40, No. 1, Jan 2014, pp. 23–42.

[10] A.T. Acree, "On mutation," Ph.D. dissertation, Georgia Inst of Tech Atlanta School of Information and Computer Science, Aug 1980.

[11] T.A. Budd, "Mutation analysis of program test data," Ph.D. dissertation, Yale Univ, 1980.

[12] I. Bluemke and K. Kulesza, *New Results in Dependability and Computer Systems: Proceedings of the 8th International Conference on Dependability and Complex Systems DepCoS-RELCOMEX, September 9-13, 2013, Brunów, Poland*. Heidelberg: Springer International Publishing, 2013, ch. Reduction of Computational Cost in Mutation Testing by Sampling Mutants, pp. 41–51. [Online]. http://dx.doi.org/10.1007/978-3-319-00945-2_4

[13] H. Agrawal, R.A. DeMillo, R. Hathaway, W.M. Hsu, W. Hsu, E. Krauser, R.J. Martin, A.P. Mathur, and E. Spafford, "Design of mutant operators for the C programming language," Software Eng. Research Center, Computer Science Dept., Purdue Univ., Technical Report SERC-TR-41-P, 1989.

[14] A.P. Mathur, "Performance, effectiveness, and reliability issues in software testing," in *Computer Software and Applications Conference, 1991. COMPSAC '91., Proceedings of the Fifteenth Annual International*, Sep 1991, pp. 604 – 605.

[15] K.N. King and A.J. Offutt, "A fortran language system for mutation-based software testing," *Softw. Pract. Exper.*, Vol. 21, No. 7, Jun. 1991, pp. 685–718. [Online]. http://dx.doi.org/10.1002/spe.4380210704

[16] A.J. Offutt, G. Rothermel, and C. Zapf, "An experimental evaluation of selective mutation," in *Software Engineering, 1993. Proceedings., 15th International Conference on*, May 1993, pp. 100–107.

[17] S. Hussain, "Mutation clustering," Master's thesis, King's College London, 2008.

[18] A.J. Offutt, "Investigations of the software testing coupling effect," *ACM Trans. Softw. Eng. Methodol.*, Vol. 1, No. 1, Jan. 1992, pp. 5–20. [Online]. http://doi.acm.org/10.1145/125489.125473

[19] M. Kintis, M. Papadakis, and N. Malevris, "Evaluating mutation testing alternatives: A collateral experiment," in *Software Engineering Conference (APSEC), 2010 17th Asia Pacific*, Nov 2010, pp. 300–309.

[20] M. Polo, M. Piattini, and I. García-Rodríguez, "Decreasing the cost of mutation testing with second-order mutants," *Softw. Test. Verif. Reliab.*, Vol. 19, 2009, pp. 111–131.

[21] M. Harman, Y. Jia, and W.B. Langdon, "A manifesto for higher order mutation testing," in *Software Testing, Verification, and Validation Workshops (ICSTW), 2010 Third International Conference on*, April 2010, pp. 80–89.

[22] Y. Jia and M. Harman, "Constructing subtle faults using higher order mutation testing," in *Eighth IEEE International Working Conference onSource Code Analysis and Manipulation*, Sept 2008, pp. 249–258.

[23] P.M. Herman, "A data flow analysis approach to program testing," *Australian Computer Journal*, Vol. 8, No. 3, 1976, pp. 92–96.

[24] S. Rapps and E.J. Weyuker, "Selecting software test data using data flow information," *IEEE Transactions on Software Engineering*, Vol. SE-11, No. 4, April 1985, pp. 367–375.

[25] P.G. Frankl and E.J. Weyuker, "An applicable family of data flow testing criteria," *IEEE Transactions on Software Engineering*, Vol. 14, No. 10, Oct 1988, pp. 1483–1498.

[26] A.S. Ghiduk, M.J. Harrold, and M.R. Girgis, "Using genetic algorithms to aid test-data generation for data-flow coverage," in *Software Engineering Conference, 2007. APSEC 2007. 14th Asia-Pacific*, Dec 2007, pp. 41–48.

[27] I. Burnstein, *Practical Software Testing*, 1st ed. Springer Science+Business Media New York: Springer Professional Computing, 2003, ch. A Process-Oriented Approach.

[28] M. Papadakis and N. Malevris, "Automatically performing weak mutation with the aid of symbolic execution, concolic testing and search-based testing," *Software Quality Journal*, Vol. 19, No. 4, Dec. 2011, pp. 691–723. [Online]. http://dx.doi.org/10.1007/s11219-011-9142-y

[29] M.S. Hecht, *Flow Analysis of Computer Programs*. New York, NY, USA: Elsevier Science Inc., 1977.

[30] S. Rapps and E.J. Weyuker, "Data flow analysis techniques for test data selection," in *Proceedings of the 6th International Conference on Software Engineering*, ser. ICSE '82. Los Alamitos, CA, USA: IEEE Computer Society Press, 1982, pp. 272–278. [Online]. http://dl.acm.org/citation.cfm?id=800254.807769

[31] F.E. Allen and J. Cocke, "A program data flow analysis procedure," *Commun. ACM*, Vol. 19, No. 3, Mar. 1976, p. 137. [Online]. http://doi.acm.org/10.1145/360018.360025

[32] B.H. Smith and L. Williams, "On guiding the augmentation of an automated test suite via mutation analysis," *Empirical Softw. Engg.*, Vol. 14, No. 3, Jun. 2009, pp. 341–369. [Online]. http://dx.doi.org/10.1007/s10664-008-9083-7

[33] B.H. Smith and L. Williams, "An empirical evaluation of the muJava mutation operators," in *Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION, 2007. TAICPART-MUTATION 2007*, Sept 2007, pp. 193–202.

[34] Y.S. Ma and J. Offutt, "Description of method-level mutation operators for Java," 2005. [Online]. https://cs.gmu.edu/~offutt/mujava/mutopsMethod.pdf

[35] P. May, J. Timmis, and K. Mander, *Artificial Immune Systems: 6th International Conference, ICARIS 2007, Santos, Brazil, August 26-29, 2007. Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, ch. Immune and Evolutionary Approaches to Software Mutation Testing, pp. 336–347. [Online]. http://dx.doi.org/10.1007/978-3-540-73922-7_29

[36] C.C. Michael, G. McGraw, and M.A. Schatz, "Generating software test data by evolution," *IEEE Transactions on Software Engineering*, Vol. 27, No. 12, Dec. 2001, pp. 1085–1110. [Online]. http://dx.doi.org/10.1109/32.988709

[37] R.P. Pargas, M.J. Harrold, and R. Peck, "Test-data generation using genetic algorithms," *Softw. Test., Verif. Reliab.*, Vol. 9, No. 4, 1999, pp. 263–282. [Online]. http://dx.doi.org/10.1002/(SICI)1099-1689(199912)9:4<263::AID-STVR190>3.0.CO;2-Y

[38] Y. Jia and M. Harman, "An analysis and survey of the development of mutation testing," *IEEE Transactions on Software Engineering*, Vol. 37, No. 5, Sept 2011, pp. 649–678.

[39] T.A. Budd, R.J. Lipton, R. DeMillo, and F. Sayward, "The design of a prototype mutation system for program testing," in *Proceedings NCC, AFIPS Conference Records*, 1978, pp. 623–627.

[40] T.A. Budd, R.A. DeMillo, R.J. Lipton, and F.G. Sayward, "Theoretical and empirical studies on using program mutation to test the functional correctness of programs," in *Proceedings of the 7th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, ser. POPL '80. New York, NY, USA: ACM, 1980, pp. 220–233. [Online]. http://doi.acm.org/10.1145/567446.567468

[41] A. J.Offutt and K.N. King, "A FORTRAN 77 interpreter for mutation analysis," *SIGPLAN Not.*, Vol. 22, No. 7, Jul. 1987, pp. 177–188. [Online]. http://doi.acm.org/10.1145/960114.29669

[42] J. Bowser, "Reference manual for Ada mutant operators," Georgia Inst. of Technology, Technical Report GIT-SERC-88/02, 1988.

[43] A.J. Offutta, J. Voas, and J. Payn, "Mutation operators for Ada," George Mason Univ., Technical Report ISSE-TR-96-09, 1996.

[44] S. Kim, J.A. Clark, and J.A. McDermid, "The rigorous generation of Java mutation operators using HAZOP," in *12th International Conf. Software and Systems Eng. and Their Applications*, 1999.

[45] Y.S. Ma, Y.R. Kwon, and J. Offutt, "Inter-class mutation operators for Java," in *Software Reliability Engineering, 2002. ISSRE 2003. Proceedings. 13th International Symposium on*, 2002, pp. 352–363.

[46] R.T. Alexander, J.M. Bieman, S. Ghosh, and B. Ji, "Mutation of Java objects," in *Proceedings of the 13th International Symposium on Software Reliability Engineering*, 2002, pp. 341–351.

[47] J.S. Bradbury, J.R. Cordy, and J. Dingel, "Mutation operators for concurrent Java (J2SE 5.0)," in *Mutation Analysis, 2006. Second Workshop on*, Nov 2006, p. 11.

[48] A. Derezińska, *Software Engineering Techniques: Design for Quality*. Boston, MA: Springer US, 2007, ch. Advanced mutation operators applicable in C# programs, pp. 283–288. [Online]. http://dx.doi.org/10.1007/978-0-387-39388-9_27

[49] A. Derezinska, "Quality assessment of mutation operators dedicated for C# programs," in *Quality Software, 2006. QSIC 2006. Sixth International Conference on*, Oct 2006, pp. 227–234.

[50] F.C. Ferrari, J.C. Maldonado, and A. Rashid, "Mutation testing for aspect-oriented programs," in *Software Testing, Verification, and Validation, 2008 1st International Conference on*, April 2008, pp. 52–61.

[51] P. Anbalagan and T. Xie, "Efficient mutant generation for mutation testing of pointcuts in aspect-oriented programs," in *Proceedings of the Second Workshop on Mutation Analysis*, ser. MUTATION '06. Washington, DC, USA: IEEE Computer Society, 2006, p. 3. [Online]. http://dx.doi.org/10.1109/MUTATION.2006.3

[52] P. Anbalagan and T. Xie, "Automated generation of pointcut mutants for testing pointcuts in AspectJ programs," in *19th International Symposium on Software Reliability Engineering, 2008. ISSRE 2008.*, Nov 2008, pp. 239–248.

[53] A.J. Offutt and R.H. Untch, "Mutation testing for the new century," W.E. Wong, Ed. Norwell, MA, USA: Kluwer Academic Publishers, 2001, ch. Mutation 2000: Uniting the Orthogonal, pp. 34–44. [Online]. http://dl.acm.org/citation.cfm?id=571305.571314

[54] A.P. Mathur and W.E. Wong, "An empirical comparison of data flow and mutation-based test adequacy criteria," *Software Testing, Verification and Reliability*, Vol. 4, No. 1, 1994, pp. 9–31. [Online]. http://dx.doi.org/10.1002/stvr.4370040104

[55] C. Ji, Z. Chen, B. Xu, and Z. Zhao, "A novel method of mutation clustering based on domain analysis," in *Proceedings of the 21st International Conference on Software Engineering & Knowledge Engineering (SEKE'2009), Boston, Massachusetts, USA, July 1-3, 2009*, 2009, pp. 422–425.

[56] A.S. Namin and J.H. Andrews, "Finding sufficient mutation operators via variable reduction," in *Proceedings of the Second Workshop on Mutation Analysis*, ser. MUTATION '06. Washington, DC, USA: IEEE Computer Society, 2006, p. 5. [Online]. http://dx.doi.org/10.1109/MUTATION.2006.7

[57] A.S. Namin and J.H. Andrews, "On sufficiency of mutants," in *Companion to the Proceedings of the 29th International Conference on Software Engineering*, ser. ICSE COMPANION '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 73–74. [Online]. http://dx.doi.org/10.1109/ICSECOMPANION.2007.56

[58] W.E. Wong, "On mutation and data flow," Ph.D. dissertation, West Lafayette, IN, USA, 1993, uMI Order No. GAX94-20921.

[59] W.E. Wong and A.P. Mathur, "Reducing the cost of mutation testing: An empirical study," *J. Syst. Softw.*, Vol. 31, No. 3, Dec. 1995, pp. 185–196. [Online]. http://dx.doi.org/10.1016/0164-1212(94)00098-0

[60] A.J. Offutt, A. Lee, G. Rothermel, R.H. Untch, and C. Zapf, "An experimental determination of sufficient mutant operators," *ACM Trans. Softw. Eng. Methodol.*, Vol. 5, No. 2, Apr. 1996, pp. 99–118. [Online]. http://doi.acm.org/10.1145/227607.227610

[61] E.S. Mresa and L. Bottaci, "Efficiency of mutation operators and selective mutation strategies: an empirical study," *Software Testing, Verification and Reliability*, Vol. 9, No. 4, 1999, pp. 205–232. [Online]. http://dx.doi.org/10.1002/(SICI)1099-1689(199912)9:4<205::AID-STVR186>3.0.CO;2-X

[62] W.B. Langdon, M. Harman, and Y. Jia, "Multi objective higher order mutation testing with genetic programming," in *Testing: Academic and Industrial Conference - Practice and Research Techniques, 2009. TAIC PART '09*, Sept 2009, pp. 21–29.

[63] M.R. Girgis and M.R. Woodward, "An integrated system for program testing using weak mutation and data flow analysis," in *Proceedings of the 8th International Conference on Software Engineering*, ser. ICSE '85. Los Alamitos, CA, USA: IEEE Computer Society Press, 1985, pp. 313–319. [Online]. http://dl.acm.org/citation.cfm?id=319568.319662

[64] A.C. Marshall, D. Hedley, I.J. Riddell, and M.A. Hennell, "Static dataflow-aided weak mutation analysis (sdawm)," *Inf. Softw. Technol.*, Vol. 32, No. 1, Jan. 1990, pp. 99–104. [Online]. http://dx.doi.org/10.1016/0950-5849(90)90053-T

[65] A.J. Offutt and K. Tewary, "Empirical comparisons of data flow and mutation testing," 1992.

[66] W.E. Wong and A.P. Mathur, "Fault detection effectiveness of mutation and data flow testing," *Software Quality Journal*, Vol. 4, No. 1, pp. 69–83. [Online]. http://dx.doi.org/10.1007/BF00404650

[67] S. Kakarla, S. Momotaz, and A.S. Namin, "An evaluation of mutation and data-flow testing: A meta-analysis," in *IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, March 2011, pp. 366–375.

e-Informatica Software Engineering Journal, Volume 10, Issue 1, 2016, pages: 51–67, DOI 10.5277/e-Inf160103

Automatic SUMO to UML translation

Bogumiła Hnatkowska[a]
[a] *Faculty of Computer Science and Management, Wrocław University of Science and Technology*
bogumila.hnatkowska@pwr.edu.pl

Abstract

Existing ontologies are a valuable source of domain knowledge. This knowledge could be extracted and reused to create domain models. The extraction process can be aided by tools that enable browsing ontology, marking interesting notions and automatic conversion of selected elements to other notations. The paper presents a tool that can be used for SUMO to UML translation. Such a transformation is feasible and results in a high-quality domain model which is consistent, correct, and complete, providing that input ontology has the same features.

Keywords: SUMO ontology, information retrieving, domain model, UML, class diagram

1. Introduction

A domain model is a key development artifact. It captures the most important types of objects in the context of the domain, i.e. entities that exist or events that transpire in the environment in which the system works [1,2]. The domain model, besides business object models and glossary [1], is used to document the domain to which the system relates. The domain model could be represented with the use of different notations, among which the most popular are Entity Relationships Diagrams and UML class diagrams.

Domain models should be of high quality to reduce the number of changes when the development proceeds. Among quality factors the most important are [3]: consistency, completeness, and correctness (3C).

Consistency and completeness could be perceived from 2 perspectives: external and internal, from which the external one is more difficult to achieve. External completeness means that we have identified all important entities and relationships in the domain, while external consistency means that we have documented the identified elements in a way that preserves their semantics [4]. On the other hand, the domain model is internally consistent when it contains no contradictions and it is internally complete when it does not include any undefined object, no information is left unstated or is to be determined [2].

The definition of model correctness is much vaguer. Some authors define it as a mixture of consistency and completeness [3], others [4] refer it to syntactic correctness (this meaning of correctness is used further in the paper).

A business analyst typically elaborates a domain model during a business modelling or requirement specification phase [2].

Different elicitation techniques serve to discover entities in the domain, e.g. interviews. However, the obtained results strongly depend on the complexity of the domain, business analyst experience and the quality of information sources. The more difficult domain, the less experienced an analyst or poor quality sources, the more likely worse quality of the resultant domain model.

On the other hand, domain knowledge is often included in existing ontologies and could be extracted from them. The extraction process could be (partially) automated, which would result in a high- quality domain model. Consistency and correctness of such a model could be guaranteed by construction, assuming that the source (ontology) itself is correct and consistent with the

domain. The model completeness, at least the internal one, could also be checked.

Many papers prove that the domain knowledge represented by ontology can be widely used in the design process of information systems. For example, in [5] author analyses the role of ontologies in software engineering process. The author claims that ontology is a significant source of knowledge in the conceptualization phase and proposes the ontology life cycle as the background for a software development. A similar view is presented in [6] where the authors state that the integration with ontology can improve software modelling. An application of domain ontologies to conceptual model development is also in presented in [7].

There are many high-level ontologies currently developed, e.g. BFO, Cyc, GFO, SUMO. The last one, SUMO, seems to be very promising because it became the basis for the development of many specific domain ontologies. A particularly useful feature is that the notions of SUMO have formal definitions (expressed in SUO-KIF language) and at the same time they are mapped to the WordNet lexicon [8]. SUO-KIF is a variant of the KIF (Knowledge Interchange Format) language [9]. Knowledge is described declaratively as objects, functions, relations, and rules. SUMO and related ontologies form the largest formal public ontology in existence today [8, 9]. What is more, the ontologies that extend SUMO, are available under GNU General Public License.

The paper presents a tool for automatic SUMO to UML translation. It is thought as a support for a business analyst collaborating with business experts. The main functionalities include: browsing ontology content, selection of interesting elements, and translation of selected elements to a UML class diagram. The solution presentation covers the meta-model of SUMO notions (the main input to the transformation process), tool architecture and an example of the domain model which results from tool application. The genesis of the tool (related works) is also shortly described as well as the problems met during implementation, and the elements that will be included in the next release.

The only tool available on the Internet that supports SUMO is SUMO browser, called Sigma [10]. Tools that allow to create a UML class diagram from the existing ontology exist for other formalisms, e.g. OWL [11], but not for SUO-KIF. However, SUO-KIF could be translated to other formalisms, e.g. DLP [12].

SUMO was selected from existing ontologies for the following reasons:

- It constitutes the biggest set of ontologies which is freely available; SUMO contains definitions of more than 21 thousands of terms, and more than 70 thousands of axioms; moreover, the mapping of SUMO notion to WordNet is also available [9];
- SUO-KIF language is very flexible; it allows to handle relations among three or more things directly (e.g. OWL does not); it supports statements and rules written not only in First-Order Logic, but also (at least partially) in the Higher-Order Logic (e.g. "(believes John (likes Bob Sue))", when the second argument of "believes" is a proposition) [9];
- Existing translation of SUMO to OWL is a provisional and necessarily lossy [9], which calls into question its usefulness; on the other hand, it is possible to perform the reverse translation from OWL to SUMO, which seems more promising, because the result could be extended with the usage of SUO-KIF features;
- The flexibility of SUO-KIF is very similar to the SBVR standard [13], promoted by OMG, defining the meta-model for representation of business vocabulary, and business rules; SBVR statements could be directly translated either to SUO-KIF or UML.

UML was selected as the target language for translation because it is a general purpose modeling and specification language commonly used not only by programmers but also by business analysts. Besides, the Entity Relationship Diagram is a frequently selected notation to describe domain models. Together with OCL it forms a very useful tandem to define constraints on the domain behavior in a formal way. The UML class diagram could be easily translated to other representations, either more business oriented like SBVR (e.g. [13]) or

more program oriented like Java, C++, SQL (e.g. [14]).

A tool for automatic SUMO to UML translation can be useful for anyone (especially a business analyst) who would like to familiarize with some specific domain. Theoretically, he or she can read the ontology definition for that purpose. Unfortunately, even if the SUMO browser is in use, knowledge extraction from SUMO is a challenge. SUMO is expressed in the textual SUO-KIF language which is not not commonly known. After a while, a reader is overloaded with textual definitions. The aim of the paper is to propose a solution to this problem. The solution is based on the observations that: (1) UML is a universal specification and modeling language to present data models, software architecture or business models; moreover it is supported by many tools (CASE, IDE), (2) graphical notations are easier to understand especially if the model is complex, with many relationships among model elements.

The rest of the paper is structured as follows. Section 2 presents related works and clearly states the paper's contribution. The proposed SUMO meta-model which supports the transformation process is described in Section 3. The tool and its main functional components are presented in Section 4. Newly introduced transformation rules for SUMO attributes and their relations are the subject of Section 5. Section 6 shortly defines existing transformation rules. An example of a transformation with a short discussion of its shortcomings is given in Section 7. Section 8 presents the problems to be addressed in the future. Section 9, the last one, concludes the paper.

2. Related Works

The paper [15] is the first in a series considering the SUMO ontology as a source for domain modelling. It presents an initial set of mapping rules between SUMO notions and UML notions, and identifies the elements difficult to extract, e.g. attributes.

The paper [16] presents an outline of a systematic approach to the development of a domain model on the basis of selected SUMO ontologies. It involves only a few steps. It starts with needs description, next it goes through the identification of business processes in the area of interests which help to decide if a notion of an ontology is in the area of interests (and should be translated to UML) or not. After the analysis of the selected elements, they are translated (manually) to a UML class diagram. The approach was tested on a few examples. Some SUMO-UML mappings were also refined. The biggest problems the authors found are:

- ontology size – it contains many irrelevant (out of scope) elements,
- domain knowledge is spread over many ontologies (files),
- some facts are defined at a very general level (predicates between *Object*, *Physical*) which makes the interpretation more difficult.

In the paper [17] the refined version of the approach from [16] is presented. It also consists of only a few steps, but their definition is much more formal and close to implementation needs. The main idea of the approach is a guided selection of the SUMO extract, which will be further translated to UML. The paper also proposes some new transformation rules, e.g. transformation of unary functions. The general finding of that work is that the process of knowledge extraction must be supported by a tool. Otherwise, the process, even if the results are promising, is very time-consuming, and error prone.

The contributions of this paper are as follows:

- The meta-model of SUMO notions used within a transformation process (see Section 3).
- Definition of a tool architecture (see Section 4).
- New transformation rules for SUMO attributes and their relations (see Section 5).
- Verification and correction of transformation rules defined in [15–17]; the subset of implemented rules (including the changed ones) is presented in Section 6.

The transformation process between two models can be specified and performed in many ways. If the source and targeted models are expressed in the XML language, the transformation process can be defined as the Extensible Stylesheet

Language Transformation (XSLT) and executed by a dedicated engine (see [18] for an example). This approach suffers from low readability and maintainability, this is why the transformation between meta-models is considered more often (e.g. [19]). In this approach at first the meta-models of the source and target models are prepared or adopted, and next the transformation rules between meta-classes are defined. Transformation rules can be expressed either in operative languages, like the Atlas Transformation Language (ATL), java or declarative ones like QVT-Relations. In the paper, the approach based on meta-models is in use. The SUMO meta-model is defined by the author of that paper. The UML meta-model is freely available (eclipse.uml2 framework).

The SUMO to UML transformation rules defined in [15–17] answer the question how to map elements such as classes and their hierarchies or relations and their hierarchies but they do not address SUMO attributes and their relationships. The problem with the SUMO attributes is that they are represented differently than attributes in the UML language. In SUMO the attribute is defined as "a quality which we cannot choose not to reify into subclasses of Objects" [8]. Because of that, attributes are assigned not to classes as in UML but to class instances. This paper fills this gap. The thorough analysis of SUMO relations between attributes is conducted here. On that basis the mapping of SUMO attributes to UML language is proposed. The mapping involves the definition of a UML profile, presented in Section 5.

The set of transformation rules defined in [15–17] was verified and extended in the meantime. The newly introduced transformation rules (including those defined for attributes), and the changed transformation rules with their justification are presented in Section 6.

3. Meta-model of SUMO Notions

To support the SUMO to UML transformation process the content of SUO-KIF files has to be represented at the higher abstraction level, which enables both: checking static consistency rules and performing the transformation process itself. It is achieved with so called meta-model of SUMO notions – see Fig. 1. The initial version of the meta-model was presented in [20]. Here the diagram is extended by new meta-classes.

The diagram reflects the logical structure of the SUO-KIF file which can be perceived as a set of sentences. A SUMO sentence is represented by the *Sentence* abstract class – the parent of all possible kinds of statements in SUMO. Each sentence belongs to exactly one *OntologySegment* (SUO-KIF file). Below there is a short description of concrete sentence classes:

1. *LogicalSentence* – a sentence starting with a logical operator, e.g. "(=> ...), (<=> ...)"; a tautology built with an implication and/or an equivalence operator;
2. *QuantifiedSentence* – a sentence starting either with a universal or existential quantifier: "(forall ...) or (exists ...)";
3. *RelationalSentence* – a sentence starting with a name of function or relation: "(name ...)"; a fact in the considered domain stating, for example, that John likes Karin.

It is assumed that only sentences written at the first level are instantiated by the SUMO to UML translator, e.g. the text: "(=> (instance ?REL BinaryPredicate) (valence ?REL 2))" will be instantiated as one sentence even if it contains two internal sub-sentences. The parser omits SUMO comments.

The right side of the class diagram shows the structure of SUMO notions. The *Entity* is "the root node of the ontology" [8]. It is associated with all sentences it belongs to (as a part).

Entity is the parent for two UML classes interesting in the context of the considered transformation:

- *Relation* – definition of a SUMO relation or function, together with its domains and/or range (see Fig. 2);
- *Type* – represents a SUMO notion that can be instantiated, e.g. *BinaryPredicate*; types that represent SUMO *Attributes* are distinguished with *isAttribute=true* field.

Each instance of *RelationalSentence* is linked to one *Relation* (*basicRelation* role) and many

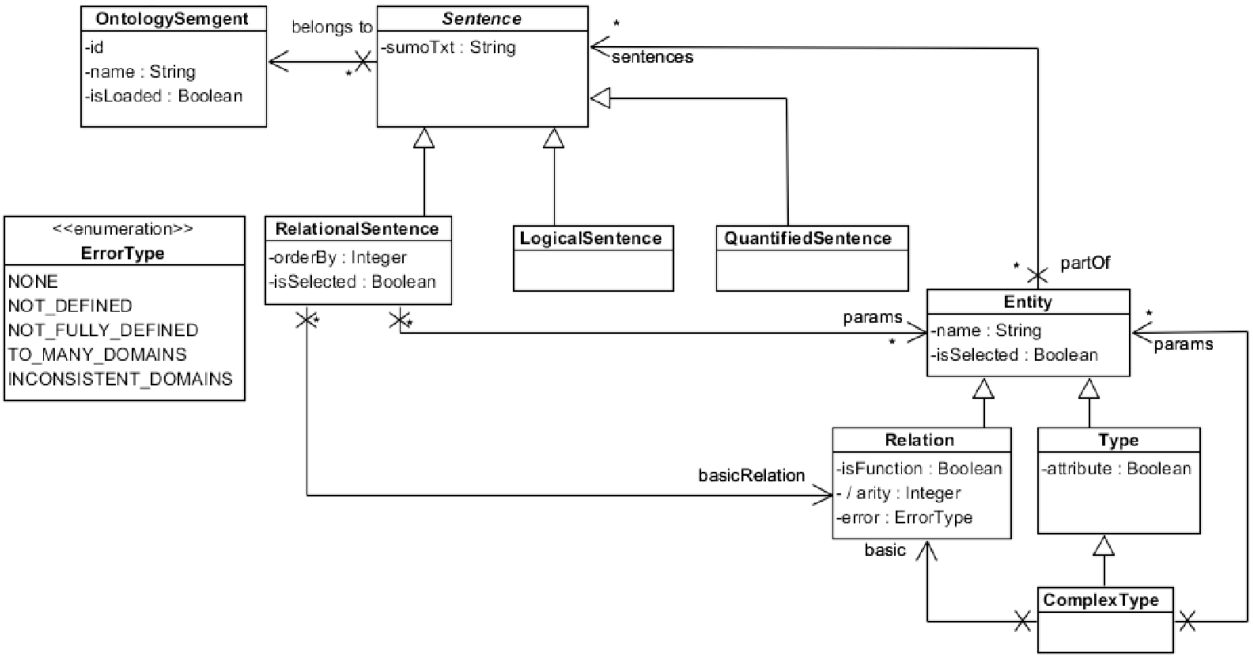


Figure 1. Meta-model of SUMO notions – main elements

Entities involved (*params* role), e.g. the sentence: "(domain part 1 Object)" is linked to *domain* relation, and has three parameters.

Sometimes relational sentences point out a type indirectly by referencing to a function which returns a type; see the sentence: "(subclass Fodder (FoodForFn DomesticAnimal))" for example. *Fodder* is a subclass of the type returned by the function *FoodForFn* called with the *DomesticAnimal* parameter. According to the specification this function returns a subclass of *SelfConnectedObject*. Such cases are represented in the proposed SUMO meta-model by a *ComplexType* class. An instance of the *ComplexType* class refers to the function it is built upon (*basic* role) – *FoodForFn* – and remembers the function parameters (*params* role) – *DomesticAnimal*.

Some specific relational sentences (defined in the SUMO upper ontology) play a crucial role in the transformation process. Up to now seven types of such sentences have been identified:

1. *Documentation* sentence (*DocumentationSent*) – a sentence starting with "(documentation...)"; contains documentation (an instance of *SymbolicString*) in a specific language for a specific entity;
2. *Instance* sentence (*InstanceSent*) – a sentence starting with "(instance...)"; is associated with an entity (instance) and a type for that instance;
3. *Subclass* sentence (*SubclassSent*) – a sentence starting with "(subclass...)"; used to describe inheritance hierarchy between SUMO classes; it is associated with parent and child types;
4. *Subrelation* sentence (*SubrelationSent*) – a sentence starting with "(subrelation...)"; allows to describe the inheritance hierarchy between SUMO relations; it is associated with parent and child relations;
5. *Domain* sentence (*DomainSent*) – a sentence starting either with "(domain...)" or "(domainSubclass...)"; it represents the domain element (*Type*) for a specific relation;
6. *Range* sentence (*RangeSent*) – a sentence starting either with "(range...)" or "(rangeSubclass...)"; represents a range (*Type*) for a function (*Relation* with *isFunction* attribute set to true);
7. *Partition* sentence (*PartitionSent*) – a sentence starting either with "(partition...)" or "(disjointDecomposition...)" or "(exhaustiveDecomposition...)"; all sentences represent partition of class C into subclasses but

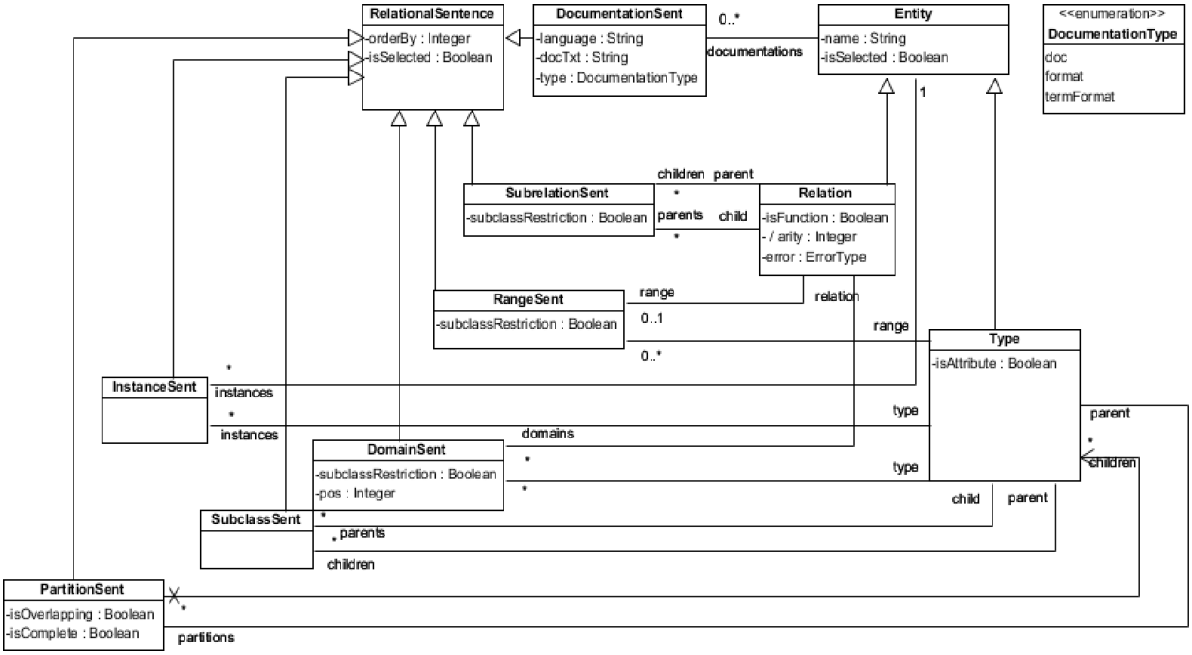


Figure 2. Meta-model of SUMO notions – hierarchy of relational sentences

they are characterized by different properties represented by *PartitionSent* attributes (*isOverlapping*, *isComplete*); i.e. a normal *partition* assumes that the subclasses are mutually disjoint and cover C; *disjointDecomposition* requires only that the subclasses are disjoint; and *exhaustiveDecomposition* disallows to have instances of C which do not belong to any of its subclasses (the subclasses do not need to be disjoint).

4. Architecture of the SUMO to UML Translator

The SUMO to UML translator is implemented in Java 8 with the Swing library. The main functional elements of the translator are presented on a component diagram – see Fig. 3.

An end-user is allowed to select any subset of ontology SUO-KIF files (called ontology segments) to be read by the tool. The loading process is controlled by a *SumoLoadConttroller* component and is presented – with the use of a sequence diagram – in Fig. 4.

SumoLoadController runs *SumoParser* to: (a) check the syntax correctness of the file, (b) walk through all tokens in the file and call *SumoModelBuilder* to translate SUMO sentences into an internal SUMO meta-model representation. *SumoParser* was generated with antlr [21] from the SUO-KIF context-free grammar [22].

Unfortunately, it turned out that SUMO ontology suffered from some bugs that could not be found by the parser (according to the rules formulated in context-free grammar). The bugs could negatively influence the correctness of the intended transformation process. So, there was a strong need to implement the *SumoChecker* component whose main functionality is to perform different consistency checks. The buggy elements are marked and reported by the tool, so the user has an opportunity to correct the input.

As it was mentioned in the previous Section, domain knowledge is spread over different SUO-KIF files which is not very convenient for transformation. That is why a separate component – *SumoReasoner* – was introduced. Its main responsibility is to update the previously generated SUMO model by inferring information indirectly defined in SUMO, e.g.: a subrelation could

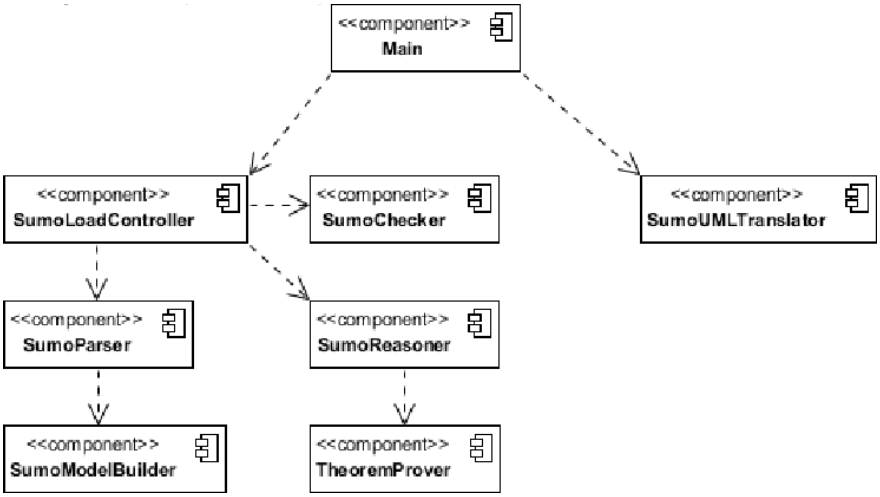


Figure 3. Architecture (functional view) of SUMO to UML translator

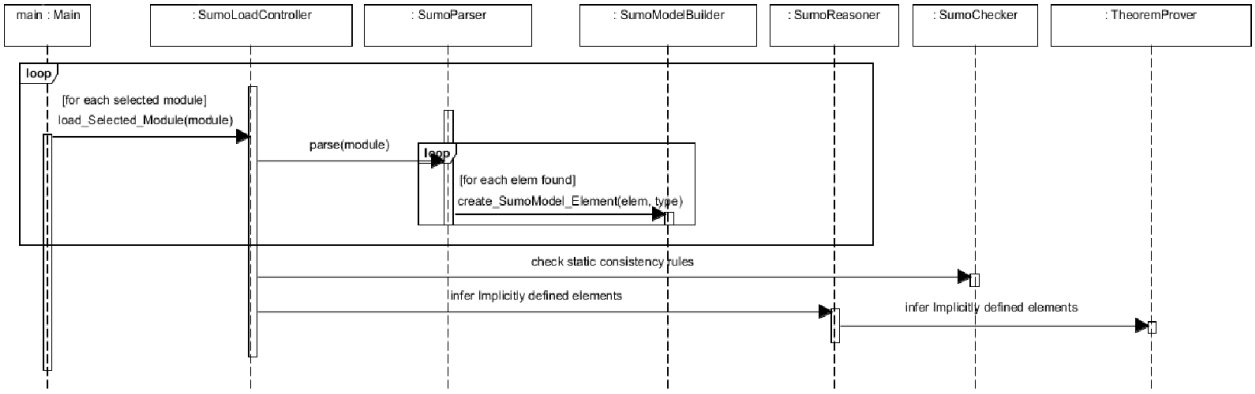


Figure 4. Processing of an ontology segment

inherit the domain definition from its parents; in such a case *SumoReasoner* copies domains from the parent to all its children.

It is also planned (this feature has not been implemented yet), that *SumoReasoner* will communicate with a selected theorem prover to reason knowledge from the rules. The new version of the Sigma tool [10] is prepared to collaborate with E prover [23]. E prover can deliver answers for specifically marked conjecture formulas. Sigma implemented mapping rules between SUO-KIF and TPTP formalism used by E prover. In consequence, a user can formulate questions like: "(instance ?X BinaryPredicate)" to find out all instances of *BinaryPredicate*.

The *SumoUMLTranslator* component conducts the transformation process. It produces – with the use of eclipse.emf and eclipse.uml2 frameworks – an instance of a UML model (version 2.5 [24]) and stores it in a file (*.uml) which can be read in a form of a tree or can be visualized on a diagram with additional tools, like e.g. Papyrus [25].

5. Translation of SUMO Attributes and Their Relations to UML

This section presents a proposal of SUMO attributes translation to UML.

5.1. Attributes and Attributes' Relations in SUMO

The *Attribute* in SUMO is a subclass of the *Abstract* class. Instances of the *Abstract* class "can-

not exist at a particular place and time without some physical encoding or embodiment" [8]. In other words, attributes represent some properties or the characteristics of instances.

The *Attribute* class has two direct subclasses (*InternalAttribute*, and *RelationalAttribute*) which in turn have many own subclasses. The hierarchy of attributes is more than five levels deep.

Attribute as a class is the domain of several SUMO relations (given below in an alphabetical order):

- *contraryAttribute*: *Attribute* x . . . x *Attribute* – is used to define "a set of Attributes such that something cannot simultaneously have more than one of these Attributes. For example, (contraryAttribute Pliable Rigid) means that nothing can be both Pliable and Rigid" [8];
- *exhaustiveAttribute*: *AttributeSubclass* x *Attribute* x . . . x *Attrbute* – "relates a class to a set of Attributes, and it means that the elements of this set exhaust the instances of the class. For example, (exhaustiveAttribute PhysicalState Solid Fluid Liquid Gas Plasma) means that there are only five instances of the class PhysicalState" [8];
- *subAttribute*: *Attribute* x *Attribute* – means that "the second argument can be ascribed to everything which has the first argument ascribed to it" [8]; it is a partial ordering relation which means that the hierarchy of attributes can form a tree;
- *successorAttribute*: *Attribute* x *Attribute* – means that the second attribute comes immediately after the first attribute on the scale that they share, e.g. "(successorAttribute DeluxeRoom SuiteRoom)"; *subAttribute* tuples have nothing in common (are disjoint) with *successorAttribute* tuples; moreover, *successorAttribute* is not a partial ordering relation which means that the involved attributes must be directly ordered;
- *successorAttributeClosure*: *Attribute* x *Attribute* – means that there is a chain of *successorAttribute* assertions connecting the first and the second parameter, e.g. "(successorAttributeClosure StandardRoom SuiteRoom)".

An assignment of an attribute instance to an entity instance can be done with a *property* relation (or one of its subrelations), e.g. "(property ?Entity ?Attr)" means that ?Entity has the attribute ?Attr.

The extended version of SUMO meta-model, covering the newly introduced relations, is presented in Fig. 5. The *successorAttributeClosure* relation is not included as it will not be translated to the UML. The meta-class representing *contraryAttribute* relation (*contraryAttributeSent*) inherits all necessary assotiations from its parent.

5.2. Mappings of SUMO Attributes and Attributes' Relations to UML

5.2.1. Mappings of SUMO Attributes

Transformations of SUMO notions to UML should preserve the original semantics as much as it is possible. An existing transformation rule maps any SUMO class to a UML class with the same name. This rule needs to be refined for attributes (understood as classes). As an attribute can have many instances (e.g. *Solid*, *Fluid*, *Liquid*, *Gas*, *Plasma* are instances of *PhysicalStateemph* attribute), it would be valuable to represent directly these instances on a UML class diagram. So this is why the *Attribute* class and their subclasses are mapped to a UML enumeration data type with the same name. "As a specialization of classifier, enumeration can participate in generalization relationships" [8]. This feature enables to represent also the inheritance hierarchy between *Attribute* subclasses. An enumeration value corresponds to one of user-defined enumeration literals.

These literals are used to represent attribute instances.

Not all relations between SUMO attributes can be represented graphically on a class diagram. Fortunately, UML is a very flexible language which can be extended for a specific purpose with the use of profiles.

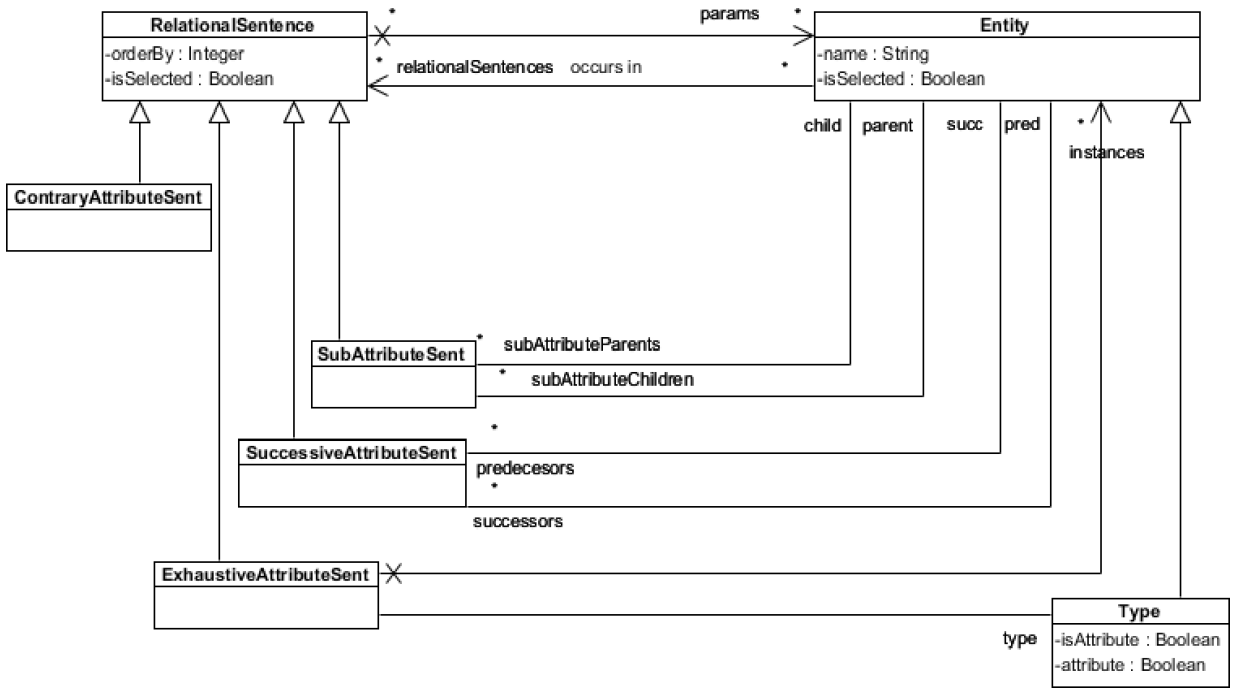


Figure 5. Extended version of the SUMO meta-model – definition of attributes' relations

5.2.2. UML Profile for Modelling SUMO Attributes

A UML profile is a lightweight extension mechanism to the UML by defining custom stereotypes, tagged values, and constraints. Profiles allow to adapt the UML metamodel for different domains [26]. UML profiles were defined for other ontology languages, e.g. OWL [27]. In the paper "UML Profile for OWL" authors define two-way mappings between the ontology definition meta-model (ODM) and the ontology UML profile.

The UML profile is defined as a specific package, containing stereotypes and constrains. These stereotypes can have meta-attributes called tagged values. "A stereotype is a profile class which defines how an existing metaclass may be extended as part of a profile. It enables the use of a platform or domain specific terminology or notation in place of, or in addition to, the ones used for the extended metaclass" [27].

UML profile for SUMO attributes introduces only two stereotypes (see Fig. 6):

– «Attribute» which is applied to enumerations, and
– «AttributeInstance» which is applied to enumeration literals being owned by the enumeration with «Attribute» stereotype; this stereotype has one property (pos: Integer), which introduces a tag definition; its semantics is explained in subsection 5.1.

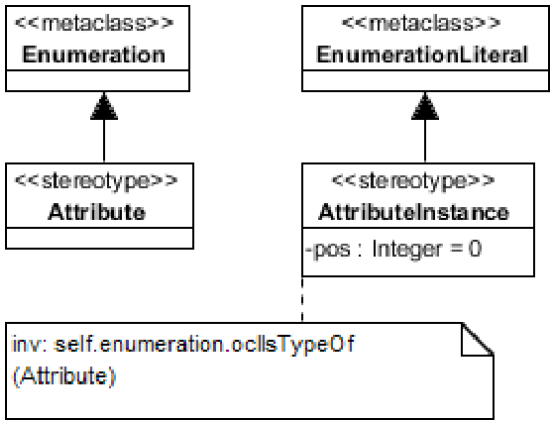


Figure 6. UML profile to represent SUMO attributes

5.2.3. Mappings of Attributes' Relations

This subchapter defines possible mappings for all relations between SUMO attributes, identified in Section 5.1, to UML language.

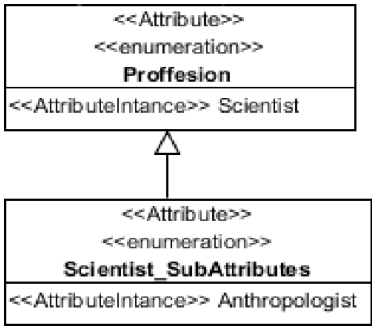


Figure 7. Transformation of the *subAttribute* relation

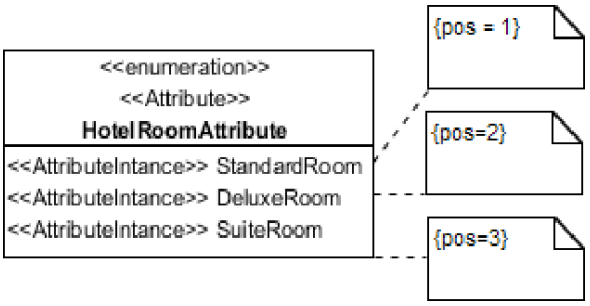


Figure 8. Transformation of *successorAttribute* relation

Transformation of *contraryAttribute* relation

The *contraryAttribute* relation is used to describe the fact that two specific attributes cannot be assigned to the same instance. Such a demand can be represented by an Object Constraint Language (OCL) invariant. OCL [28] is the language which enables to formally define constraints on UML models. Thus, any SUMO sentence of the form "(contraryAttribute atr1 atr2)" will be transformed as an invariant defined in the context of *Entity* class, according to the schema:

```
context Entity:
inv: not Entity.allInstances()->exists(e |
     e.hasProperty('atr1')
     and e.hasProperty('atr2'))
```

where *hasProperty(name: String): Boolean* is an auxiliary function which checks whether a specific entity e has assigned the attribute with a name equal to the input parameter.

Transformation of the *exhaustiveAttribute* relation

The *exhaustiveAttribute* relation lists all instances of a given attribute class. The list of instances cannot be further extended. To achieve the same semantics in the UML language, the UML class representing a SUMO attribute will be marked as a leaf class (*isLeaf* = *true*).

Transformation of *subAttribute* relation

The *subAttribute* relation defines the hierarchy of attribute instances. One attribute instance can be a parent for many sub-attributes, e.g. "(subAttribute Antropologist Scientist)", "(subAttribute Archeologist Scientist)". It would be valuable to present all these sub-attributes directly on a UML class diagram in the same way the other attributes' instances are represented, i.e. as enumeration literals. However, the children of a specific instance should be grouped together.

To achieve the demands mentioned above the following transformation rule is proposed. Each sentence of the form "(subAttribute atrSpec atrGen)" will be transformed according to the schema:

- If it does not exists a new artificial enumeration data type with «Attribute» stereotype and name *atrGen_SubAttributes* is created, e.g. *Scientists_SubAttributes*; the newly created enumeration will inherit from the enumeration data type for which *atrGen* is an enumeration literal; in the example the *Scientists_SubAttributes* enumeration data type will inherit from the *Proffesion* enumeration data type (see Fig. 7);
- *atrSpec* is defined as a new enumeration literal in the *atrGen_SubAttributes* enumeration data type; e.g. the *Anthropologist* enumeration literal will be added to the *Scientists_SubAttributes* enumeration.

Transformation of the *successorAttribute* relation

The *successorAttribute* relation defines the direct order between attributes. Such an order can be represented by UML tag definitions ({pos = value}). An attribute instance which is the first "in the queue" will have pos set to 1, its direct successor – pos set to 2, etc. For example, see Fig. 8 on which the transformation of SUMO sentence: "(successorAttribute StandardRoom DeluxeRoom)" is presented.

Transformation of the *successorAttributeClosure* relation

The *successorAttributeClosure* relation can be in-

ferred from *successorAttribute* relation, and this is why it is not translated to UML.

6. Examples of Transformation Rules

This section shortly presents the implemented transformation rules focusing on those that were changed in comparison to the previous publications [15–17]. Selected transformation rules are described below.

6.1. Rule 1

SUMO Element: Direct or indirect subclass of *Entity*, e.g. *City*, *Nation*
UML Element: Class
Comment: Data values like *Integers* are also represented as separate classes (which results in uniform representation of relations).

6.2. Rule 2

SUMO Element: Binary (including self) and higher arity relations with all domains defined in the form "(domain *relation int class*)", e.g. "(domain citizen 1 Human)", "(domain citizen 2 Nation)"
UML Element: Association with a proper arity, e.g. *citizen*, *capitalCity*
Comment: Previously, when one of the domains in a relation was a data value, e.g. *Integer*, the relation was represented either as an attribute (for a binary relation) or an association class; now, all binary or higher arity relations are represented in the same way as associations.

6.3. Rule 3

SUMO Element: A relation domain or a function range defined in the form "(domainSubclass *relation int class*)", e.g. "(domainSubclass roomAmenity 1 HotelUnit)", or "(rangeSublcass *function class*)", e.g. "(rangeSublcass FoodForFn SelfConnectedObject)"
UML Element: Generalization set, e.g. *HotelUnit_Subclasses*, *SelfConnectedObject_Subclasses*
Comment: *domainSublcass* is a constraint meaning that the int'th element of each tuple in relation must be a subclass of a specific class; similarly, *rangeSubclass* stays the same for function ranges; that this notion is represented by the UML generalization set.

6.4. Rule 4

SUMO Element: Binary (including self) and higher arity relations for which at least one domain is defined in the form "(domainSublcass *relation int class*)", e.g. "(domainSublcass roomAmenity 1 HotelUnit)", "(domainSubclass roomAmenity 2 Physical)"
UML Element: Association among the results of the transformation of relation domains including generalization sets, e.g. *roomAmenity* (association between *Physical_Subclasses* and *HotelUnit_Subclasses*)
Comment: The previous transformation was incorrect (misinterpreted semantics); the association used to link classes; the new association links generalization sets.

6.5. Rule 5

SUMO Element: Subrelation relationship "(subrelation *child-relation parent-relation*)" e.g."(subrelation geographicSubregion located)"
UML Element: An association with a "subsetted" property; the association ends of the *child-relation* will be the subsets of association ends of *parent-relation*; e.g. *geographicSubregion* association ends will be the subsets of *located* association ends
Comment: A *subrelation* is a constraint meaning that every tuple of a child relation is also a tuple of a parent relation; in the UML 2.5 such a feature is represented by a subset constraint.

6.6. Rule 6

SUMO Element: Partition relationship in the form "(partition *C C1 C2...*)"
UML Element: Generalization set with *isOver-*

lapping=false and *isComplete=true*
Comment: New.

6.7. Rule 7

SUMO Element: Exhaustive decomposition relationship in the form "(exhaustiveDecomposition *C C1 C2...*)"
UML Element: Generalization set with *isOverlapping=true* and *isComplete=false*
Comment: New.

6.8. Rule 8

SUMO Element: Disjoint decomposition relationship in the form "(disjointDecomposition *C C1 C2...*)"
UML Element: Generalization set with *isOverlapping=false* and *isComplete=false*
Comment: New.

6.9. Rule 9

SUMO Element: *Attribute* class or its subclass
UML Element: Enumeration data type with «Attribute» stereotype
Comment: New.

6.10. Rule 10

SUMO Element: *Attribute* instance
UML Element: Enumeration literal with «AttributeInstance» stereotype in the enumeration data type
Comment: New.

6.11. Rule 11

SUMO Element: subclass relation between *Attribute* classes, e.g. "(subclass HotelRoomAttribute RelationalAttribute)"
UML Element: Generalization relationship between enumerations
Comment: New.

6.12. Rule 12

SUMO Element: *contraryAttribute* relation, e.g. "(contraryAttribute Dirty Clean)"
UML Element: An OCL invariant defined in the context of *Entity* class
Comment: New.

6.13. Rule 13

SUMO Element: *exhaustiveAttribute* relation, e.g. "(exhaustiveAttribute SexAttribute Female Male)"
UML Element: Property *isLeaf* in the class representing the attribute is set to true
Comment: New.

6.14. Rule 14

SUMO Element: *subAttribute* relation, e.g. "(subAttribute Anthropologist Scientist)"
UML Element: A new enumeration data type gathering all sub attributes (left parameter) of the right parameter as literals; this new data type inherits from the enumeration data type representing the right parameter
Comment: New.

6.15. Rule 15

SUMO Element: *successorAttribute* relation, e.g. "(successorAttribute StandardRoom DeluxeRoom)"
UML Element: Tag definitions assigned to enumeration literals with pos tag set to the order number of the attribute instance
Comment: New.

7. SUMO to UML Transformation Example

The functionality of SUMO to UML translator will be presented with the use of a simple example. It aims at elaborating an initial version of domain diagram based on the *Countries and Regions* ontology and the ontologies it is based upon (e.g. Merge.kif, Mid-level-ontology.kif, Goverment.kif, all downloaded on the 1st January 2016) [8]. Figure 9 shows a form which allows a user to select interesting ontologies (ontology segments).

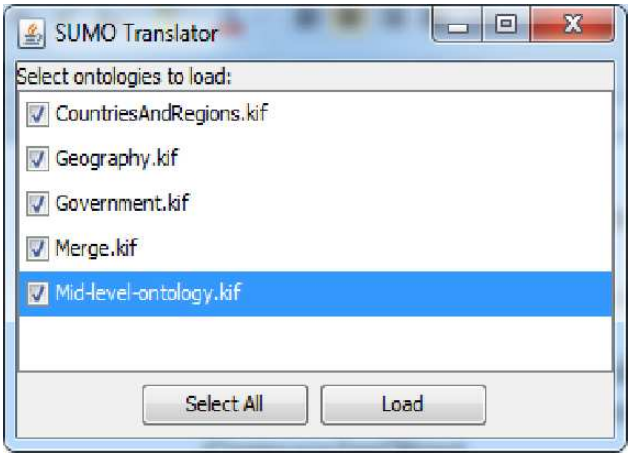


Figure 9. SUMO to UML translator – the initial form

After file loading the *SumoChecker* component reports found bugs. SUMO sentences which are the source of bugs are marked in red in the main window. Examples of such bugs are presented below:

- Entity: *DeviceNormal* has two different informal documentation sets.
- Relation: *defendant* 1st domain: *CognitiveAgent* does not fit parent: *patient* domain: *Process*.
- Type: *PostalAddressText* lacks its documentation.

Let us assume that a user wants to propose a domain model to represent the structure of geographic areas, their types, inclusions, as well as capital cities for particular geopolitical regions. He needs to find among SUMO notions those to be translated to UML and to mark them. The tool helps to identify interesting concepts by providing all sentences in which a given concept is used, grouped by their type; for example, for a relation the documentation sentence is presented first, next relation domains, sub-relations and relation instances (see Fig. 10).

Within the main window, a user can search or browse SUMO content. On the left there is a list of all entities found in selected SUMO ontologies. Because the number of entities is huge, the view could be limited only to entities whose names start with a specific letter. On the right, there is a set of sentences the entity is part of. There is also Rule tab containing axioms referring to a selected entity.

By a double click a user can select either entities or sentences to be translated to UML. Selected elements are marked in yellow – see Fig. 10. If a relation is selected, its domains are automatically selected as well. For example, among relationships in which the *City* class is involved, *capitalCity* was chosen to be translated into UML. When the selection process is completed, the user runs the translation process.

Figure 11 presents the result of a transformation made by the translator. The resulting UML class diagram has a form of a tree with properties set for classes and associations.

For readability purposes the generated file was rewritten in the Visual Paradigm tool and presented as a graph in Fig. 12. Examples of elements that cannot be visualized (e.g. *subset* constraint for association ends) are given in comments.

As one can observe, the resulting class diagram may consist of more than one sub-graphs – see the *located* association between *Object* and *Physical* classes. There could be the following reasons for that:

- The user did not mark SUMO sentences describing the inheritance hierarchy to be transformed; e.g. *GeographicArea* is an indirect child of *Object* and *Physical* which means, that – in this context – *located* relation can happen between *GeographicAreas*.
- Some knowledge is contained in qualified sentences which are not processed at that moment in any way.

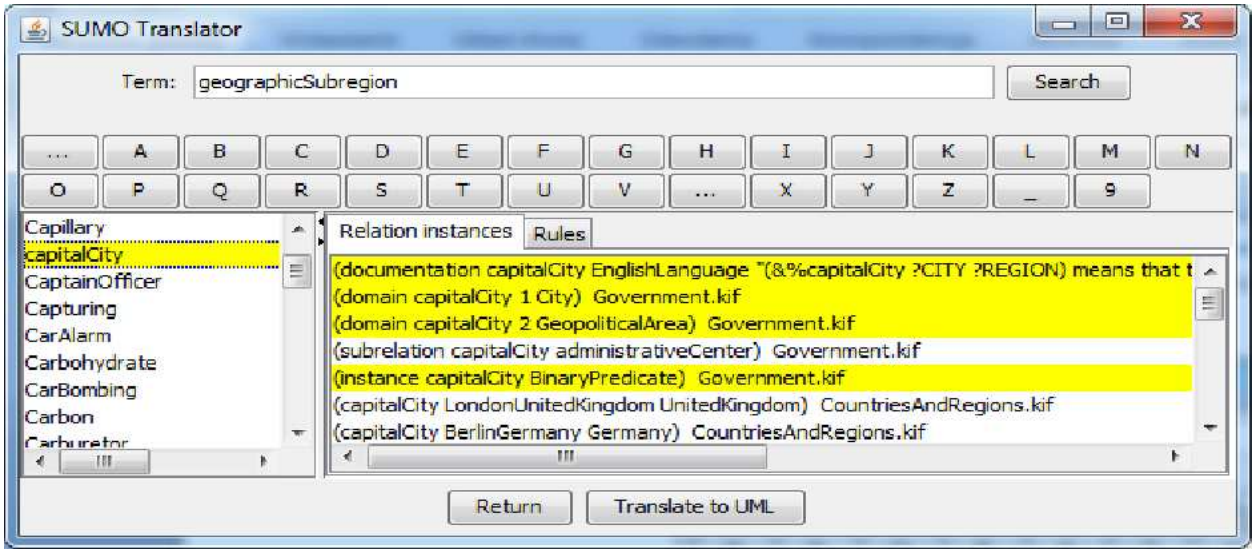


Figure 10. SUMO to UML translator – the main window

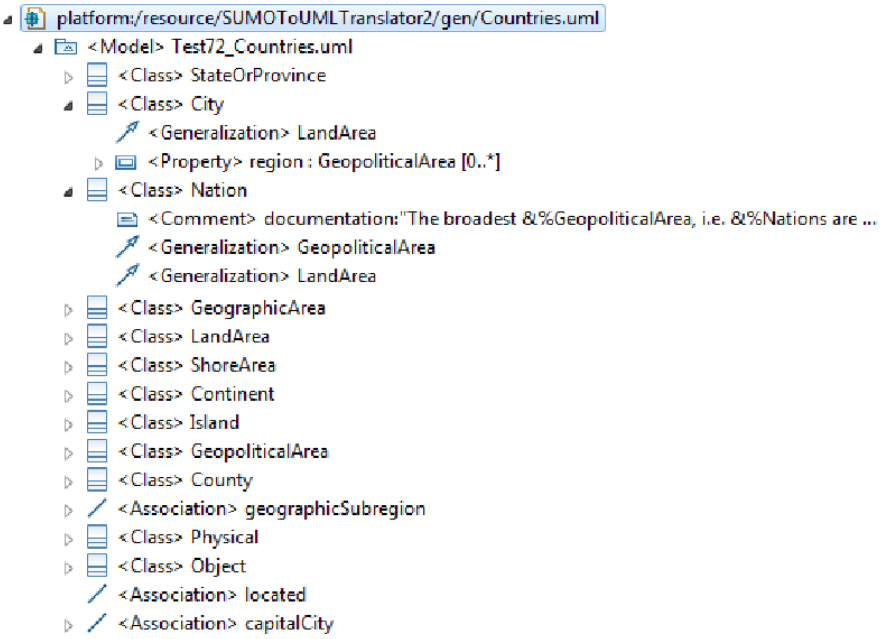


Figure 11. SUMO to UML transformation example (automatic translation)

SUMO ontologies form a set. The upper layer is included in Merge.kif file. At this level, many basic relations are defined, including *located*, so this is why their domains are top classes from SUMO class hierarchy (*Physical* and *Object* for *located* relation). When considering a specific domain, e.g. countries and regions, one deals with subclasses of the top level classes; the instances of these subclasses can be used in all places where their parents are allowed. It means that an interesting relation could be defined between classes being far away (in the inheritance hierarchy) from classes of the considered domain. To solve this problem, the translation tool can add indirect inheritance relationships between classes presented on the class diagram.

The domain diagram resulting from the transformation process is a starting point to understand a given domain. It is consistent with domain ontology by construction, but it can lack

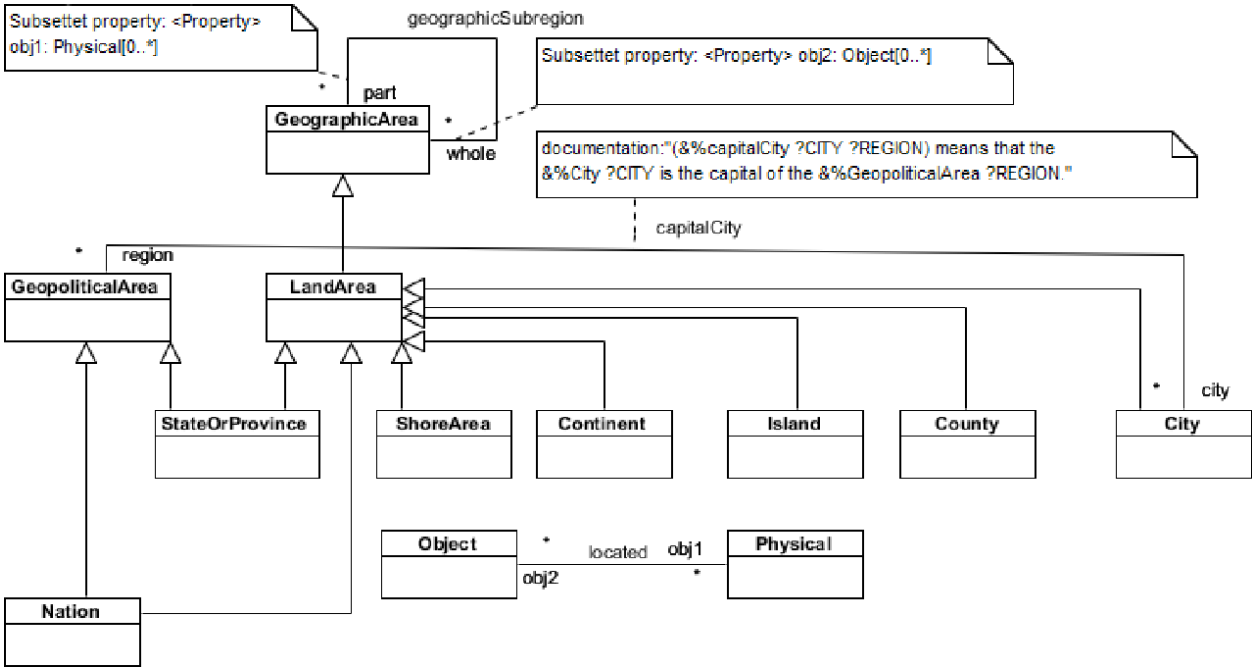


Figure 12. SUMO to UML transformation example – results presented as a class diagram

some important information. The quality of the diagram strongly depends on the initial step performed by a system analyst – identification of SUMO notions to be translated. This problem is addressed in [29].

8. Problems to be Addressed

8.1. Meta-classes and Meta-relations

SUMO, similarly to UML, is described in SUMO itself. Some elements of SUMO play the role of meta-classes, i.e. classes the instances of which are functions or relations; examples include *BinaryPredicate*, *IrreflexiveRelation*. Meta-classes are not directly translated to a UML class diagram, but they define important properties of other transformed elements, e.g. the arity of relations or functions. At this moment, only arity is transformed. Another relation properties, e.g. the "reflexivity" constraint is not translated, but that could be done with the use of OCL.

Meta-relations are the relations describing relationships between 2 or more classes or 2 or more relations; examples include: *subclass*, *partition*, *disjoint* for classes, and *subrelation*, *disjointRelation* for relations. In the current version of the tool most of them are addressed (see Sections 4–6 for details) but still some other can be taken into consideration, e.g. *disjointRelation*.

8.2. Axioms

SUMO axioms introduce constraints on ontology instances. The example below states that every instance of *EuropeanCity* must be part of *Europe*.

```
(=>
(instance ?CITY EuropeanCity)
(part ?CITY Europe)
)
```

The other example stays that if an instance belongs to *VirginIslands* it must be also an instance of *Island*.

```
(=>
(member ?ISLAND VirginIslands)
(instance ?ISLAND Island)
)
```

Some of such axioms could be expressed directly in UML (e.g. with the use of a composition relationship), some other could be translated into OCL. The current version of the SUMO to UML

transformation tool allows reading axioms but they cannot be selected for transformation.

9. Summary

The paper presents an approach to SUMO-UML translation. The translation is defined as a set of transformation rules between SUMO and UML meta-models.

The SUMO meta-model was proposed for this purpose. The initial set of transformation rules, identified and described in [15–17], was revised and extended with new rules e.g. for SUMO attributes and their relations.

The results of the tool applications are promising. The obtained domain class diagrams are consistent, correct and complete to the level to which the input ontology has these features. These are the main benefits the tool can bring to potential users. Business expert or business analyst can use the tool to find out interesting notions, select them, and translate to a UML class diagram with a set of OCL constraints with one click. The user is warned about incompleteness and inconsistencies found in the original files. He or she can experiment with transformation results, selecting new elements or un-selecting the previously selected ones. The obtained UML model can be re-factored, and next transformed to other representations, e.g. programming languages, database schemas, etc.

The tool to be effectively used needs a qualified business analyst or business expert to select all interesting SUMO notions for transformation. Otherwise, the resulting domain model will be incomplete. To address this matter a research group, the author of this paper belongs to, is trying to propose an algorithm to extract knowledge from ontology on the basis of limited input only – see [29].

A kind of a side effect of the tool implementation is the definition of static consistency rules which allow to detect inconsistencies in existing ontologies. In the future, this module can be used as a part of an ontology editor.

The next release of the tool will address problems presented in Section 8. Additionally, the transformations at the instance level, represented by object diagrams, are planned to be implemented. It seems to be especially important because in domain ontologies more than half of sentences represent instances and links among them, e.g. "(instance Mauritius Nation) (geographicSubregion Mayotte SouthernAfrica)" for CountiresAndRegions.kif.

References

[1] K. Bittner and I. Spencer, *Use Case Modeling*. Addison-Wesley Professional, 2002.

[2] I. Jacobson, G. Booch, and J. Rumbaugh, *The Unified Software development process*. Addison-Wesley Professional, 1999.

[3] D. Zowghi and V. Gervasi, "The three cs of requirements: Consistency, completeness, and correctness," in *Proceedings of 8th International Workshop on Requirements Engineering: Foundation for Software Quality, (REFSQ '02*, 2002, pp. 155–164.

[4] P. Mohagheghi, V. Dehlen, and T. Neple, "Definitions and approaches to model quality in model-based software development – a review of literature," *Information and Software Technology*, Vol. 51, No. 12, Dec. 2009, pp. 1646–1669.

[5] W. Hesse, "Ontologies in the software engineering process," in *EAI 2005: Enterprise Application Integration – Proceedings of the 2nd GI-Workshop on Enterprise Application Integration*, R. Lenz, U. Hasenkamp, W. Hasselbring, and M. Reichert, Eds., 2005.

[6] H.J. Happel and S. Seedorf, "Applications of ontologies in software engineering," in *Proc. of Workshop on Sematic Web Enabled Software Engineering"(SWESE) on the ISWC*, 2006, pp. 5–9.

[7] F. Gailly and G. Poels, "Conceptual modeling using domain ontologies: Improving the domain- specific quality of conceptual schemas," in *Proceedings of the 10th Workshop on Domain-Specific Modeling*, ser. DSM '10. New York, NY, USA: ACM, 2010, pp. 18:1–18:6.

[8] Suggested Upper Merged Ontology, last access: 10 Jan 2016. [Online]. http://www.ontologyportal.org

[9] A. Pease, *Ontology: A practical Guide*. Articulate Software Press, 2011.

[10] Sigma, last access: 10 Jan 2016. [Online]. http://sourceforge.net/projects/sigmakee/files/

[11] I. Istochnick, OWL2UML, last access: 10 Jan 2016. [Online]. http://protegewiki.stanford.edu/wiki/OWL2UML

[12] F. Suchanek, "Ontological reasoning for natural language understanding," Master Thesis in Computer Science, Saarland University, Germany, March 2005.

[13] Semantics of Business Vocabulary and Business Rules (SBVR). Version 1.3, OMG, (2015, May). [Online]. http://www.omg.org/spec/SBVR/1.3/

[14] A. Marinos, S. Moschoyiannis, and P.J. Krause, "An SBVR to SQL compiler," in *Proceedings of the RuleML-2010 Challenge, at the 4th International Web Rule Symposium*, 2010.

[15] B. Hnatkowska, Z. Huzar, I. Dubielewicz, and L. Tuzinkiewicz, "Problems of SUMO-like ontology usage in domain modelling," in *Intelligent Information and Database Systems*, ser. Lecture Notes in Computer Science, N. Nguyen, B. Attachoo, B. Trawinski, and K. Somboonviwat, Eds. Springer International Publishing, 2014, Vol. 8397, pp. 352–363.

[16] I. Dubielewicz, B. Hnatkowska, Z. Huzar, and L. Tuzinkiewicz, "Domain modelling in the context of ontology," *Foundations of Computing and Decision Sciences*, Vol. Volume 40, No. 1, 2015, pp. 3–15.

[17] B. Hnatkowska, Z. Huzar, I. Dubielewicz, and L. Tuzinkiewicz, "Development of domain model based on SUMO ontology," in *Theory and Engineering of Complex Systems and Dependability*, ser. Advances in Intelligent Systems and Computing, W. Zamojski, J. Mazurkiewicz, J. Sugier, T. Walkowiak, and J. Kacprzyk, Eds. Springer International Publishing, 2015, Vol. 365, pp. 163–173.

[18] D. Gasevic, D. Djuric, V. Devedzic, and V. Damjanovi, "Converting UML to OWL ontologies," in *Proceedings of the 13th International World Wide Web Conference on Alternate Track Papers &Amp; Posters*, ser. WWW Alt. '04. New York, NY, USA: ACM, 2004, pp. 488–489.

[19] J. Zedlitz, J. Jörke, and N. Luttenberger, *Knowledge Technology*. Berlin, Heidelberg: Springer-Verlag, 2012, ch. From UML to OWL 2, pp. 154–163.

[20] B. Hnatkowska, *From requirements to software: research and practice*. Warszawa: Polish Information Processing Society, 2015, ch. Towards automatic Sumo to UML translation, pp. 87–99.

[21] ANTLR, last access: 10 Jan 2016. [Online]. http://www.antlr.org/

[22] A. Pease, Standard upper ontology knowledge interchange format, (2009). [Online]. http://sigmakee.cvs.sourceforge.net/viewvc/sigmakee/sigma/suo-kif.pdf

[23] S. Schulz, "System description: E 1.8," in *Logic for Programming, Artificial Intelligence, and Reasoning*, ser. Lecture Notes in Computer Science, K. McMillan, A. Middeldorp, and A. Voronkov, Eds. Berlin Heidelberg: Springer-Verlag, 2013, Vol. 8312, pp. 735–743.

[24] Unified Modeling Language. Version 2.5, OMG, (2013, September). [Online]. http://www.omg.org/spec/UML/

[25] Papyrus modeling environment, last access: 10 Jan 2016. [Online]. http://www.eclipse.org/papyrus/

[26] UML profile diagrams, last access: 28 May 2016. [Online]. http://www.uml-diagrams.org/profile-diagrams.html

[27] D. Djurić, D. Gašević, V. Devedžic, and V. Damjanović, *Proceedings of the Web Engineering: 4th International Conference*. Berlin, Heidelberg: Springer-Verlag, 2004, ch. UML Profile for OWL, pp. 607–608.

[28] Object Constraint Language. Version 2.4, OMG, (2014, February). [Online]. http://www.omg.org/spec/OCL/2.4/

[29] B. Hnatkowska, Z. Huzar, L. Tuzinkiewicz, and I. Dubielewicz, *Intelligent Information and Database Systems*, ser. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer-Verlag, 2016, Vol. 6592, ch. Conceptual Modeling Using Knowledge of Domain Ontology, pp. 554–564.

e-Informatica Software Engineering Journal, Volume 10, Issue 1, 2016, pages: 69–87, DOI 10.5277/e-Inf160104

Highly Automated Agile Testing Process: An Industrial Case Study

Jarosław Berłowski[a], Patryk Chruściel[a], Marcin Kasprzyk[a], Iwona Konaniec[a], Marian Jureczko[b]

[a] *NetworkedAssets Sp. z o. o.*

[b] *Faculty of Computer Science and Management, Wrocław University of Science and Technology*

marian.jureczko@pwr.edu.pl

Abstract

This paper presents a description of an agile testing process in a medium size software project that is developed using Scrum. The research methods used is the case study were as follows: surveys, quantifiable project data sources and qualitative project members opinions were used for data collection. Challenges related to the testing process regarding a complex project environment and unscheduled releases were identified. Based on the obtained results, we concluded that the described approach addresses well the aforementioned issues. Therefore, recommendations were made with regard to the employed principles of agility, specifically: continuous integration, responding to change, test automation and test driven development. Furthermore, an efficient testing environment that combines a number of test frameworks (e.g. JUnit, Selenium, Jersey Test) with custom-developed simulators is presented.

Keywords: software engineering, testing process, agile software development, case study

1. Introduction

Software testing is a very costly part of the software development process, it is sometimes estimated to make 50% of the whole development cost [1], [2]. It is one of the main activities (at least should be) in agile software development methods. Beck and Andres [3] claimed it to be a measure of project progress and the main mean of assuring software quality. On the other hand, applying agile software development methods significantly affects the testing process. The agile methods usually require to test early and to have the tests automated. In consequence, testing is not left to the final phase, but requires sustainable investments during the whole process (including after-release maintenance of automated test cases). Some of the authors even recommend to start with testing [4]. The agile testing has been studied for several years, however, there are still unanswered questions regarding scalability [5], the role of testers [6] or test automation [7].

The goal of this research is to extend the body of knowledge concerning agile testing by documenting a real life software testing process. This paper presents a case study of a medium-size software project with special factors that affects the aforementioned process, i.e. requests for unscheduled releases and high complexity of project environment. The project is a Java Enterprise system in the telecommunication domain and its main purpose is to ease network devices controlling and management. Thus, there is a number of features that concern integration and communication with other systems and devices. Functional tests of such features involve items that are outside of the tested system, but are necessary for a successful test execution. Therefore, the test environment is complex and its management can consume a considerable amount of resources, specifically in the case of automated tests, where all changes

that come from test execution should be verified and reverted in order to ensure tests repeatability. On the other hand there are many, unscheduled release requests that concern the newest version of the developed system. Those releases regard presentations for end users or deployments into an environment for acceptance tests. Nonetheless, the released version must satisfy company quality standards and must be 'potentially shippable' [8]. The unscheduled releases could be explained as releases in the middle of a Sprint (the project is developed using Scrum), that are announced one or two days in advance and are requested to contain some features from the current Sprint and, of course, all the features developed in previous Sprints. The unscheduled releases create challenges. It is critical to ensure that the 'old' functionality still works correctly and there are very limited resources for regression tests, since there are the 'in-progress' features that must be straightened up before the release. The solution is somewhat obvious – test automation. In order to support the unscheduled releases a complex set of automated regression tests must be present and frequently executed. In consequence, it is possible to evaluate the current version of software instantly and decide if it is 'potentially shippable'. The paper describes how the challenges are satisfied, in which ways they affected the testing process and what results were obtained.

The case study presents an insider perspective, i.e. it is conducted by members of the team that develops the project. Therefore, it is possible to get a really deep insight, but unfortunately the risk of biased observation or conclusion grows simultaneously. In consequence, the case study is focused on a quantitative approach, which is associated with lower risk of subjectivity instead of the qualitative one, which is usually employed in similar case studies. The study is designed according to guidelines given by Runeson and Höst [9].

A number of concepts is used in this paper. Let us present explanations of them in order to clarify possible ambiguities. The authors refer to **project size**. The concept is in fact the metric which Mall [10] identified as one of the two most popular and the simplest measures of project size, namely the number of Lines of Code (LOC). The project investigated here is described as **a medium-size software project**. The small, medium and large scale is not a precise term, it is very intuitive (e.g. more intuitive than LOC). The investigated project has been classified as a medium-size one, since it seems to be a bit smaller than projects described in [5] and [6] which are reported as large ones. The authors refer to **costs of development and test**. This should be considered as the amount of time committed by the development team during project related activities. One of the investigated aspects of the testing process is **availableness for unscheduled release**. A possibility of making an application release is evaluated using results of the functional tests (Selenium tests and REST tests) executed in a continuous integration system (100% success of unit tests is a prerequisite of the functional ones). The application is truly ready for release if 100% of the functional tests are passed. The application fits for a presentation in the case when at least 90% of the tests are passed. If more than 10% of the tests failed, no form of release shall be considered. The investigated testing process was assessed using the concept of **the level of agility** that evaluates the adoption of different agile principles using the concept of **the level of adoption**. Both aforementioned concepts are borrowed from [5], further details about them can also be found in section 2.3.

The rest of this paper is organised as follows. The next section presents the goal and study design as well as a detailed description of the study context. The obtained results are documented in section 3. In section 4, the threats to validity are discussed and in section 5, related work is summarized. Finally, the discussion and conclusions are given in section 6.

2. Goal and Study Design

2.1. Context

The context is described according to the guidelines given by Petersen and Wohlin [11]. It corresponds with the suggested description structure

as well as the recommended elements of the context.

2.1.1. Product

The investigated project regards the development of a system which is used in the management of telecommunication services delivery. The main functionalities of the product are:

- managing customer access devices,
- managing and configuring network delivery devices,
- managing resources of IP addresses from the public and private pools.

The following technologies are used in the software development:

- `GWT` – Google Web Toolkit is a development toolkit for building and optimizing complex browser-based applications (https://developers.google.com/web-toolkit/);
- `Spring` – a comprehensive programming and configuration platform for modern Java-based enterprise applications without unnecessary ties to specific deployment environments (http://www.springsource.org/);
- `Hibernate` – It is a framework which allows to provide the service of the data storage in a database, regardless of the database system (http://www.hibernate.org/);
- `Oracle DB` – Relational database management system which is commonly used in corporations because of data maintaining capabilities, security and reliability (http://www.oracle.com);

Maturity. The project was conducted from May 2011, the first release was carried out in Autumn 2011. The study was conducted in January 2013.
Quality. The main means of ensuring the product quality are tests. Further details regarding the testing process are presented in the next sections.
Size. The total number of Lines of Code is 327387 (calculated by StatSVN – all lines are counted).
System type. Enterprise system with a web UI. The development team makes use of functionalities accessible as REST services and also integrates a them with existing systems.
Customisation. It is custom software development. The product is tailored to the requirements of a customer (a telecommunication vendor).
Programming language. The system is developed mostly in Java.

2.1.2. Processes

Figure 1 presents the testing process. In fact, no defined process is used, and what the diagram shows is the result of the 'definition of done' that is being employed. New user stories are first identified, then acceptance criteria for them are defined and stories are estimated. When the Sprint begins, the development team makes the Sprint planning and later software development and unit tests. At the same time a specification for functional tests must be prepared. If pair programming was not carried out, than a code review is needed. Before the end of the Sprint, functional tests must be conducted. When there is time, automated functional tests are prepared, in other case, automation is postponed to the next Sprint. Test automation is not a part of the 'definition of done.' It might seem strange as automation is very important in agile processes, but unfortunately automation is very time-consuming and for technical reasons sometimes must be done after implementation (this remark does not regard unit tests). In consequence, it was not possible to always do implementation and automation in the same Sprint. Having the choice to either do longer Sprints or postpone automation it was preferable to postpone the automation as there is the requirement for unscheduled releases, which does not correspond well with long Sprints. Postponing test automation is not a good practice, but conducting functional tests manually [12, 13] or automating after the sprint [5,14] is not unique in software development. Puleio [14] even invented a name for not doing the test automation on time, i.e. testing debt. At the end of the Sprint the software is potentially shippable as the new features which are not covered by automated tests (if there are such) are tested manually. As presumably it already emerged from the above description, the employed development process is Scrum.

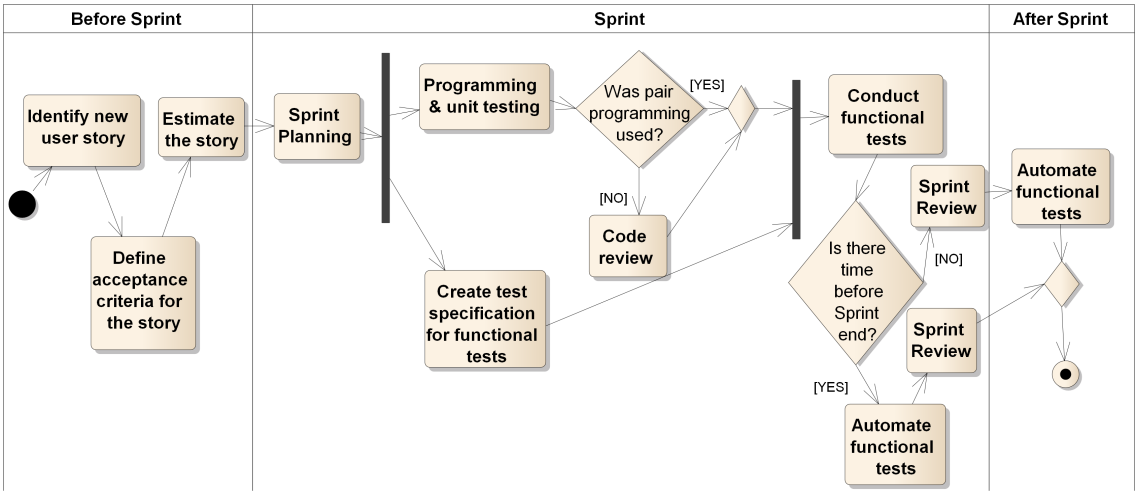


Figure 1. Testing process

2.1.3. Practices, Tools, Techniques

CASE tools

Eclipse STS (http://www.springsource.org/sts) is used as the primary development environment. Furthermore, the development team uses a set of frameworks for test automation which are employed in a continuous integration system (details in the next section).

Practices and techniques

- Time–boxing.
- Frequent (in fact continuous) integration.
- Frequent, small releases.
- Early feedback.
- Test automation.

2.1.4. People

Roles

- Product Owner – comes up with new ideas, updates priorities and chooses the most important issues, he is also responsible for contact with customers.
- Scrum Master – takes control of the Scrum process and enables team members to reach the goal of the Sprint by helping them with all impediments. For a short period of time there were two Scrum Masters, as the project was developed by two different teams.
- Team – responsible for reaching the goal of the Sprint. The development team currently consists of 8 people including: software developers, testers (one tester performs also the role of a business analyst), the Scrum Master (who takes part in software development) and the Product Owner (who does not take part in the software development):
  - Software developers (5),
  - Scrum Master (1),
  - Testers (2).

2.2. Experience

Some software developers have longiterm experience in the IT field, i.e. two people 5+ years of experience (one of them is a Scrum Master), the Product Owner 15+ years of experience and one of the testers who is also a business analyst 10+ years of experience. For the rest of the team, this is the first job. They take part in the project, learn technical skills and expand knowledge using different tools. The entire team has completed bachelor's or master's degree studies in computer science. In order to improve the qualifications, the members of the development team have participated in some trainings and certification programs, for example ISTQB (International Software Testing Qualifications Board) – both testers, Professional Scrum Master I – the Scrum Master, CISCO certificates (CCNA/CCNP, CCAI) – one of the testers.

The development process was configured by the experienced team members who also take care

of the introduction of their younger colleagues. Thus, there are no reasons to believe that there are issues in the process resulting a lack of experience. Additionally, the trainings and certification program were used to ensure high development standards and to avoid shortcomings.

2.3. The Level of Agility of the Investigated Testing Process

The paper is to be focused on agile testing process. Hence, it is important to take carefully analyze the principles of agility and assess the process. Otherwise, there would be a considerable risk of investigating other phenomena, not the ones that should be investigated.

2.3.1. Survey Design

The issue is addressed with a survey conducted among all the staff that is or was involved in project development. The survey is designed according to a set of weighted criteria suggested by Jureczko[1] [5], i.e. the following criteria are used (each of them can be met fully, partly or not at all):

- Test driven development – TDD is one of the most remarkable improvements, which has been brought to testing with the agile methods [15], [3]. Presumably not every agile testing process uses TDD, on the other hand TDD affects the process and system design in such a significant way, that it cannot be ignored when evaluating the level of agility (weight = 3).
- Test automation – most of the agile methods move the focus from manual to automated tests, e.g. [3]. Furthermore, the automation is often considered an essential practice [16], [14] (weight = 3).
- Continuous integration – what is the use of automated tests when they are not executed frequently? (weight = 2).
- Communication – it is considered at two levels, i.e. not only between team members but also with the customer, however, with respect to testing process the most relevant communication is between developers and testers (weight = 2).
- Pair programming – pair programming does not relate to testing directly. Nevertheless, it affects the quality of a source code – could be considered as on-the-fly code review (weight = 1).
- Root-cause analysis – it is one of the quality related eXtreme Programming corollary practices. It forces complex analysis for each of the identified defects. Not only the defect should be resolved, but also the software development process should be improved to avoid similar defects in future. The root-cause analysis is not recommended when some of the essential eXtreme Programming practices are not adopted. Therefore, there are agile testing processes that do not employ it (weight = 1).
- Working software (over comprehensive documentation) – one of the Manifesto for Agile Software Development rules that may have strong influence on tests (weight = 1).
- Responding to change (over following the plan) – another rule from the aforementioned manifesto. In the context of a testing process, this rule is mainly considered with respect to the flexibility of test plans (weight = 1).

The above listed criteria are extracted from principles suggested in eXtreme Programming [3] and Agile Manifesto. Each of them is somehow connected with testing and as the Authors believe they make a rich enough subset of all agile principles to offer a good understanding of the project reality with respect to the testing process. The questionnaire was generated in a paper form, nonetheless, it is available on-line: http://purl.org/MarianJureczko/TestingProcess/Survey.

2.3.2. Survey Results

The questionnaire was completed by all present team members and those past members that are still working for the company. The results are dis-

[1] The author argued the usage of weights by stating that the principles of agility have different relevancy from the testing perspective. It is also noteworthy that some of the agile software development methods, e.g. XP [3], identify categories of principles that are based on the relevancy of their adoption.

cussed in detail in the forthcoming subsections and presented in Figure 2. Each of the sub figures shows which respondents recognised given levels of adoption. It is worth mentioning that there is no single 'I do not know' response. Presumably it is a consequence of the fact that each of the respondents actively participates or participated in the investigated project.

Test driven development More than half of the respondents recognised test driven development as partly implemented and the rest of them as fully implemented (Fig. 2a). TDD has been used in the project for a long time, however, not all features are developed in this way. It is always the developer's responsibility to choose the most efficient way of the development and there are no repercussions for not doing TDD. Regardless of the selected development method, the unit tests are obligatory and hence TDD is often a good choice. Nonetheless, there is also a number of features that are closely coupled to moderately documented, external services which do not make a perfect environment for TDD.

Test automation Almost all respondents recognised test automation as fully adopted (Fig. 2b). test automation is very important in the investigated testing process. The automated unit tests are explicitly written in the project's 'Definition of Done' [8]. The automation of functional tests is also obligatory, however, sometimes it is postponed and it is not conducted in the very same sprint as the functionality which has to be tested. Typically, there is one automated test case per a user story, but there are also some very complex stories that are tested by more test cases and some complex test cases that cover more stories. All the automated tests cases are used as regression tests and are executed in a continuous integration system which brings us to the next evaluation criterion.

Continuous integration Each of the respondents acknowledged that the continuous integration principle was fully adopted (Fig. 2c). There is only one development line, and as far as it is possible the development team is avoiding using branches and promotes frequent commits. Moreover, there is a number of jobs defined in the Hudson (http://hudson-ci.org/) system to support automatic compilation, testing and deployment of the developed system.

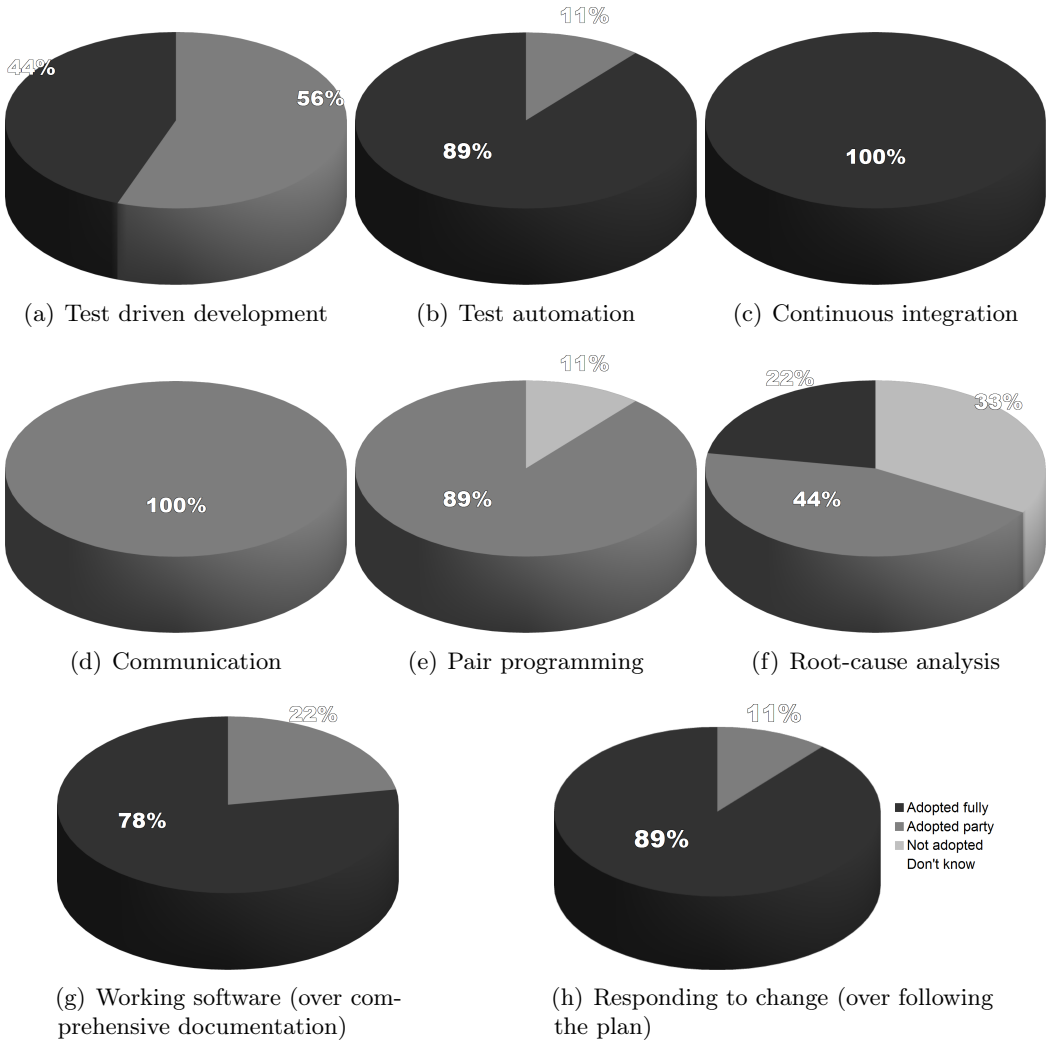
The unit tests and some of the functional tests are executed after each commit. The functional tests are split into two groups. The first of them contains test cases that do not need a great amount of time for execution, which in fact means no interaction with graphical interface – these tests are executed after each commit. The second group of tests contains GUI related tests and for performance reasons it is executed nightly.

The Hudson continuous integration system supports also releases. After a Sprint Review Meeting, i.e. when the sprint ends, a Hudson job is executed and performs the following actions:

- The version number of the developed system is updated.
- A SubVersion 'TAG' is created for the new version of the developed system.
- Release notes regarding newly implemented features are generated.
- A new version of the developed system is deployed to Apache Maven repository and saved on a FTP server.
- The new version is installed automatically in the customer acceptance tests environment.

Communication All respondents recognised the Communication principle as partly adopted (Fig. 2d). The communication was considered at two levels, namely among team members, specifically between testers and developers and between team members and the customer. The communication between team members is acknowledged to be effective. Testers and developers work in the same location. The communication is mostly verbal (but all major issues are reported in an issue tracking system) and supported by the Scrum meetings, e.g. Daily Scrum Standup Meeting. Furthermore, the roles (i.e. testers and developers) are not fixed, thus a team member has an opportunity to do developing as well as testing tasks.

The communication with customer was not assessed so well. The eXtreme Programming 'on site customer' principle [3] has not been installed. Furthermore, customer representatives do not participate in Planning and Review meetings [8]. Communication channels usually go through Product



(a) Test driven development (b) Test automation (c) Continuous integration

(d) Communication (e) Pair programming (f) Root-cause analysis

(g) Working software (over comprehensive documentation) (h) Responding to change (over following the plan)

Figure 2. The level of agility in testing process

Owner, which sometimes makes the communication inefficient.

Pair programming Most of the respondents recognised pair programming as partly adopted (Fig. 2e). The usage of pair programming in the project is limited – more often informal code reviews are used instead, usually in the form of a code walk-through. Nonetheless, presumably each team member experienced this practice since it is extensively used as a knowledge transfer tool. New employees work in pairs with the more experienced ones in order to improve their learning curve.

Root-cause analysis Presumably there is a confusion over what it means to adopt this principle since no clear message comes from the questionnaires (Fig. 2f). Definitely the root-cause analysis is not executed for all identified defects, only a fraction of them is so closely investigated. On the other hand, there are Sprint Retrospective Meetings [8] which address the most important issues and give the team an opportunity to identify remedies.

Working software (over comprehensive documentation) Most of the respondents acknowledged that the working software is valued over comprehensive documentation (Fig. 2g). As a matter of fact the customer is not interested in technical documentation. He is provided only with the user and installation guide. Therefore, the team could decide which technical documents to use and there are no reasons for preparing doc-

uments that are not useful. The Agile Modeling principles [17] are followed and in consequence the number of created documents is limited and always corresponds with one of two purposes, i.e. a model to communicate or a model to understand. The Product and Sprint Backlogs are used, but neither of these documents is delivered with the released product.

Responding to change (over following the plan) Most of the respondents recognised that responding to change is valued over following the plan (Fig. 2h). With regard to test plans the principle is fully adopted since test plans are never fixed and there is always room for an update. It looks slightly different in the case of the set of features (user stories) that are chosen for implementation during a Sprint Planning Meeting. The plan created during the aforementioned meeting shall not be changed according to Schwaber [8]. However, there is a possibility to terminate a Sprint and plan a new one (which has been done several times during the project). Furthermore, sprints are not long (for several months one week sprints were used, currently they have been extended to two weeks), thus waiting for a new sprint with an unplanned change is not painful. It should also also stressed that there is no fixed long term plan. Each new sprint is planned from scratch and hence unexpected changes can be easily handled.

Conclusion The results were evaluated in the way suggested in [5], i.e. value 1 was used for full adoption of an agile principle and 0.5 for partial adoption, then weighted average was calculated (the weights are given in teh previous subsection) and the value of 76.6% was obtained. The value is higher than the one calculated for the project investigated in [5], which could be interpreted as a slightly better adoption of agile testing principles in the described in this study process.

2.4. Objective

The study is focused on a testing process in a medium-size software project with challenging requirements for unscheduled releases and complex infrastructure to deal with. The paper presents how the testing process was tuned with respect to the aforementioned requirements, hence it can be considered as a descriptive or exploratory study [9]. The Authors believe that other practitioners who operate in a similar context will find this work helpful and use it as an example of a well working testing process. The study objective can be defined as follows:

Describe and evaluate an agile testing process with support for unscheduled releases in development of a software system that depends on a number of external services and devices.

The investigated testing process deals with two challenges. Rge first of them comes from business, i.e. there are often unexpected opportunities which must be addressed immediately, i.e. within one or two days depending on a release, otherwise they are missed. In consequence, sometimes it is not possible to wait with the release till the end of a Sprint. The second challenge comes from the project domain. The developed system operates in a complex environment that consists of a number of different network devices and services. Thus, it is a typical case when the configuration of the surroundings requires more time that the test execution itself. This challenge significantly affected the testing process, and therefore the Authors believe that it is a very important part of the project's big picture and must not be ignored in the case study.

2.5. Research Questions and Data Analysis Methods

RQ1: To what extent is the requirement for unscheduled releases satisfied? The unscheduled releases are one of the main drivers of the definition of testing process. Thus, it is critical to study this aspect. It will be evaluated using a quantitative approach. The continuous integration server will be employed as a data source and the results of the builds that execute functional tests (builds with unit tests are a prerequisite) will be used as a measure of the possibility of a release. In order to quantify the data we assumed the a release is possible when all tests are passed. Additionally, we assumed the possibility

of a release with acceptable risk of failure when not more than 10% of tests failed. Such a risk can be accepted only when the release is conducted exclusively for presentation purposes. The 10% threshold is provided to give insight into the variability of the continuous integration outcomes. It also corresponds with the business requirements as 90% of available functionality is usually enough to conduct a presentation. The Authors assumed that the release is possible when before 1 PM the build is successfully finished according to the aforementioned criteria. The time, i.e. 1 PM, was selected as it was the latest possible hour that enables installation in the customer environment or preparation of a presentation (depending on the goal of the unscheduled release) during the same working day.

RQ2: How is the test automation performed? Test automation is the main mean to address unscheduled releases as it is the only way to quickly assess the current software quality and decide if it is acceptable for a release. Furthermore, automation has a crucial role in the testing process and requires significant efforts. Specifically, in a project that operates in a complex environment that creates non-standard requirements for test configuration and in consequence out-of-the-box solutions are not sufficient. Hence, it is vital to describe in detail how the test automation is done. This research question will be addressed using the qualitative approach. All the employed testing frameworks, tools and home-made solutions will be described with respect to their role in the project.

RQ3: How much effort does the test automation require (with respect to different types of activities and tools)? Automated tests play a crucial role in deciding about an unscheduled release and since two different test frameworks were used it could be very interesting to evaluate the process from a business perspective as well. Hence, the study presents data regarding costs of test automation that allow to justify whether it is worth making the investments and which framework should be be chosen to have support for unscheduled releases. The company tracks data about committed efforts using the issue tracking system (Atlassian JIRA). Therefore, it is possible to address the third research question by mining the collected data and extracting information regarding efforts connected with creating new automated functional tests as well as with maintaining the existing ones. The results are presented using descriptive statistics and statistical tests.

3. Results

3.1. To what Extent is the Requirement for Unscheduled Releases Satisfied?

Research regarding availableness for an unscheduled release was conducted over a period of two months. The results are shown in the Figure 3. For 47.4% of the time, the application was ready for the release. A period of release with acceptable risk – 10,5%. The application was not ready for release for 42.1% of the time. The obtained results can be claimed as satisfactory with respect to availableness for an unscheduled release.

Kaner et al. [18] considered test coverage in the context of requirements-based testing, which is very similar to the REST and Selenium functional tests, as they are intended for proving that the program satisfies certain requirements. There is at least one automated test per requirement, which the investigated project is expressed using user stories. Therefore, there is 100% test coverage at the requirement level, i.e. each of the requirements is tested. However, it is an overoptimistic interpretation as not all corner cases in some of the user stories are automated and thus there are places where errors cannot be detected using the REST or Selenium functional tests. The tests execution results may also misleadingly show errors in the case of database malfunction or overload of a machine on which the test environment is located. Adding further tests would decrease the probability of releasing a low quality version of the system by limiting the number of not covered corner cases but it would also increase the probability of blocking a high quality release which can happen as a result of the aforementioned continuous integration

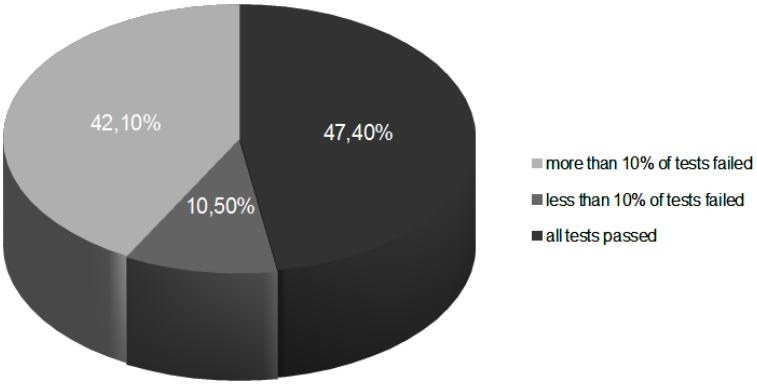


Figure 3. Results obtained from the continuous integration system

system malfunctions. Additional tests would also increase the costs of tests maintenance.

3.2. How is Test Automation Performed?

The results of the execution of automated tests are used as the primary citerion in making decisions regarding unscheduled releases. In order to ensure that tests results correspond with system quality, the team uses a wide variety of tools and frameworks. This includes also several self-developed solutions which contribute to conducting test automation in complex environments.

3.2.1. JUnit Test Framework

JUnit is a test framework for the Java language. On its own it provides the basic functionality for writing tests, which can be further enhanced by other extensions. JUnit tests also serve as an entry point to nearly every type of test in the project development (including the aforementioned self-developed solutions). It is a first class citizen with regard to the way the unit testing and test driven development are conducted. The unit tests are a part of the build, and thus the system cannot be built (and in consequence released) when some of the tests do not pass. There is also a positive side effect, i.e. developers are forced to instantly fix issues detected by unit tests.

EasyMock class extensions. EasyMock provides Mock Objects for interfaces and objects through class extension which is used to replace couplings to external systems or dependencies in unit tests.

3.2.2. Automated Selenium Tests

Automated regression tests with Selenium WebDriver are used to test whether high level functionalities and a graphical user interface work and react properly to input. These tests are combined into a suite and executed nightly within the continuous integration system (a new build of the system under test is deployed at the end of every day on a dedicated server and its database is set to a predefined state). These tests are not executed after each commit, as it is in the case of JUnit and REST tests, since, due to their complexity, the execution takes more than one hour. It is more than the recommended 10 minutes [3] and thus would have decreased the comfort of developers' work.

Tests using Selenium WebDriver are functional, they simulate a user action on the interface and check whether the output matches expectations. Typically a test covers exactly one user story, however, there are exceptions that span multiple test cases and user stories. Additionally, for tests where a connection to an external device is required (and it is not feasible to keep a real device connected to a test machine at all times), the team developed simulators which can be set up to respond like this particular device would (see 3.2.5). The development team strives to have at least one automatic functional test for every user story.

3.2.3. Jersey Test Framework

Tests prepared using Jersey Test Framework are referred to as the REST tests for short. The framework enables functional tests for REST services. It launches the service on an embedded container, sends HTTP requests and captures the responses.

The REST tests are used to test functionality of the system (each user story is covered by at least one Selenium or REST test). The communication between a client and a server and its validity is tested, unfortunately defects from a graphical user interface cannot be detected in such an approach. The REST tests are a part of the build and thus broken tests are early detected and fixed.

3.2.4. DbUnit

DbUnit is a JUnit extension targeted at database-driven tests that, among other things, puts a database into a known state between test runs. DbUnit is used for preparing database for tests. Specifically, it helps in creating the HSQLDB in-memory database that is used during the Selenium and REST tests.

3.2.5. Custom Developed Simulators

In some cases the system under test requires communication with a device or an external service. To solve this problem the team developed:

- SSH Simulator (https://github.com/NetworkedAssets/ssh-simulator/) to simulate communication with a device through the SSH protocol.
- Policy Server Simulator to test communication with a policy server through HTTP.
- WebService simulators to test WebService clients.

The simulators are used in the Selenium and REST tests.

SSH Simulator can be configured to read and respond to commands received through the SSH protocol. The exact behaviour, such as what response should be given to request, is configured in an XML settings file.

The SSH Simulator can be used in two ways. It is possible to launch it as an SSH server or as a temporary service for a single JUnit test. The SSH Simulator tool is configured using XML files. The XML configuration file contains the expected requests and responses that will be generated by the simulator:

SSH Simulator configuration file

```
<test_case ...>
  <login>login</login>
  <password>password</password>
  <device_type>CNR</device_type>
  <request delay_in_ms="3000">
    <request_command>dhcp reload</request_command>
    <response_message>
100 Ok
    </response_message>
    <response_prompt>nrcmd>$</response_prompt>
  </request>
</test_case>
```

The configuration file contains the `<test_case>` entry that represents the sequence of requests and responses mapped using series of `<request>` nodes (on the presented listing there is only one) that define the expected client requests and instruct the simulator how to reply to them. When the XML configuration file is ready, it is enough to use it in a JUnit test case as it is presented in the listing below:

Using SSH-Simulator in JUnit

```
public class MyTest extends
                    SshSimulatorGenericTestCase {
    @Test
    public void sampleTest() {
        initializeNewSshServer(xmlConfigFile,
                               ipAddress, port);
        //do the tests here
    }
}
```

The WebServices simulators are made using J2EE classes from `javax.jws` and `javax.xml.ws` packages. There is a dedicated class that is responsible for launching the WebServices in a separate thread.

Executing a simulated WebService for test purposes

```
public class WebServiceExecutor {
 private static Endpoint endpoint;
 private static ExecutorService executor;
 ...
 public WebServiceExecutor(NaWebService ws) {
   if (endpoint != null &&
                 endpoint.isPublished()) {
      endpoint.stop();
   }
   endpoint = Endpoint.create(ws);
 }
 /** Starts the web service */
 public void publish() {
   if (executor == null) {
     executor =
     Executors.newSingleThreadScheduledExecutor();
   }
   endpoint.setExecutor(executor);
   endpoint.publish(WS_URL);
 }
 /** Closes the web service and verifies
     execution results */
 public void shutdown() throws Throwable {
   endpoint.stop();
   assertTrue(webService.isOk());
   }
}
```

The web service has its own thread, but the 'shutdown' method is called from the JUnit context. Therefore, it is possible to assess what calls the received WebService. It is done by using the 'isOk' method which should be implemented by each of the simulated WebServices:

Example of a WebService implementation

```
@WebService(name = "WS",
                      serviceName = "WSService")
public class MyWs extends NaWebService
   ...
   @WebMethod(operationName = "OP",
                              action = "urn#OP")
   @WebResult(name = "Result",
                 targetNamespace = "http://...")
   @RequestWrapper(...)
   @ResponseWrapper(...)
   public Result op(@WebParam(name = "NAME"...)
                                  String name...)
         throws OperationFault_Exception {
      try {
         assertEquals("PCSCF", name);
      } catch (Throwable t) {
         log.error(t.getMessage(), t);
         reportError(t);
      }
      return = new GenericResponseType.Result();
   }
}
```

The presented example shows how to test the value of a WebService parameter. WebService simulators are usually employed in the REST tests and are useful in testing WebService clients.

3.3. How Much Effort Does the Test Automation Require (with Respect to Different Types of Activities and Tools)?

To answer this question, it is necessary to refer to historical data about the efforts made by team members to create and maintain existing automated tests. For this purpose, data from the issue tracking system used by the company (Atlassian JIRA) were collected.

Then the data about all automated tests have been divided into two categories according to the used tool: Selenium tests and REST tests. This distinction has been made due to the fact that these tools tests different layers of the software. The REST tests are focused on the REST services, while the Selenium test examines the correctness of the operations of GUI which in turn may use the aforementioned services. The difference between these two types of tests is also noticeable in in their maintenance because of time needed to activate them. The Selenium tests are started periodically at a specified time by the continuous integration system (or manually by a developer in their environment), which prolongs the response time to errors in tests. The REST tests are executed in a continuous integration system after each commit and IN A local environment when developers build the application, so tests errors are usually spotted immediately.

Team effort has been measured based on the time which has been logged on specific tasks which regard the creation of automated tests and their subsequent maintenance. Figure 4 presents the measured percentage of time spent on creation and maintenance related to the total time of the REST tests. A relatively low percentage of the REST tests maintenance (18%) is due to the fact that fixing is usually easy to perform. So the overwhelming amount of time is spent on the implementation of new test cases. Figure 5 presents the percentage of time spent on the creation and maintenance of the Selenium tests. Figure 6 shows the average time spent on the implementation and maintenance task of the Selenium and REST test. The average time spent by a developer on Selenium test creation (13.46h) was more than two times longer than the creation of a REST test (5.53h). An even greater difference between average times was observed between the maintenance of the Selenium and a REST tests. The average time of Selenium test maintenance tasks (6.75h) was more than three times longer than the average time logged on REST tests maintenance. It results from the fact that repairing Selenium tests is usually difficult to perform (in most cases there is a need to fix the code on both, the client and the server sides). Figure 6 shows the differences between the effort committed for the REST and Selenium tests. This is largely due to the difference in the complexity of these tests. On the other hand, the Selenium tests (which generally require more effort) detect errors in both, the server and the client side code.

In the case of the creation of both the Selenium and the REST tests, it is possible to present further statistics, see Table 1. The average numbers of hours committed to automation of a test case are reported once more but also the variabilities and results of analysing the differences between the Selenium and the REST tests are given. In the case of the REST tests not only the mean effort but also the variance is noticeably smaller. In order to compare the two types of tests a two-sample t-test was used to verify the following null hypothesis:

The average effort committed to creation of a Selenium test is equal to the average effort of a REST test creation.

versus an alternative hypothesis:

The average effort committed to creation of a Selenium test is larger than in the case of a REST test.

Table 1. Efforts committed to the creation of the Selenium and REST tests statistics

	Selenium tests	REST Tests
Mean	13.46	5.83
Variance	146.89	5.67
F	24	
$P(F \leq f)$ one-tail	0.0004	
t-Stat	3.2	
$P(T \leq t)$ one-tail	0.0014	

Since the alternative hypothesis is asymmetric, the one-tail version of the t-test was used. The hypotheses are tested at the significance level $\alpha = 0.05$. The F-Test was used to test whether the variances of two populations are equal and according to the obtained value of $P(F \leq f)$, i.e. smaller that 0.05, the two-sample assuming unequal variances version of t-test [2] was used. $P(T \leq t)$ value equal to 0.0014 was obtained, thus the null hypothesis can be rejected and the alternative one accepted. In other words, the effort needed to create a REST test is significantly smaller than it is in the case of the Selenium tests.

It is not possible to present analogous statistics for the efforts related to the maintenance of the tests. The maintenance task almost always spans across multiple test cases, hence there are no data regarding individual tests and the average values have already been reported.

The execution of the automated test cases is done in the continuous integration system, hence it does not involve additional efforts. The project compilation and build process, which is frequently executed by developers as a part of their work, includes only unit tests and REST tests and in consequence it is below the 10 minutes suggested by Beck [3]. Therefore,

[2] Satterthwaite's approximate t-test, a method in the Behrens–Welch family.

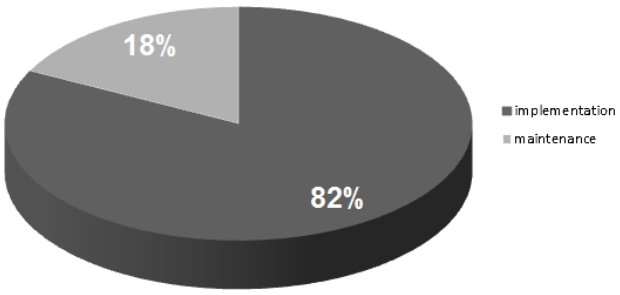


Figure 4. Time spent on the REST tests

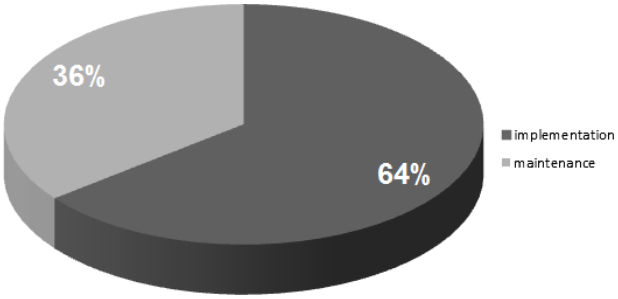


Figure 5. Time spent on the Selenium tests

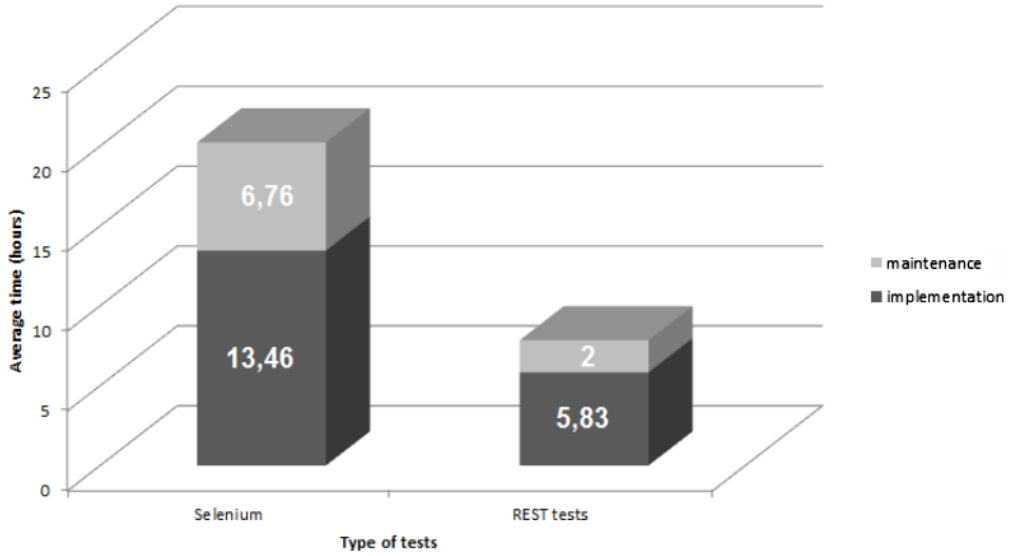


Figure 6. Average time spent on implementation and maintenance of a single Selenium and REST test

the Authors do not consider test execution as a source of additional effort for a development team.

The next logical step in answering this research question leads to the unit tests. Unfortunately, there are no empirical data in this area. As in the case of test driven development, the unit tests are created simultaneously with the code. There were no dedicated tasks regarding unit tests that carry effort related information. Furthermore, it is not possible to decide which part of the tracked effort was committed to the production and which to the test code. However, test driven development considers the preparation of a unit test as an integral part of the development process, hence decisions regarding unit tests should not be driven by cost and effort related criteria in an agile testing process.

There is one more type of tests in the investigated project, i.e. the manual tests. These tests are outside of the scope of RQ3 as they are not appropriate for supporting unscheduled releases, but still could be interesting. Unfortunately, it was not possible to correlate the manual tests efforts per particular requirement and according to the development team there is a significant fraction of untracked efforts. Therefore, the Authors decided to report the subjective interpretations of team members involved in manual testing. The interviewed team members estimated the cost of implementing and maintaining an automated test scenario in the investigated project to be up to 20 times higher than the cost of manual execution. Furthermore, the overall testing effort (it covers both manual and automated tests) was estimated by the members of the development

team to be close to 25% of the overall development effort.

4. Threats to Validity

In this section the most trustworthy results are evaluated: to what extent they are true and not biased toward the Authors' subjective point of view.

4.1. Construct Validity

Most of the employed measures are indirect. The level of agility was assessed using questionnaires, hence there is a risk of misunderstanding the questions or giving biased answers due to unintentional company influence or context. To address the risk a meeting was organized, where all concepts from the questionnaires were explained and all questions that were answered. The possibility of unintentional influence is connected with the fact that all of the questioned people (as well as the Authors) were involved in the project development. Therefore, they have a wide knowledge about the object of study, but simultaneously they cannot have an objective point of view which is necessary to spot the influence. In consequence, it must be stated that the level of agility was assessed only from a subjective perspective. Moreover, the Authors' involvement creates an even greater threat to validity in terms of bias. To mitigate the issue quantitative measures were preferred over qualitative ones in the study design as the latter ones are more vulnerable to influence.

The ability of making an unscheduled release was assessed from the perspective of automated tests and the continuous integration system. The tests results carry information about the quality of the developed system. Nevertheless, using them exclusively is a simplification of the subject. There could be some unmeasurable factors involved (like team members intuition). The test results obtained from the continuous integration system is the most convincing, quantitative measure we were able to come up with.

Test automation efforts were evaluated using data stored in the issues tracking system. There are no serious doubts about the quality of the data, but when it comes to completeness the situation changes. There is a considerable probability that a fraction of efforts was not tracked. There could be small issues that were not reported at all or issues that were missed by the developers, e.g. if they forgot about tracking their efforts. We did not find a way to evaluate which fraction of the collected data suffers from such problems. However, the data filtering was done manually. The developers, who were assigned to the test automation tasks were approached and asked how the collected data correspond with their real efforts.

4.2. Internal Validity

According to Runeson and Höst [9] the internal validity is a concern when casual relations are examined. This study does not provide such a relation explicitly. Nonetheless, it may create an impression that following the principles that are used in the project described here should lead to similar results, which the Authors in fact believe is true. However, there is still a possibility that some relevant factors are missing as they were not identified by the Authors. To mitigate the risk of this threat, the context was described according to guidelines suggested by Petersen and Wohlin [11] and the rest of the study was designed following Runeson and Höst [9] recommendations for a descriptive and to some extend exploratory case study. Specifically, the Authors used the suggested research process, terminology, research instruments and validity analysis.

4.3. External Validity

The context of this case study is described in Section 2.1 and there is no evidence that the same or similar results could be achieved in a another context. Nonetheless, the Authors believe that the environment complexity (e.g. the need for simulators) increases the efforts related to test automation and hence better results may be obtained when there are fewer couplings with external systems.

Specifically, it must be stressed out that the comparison of efforts related to different test frameworks has very limited external validity. Statistical tests were employed, but the investigated data were mined from only one project. Therefore, it is not clear whether the results are true for other projects and it cannot be empirically verified which project specific factors are relevant for the comparison results. A plausible explanation is presented in the 'Discussion and conclusions' section, however, data from additional projects are required to confirm it.

4.4. Reliability

According to Runeson and Höst [9] reliability is concerned with the extent to which the data and the analysis are dependent on specific researchers. The conducted analyses are rather straightforward and several perspectives have been presented to ensure the reliability. Nonetheless, the Authors do not publish raw data, as they contain business critical information, and that can be an obstacle in replicating the study. Additionally, the Authors were involved in project development and thus the observations as well as conclusions may be biased – the issue is discussed in Subsection 4.1.

5. Related Work

The agile methods have been used for a couple of years. Thus, a number of case studies with regard to the testing process have already been conducted. Nonetheless, the Authors are not familiar with any works that analyze the agile testing process with respect to unscheduled releases. On the other hand, the complexity of the developed system is always a factor taken into account, however, it but seldom becomes the object of a study – none of the works reported in this section consider a similar approach to handling system complexity.

Kettunen et al. [2] compared testing processes between software organizations that applied agile practices and employ traditional plan driven-methods. Altogether twelve organizations were investigated and as a result the Authors concluded that agile practices:

- tend to allow more time for testing activates, while the total time for the project remains the same,
- smooth the load of test resources,
- require stakeholders to understand and conform to the practices in agile methods,
- are usually supported by internal customers,
- allow faster reaction time for change.

The findings advocate agile testing but do not correspond with the goals of the process investigated in this study, i.e. support for unscheduled releases and complex infrastructure.

Jureczko [5] investigated the level of agility in a testing process in a large scale financial software project. The studied project is of different size and comes from another domain, nevertheless, the suggested criteria for agility evaluation have been used to assess the testing process described in this study. Therefore, a comparison is possible. The process from [5] is significantly outperformed in the scope of continuous integration, pair programming, root-cause analysis and working software (over comprehensive documentation), however, it is underperformed in the field of communication. The overall assessment is higher in the process studied in this work. Let us elaborate the development and testing process of this work. The project was planned for 150 working years, but later the duration time was lengthened. More than 150 high skilled developers, testers and managers were involved. The project was divided into five sub-projects and the paper is focused on only one of them. The sub-project was developed by a group of about 40 people. The project is a custom-build solution that supports more than 1500 initially identified case scenarios. The system is based on a well defined technical framework, called Quasar. The development team frequently delivers new releases with new functionalities. There are two major releases per year: in spring and autumn. They are used to deliver large changes and bulks of possible bugs. The two major ones are supplemented by hotfix releases that target the most important bugfixes only. The employed testing process forces

practices borrowed from the V-Model. Testers work on test concepts once the specification of a requirement is ready. Subsequently developers write a source code and perform manual tests when testers write new automated tests. Each of the tests is immediately added to the regression test set that is executed daily. Subsystem tests are performed after the integration. They are usually manual and focused on new features.

The role of test automation in a testing process was empirically evaluated by Karhu et al. [7] in five software organizations among which he was developing a complex system with a number of interfaces to customer-specific external systems and hence creates challenges in test automation. The Authors identified a number of interesting consequences of automation. Quality improvement through better test coverage and increase in the number of executed test cases were noted among benefits, whereas costs of implementation, maintenance and training were pointed out as the main disadvantages. Moreover, according to Berlino [1] great emphasis is put on this issue and the methods of extending the degree of attainable automation are in the focus of testing research. Unfortunately, there is a dichotomy with regard to the industrial reality, i.e. many companies, which claim that they have adopted XP, practice the automation in a limited scope [19]. Considerable research has been conducted on the test automation [16, 20–23] and in general this practice is strongly recommended. Among the aforementioned works especially [21] is noteworthy as it is conducted it the context of the Scrum method. Another commonly investigated agile testing practice is test driven development. There is evidence for its usefulness in the industrial environment [24, 25], reports of controlled experiments are also available [26, 27].

Winter [28] investigated the challenges regarding testing in a complex environment, however, the complexity came from evaluation usability for a variety of end users. One of the areas of interest was the agility of a testing process and the balance between the formal and informal approaches. The agility was considered in the context of the Agile Manifesto, and hence there was limited overlap with the criteria employed in our study (i.e. only Working software (over comprehensive documentation) and Responding to change (over following the plan) are considered in both studies).

Talby et al. [6] described installation of agile software testing practices in a real large-scale project in a traditional environment. The authors pointed out that in the investigated case study the agile methods dramatically improved quality and productivity. The testing process was analyzed and described in four key areas: test design and activity execution, working with professional testers, planning, and defect management. The investigated project is a sub-project of a larger one. The larger one was developed by 60 skilled developers and testers. Thus, there is a considerable probability that the sub-project is similar in size to the project described in this study. More details about the software project investigated in [6] can be found in [29].

A lesson learned from a transformation from a plan-driven to agile testing process is documented in [14]. Extreme programming and Scrum were adopted, which makes the process similar to the one described here. Hence, the challenges described by Puleio [14] may arise when trying to adopt adopt the principles advocated in this paper in a traditional (i.e. not agile) environment.

6. Discussion and Conclusions

This paper contributes to the body of knowledge in two ways. The Authors provide a detailed case study of an agile testing process, which can be used in further research and in combination with other studies to help make general conclusions. The documentation of the testing process in this project could also be beneficial to practitioners interested in installing an agile testing process in a similar environment. Especially, a number of ready to use testing tools are recommended (e.g. the simulators).

A survey was conducted in order to assess the level of agility and the results showed a high level of adoption, i.e. 76.6%. Furthermore, each of the assessment criteria was compared against the

project reality in a qualitative approach which gives an insight into the way the agile principles work. A detailed analysis indicated some areas of possible improvement, e.g. communication with customer and pair programming.

The project preparation for the 'on short notice' release was analysed and assessed according to test execution results in the continuous integration system. The results gave a value of about 47% of time in which the project was ready to be released and 10% of time it could have been released with the acceptable risk. Hence, the Authors can conclude that the requirement for unscheduled releases is supported to a considerable extent.

Comparative evaluation of the cost of the REST and Selenium tests was conducted. It measured how much time is necessary in both test frameworks for the implementation of new test and maintenance of the existing ones. A significant disadvantage was found in the case of the Selenium tests, which we believe is a result of using Google Web Toolkit for the graphical interface – this framework uses dynamic identifiers and generates complex (at least from the Selenium point of view) web pages. This extra cost is counterbalanced with an ability to detect bugs in GUI which is covered only by the Selenium test. Nonetheless, the big difference in costs encourages reconsideration of the testing approach.

A brief description of all tools that were used to implement automated tests is provided. The main contribution in this area regards the suggestion for mitigating environment complexity with simulators. There is a detailed description of an open-source project called SSH-simulator which was co-developed by the Authors, in fact, it is officially presented in this paper for the first time. Furthermore, the Authors also suggested a straightforward solution for simulating WebServices in a test environment. The paper contains listings that show how to mock a WebService in the context of the JUnit tests. The Authors believe that the detailed descriptions of those simulators will help other practitioners who face similar challenges during test automation since the presented solutions are ready to use (the necessary listings and references to external sources are given in Sec. 3.2.5).

The overall results are satisfactory with regard to the project goals. Therefore, we would like to recommend following the same rules (in similar projects), i.e. adopt the principles of agility, specifically assure high quality and coverage of automated tests and employ a continuous integration system for automated builds.

References

[1] A. Bertolino, "Software testing research: Achievements, challenges, dreams," in *Future of Software Engineering, FOSE '07*. IEEE, 2007, pp. 85–103.

[2] V. Kettunen, J. Kasurinen, O. Taipale, and K. Smolander, "A study on agility and testing processes in software organizations," in *Proceedings of the 19th international symposium on Software testing and analysis*. ACM, 2010, pp. 231–240.

[3] K. Beck and C. Andres, *Extreme programming explained: embrace change*. Addison–Wesley Professional, 2004.

[4] L. Koskela, *Test driven: practical TDD and acceptance TDD for Java developers*. Manning Publications Co., 2007.

[5] M. Jureczko, "The level of agility in the testing process in a large scale financial software project," in *Software engineering techniques in progress*, T. Hruška, L. Madeyski, and M. Ochodek, Eds. Oficyna Wydawnicza Politechniki Wrocławskiej, 2008, pp. 139–152.

[6] D. Talby, A. Keren, O. Hazzan, and Y. Dubinsky, "Agile software testing in a large-scale project," *IEEE Software*, Vol. 23, No. 4, 2006, pp. 30–37.

[7] K. Karhu, T. Repo, O. Taipale, and K. Smolander, "Empirical observations on software testing automation," in *International Conference on Software Testing Verification and Validation, ICST '09*. IEEE, 2009, pp. 201–209.

[8] K. Schwaber, *Agile project management with Scrum*. Microsoft Press, 2004.

[9] 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.

[10] R. Mall, *Fundamentals of software engineering*. PHI Learning Pvt. Ltd., 2009.

[11] K. Petersen and C. Wohlin, "Context in industrial software engineering research," in *Proceed-*

ings of the 3rd International Symposium on Empirical Software Engineering and Measurement. IEEE Computer Society, 2009, pp. 401–404.

[12] S. Harichandan, N. Panda, and A.A. Acharya, "Scrum testing with backlog management in agile development environment," *International Journal of Computer Science and Engineering*, Vol. 2, No. 3, 2014.

[13] K.K. Jogu and K.N. Reddy, "Moving towards agile testing strategies," *CVR Journal of Science & Technology*, Vol. 5, 2013.

[14] M. Puleio, "How not to do agile testing," in *Agile Conference*. IEEE, 2006, pp. 305–314.

[15] K. Beck, *Test driven development: By example*. Addison–Wesley Professional, 2003.

[16] M. Jureczko and M. Mlynarski, "Automated acceptance testing tools for web applications using test-driven development," *Electrical Review*, Vol. 86, No. 09, 2010, pp. 198–202.

[17] S. Ambler, *Agile modeling: effective practices for extreme programming and the unified process*. Wiley, 2002.

[18] C. Kaner, J. Bach, and B. Pettichord, *Lessons learned in software testing*. John Wiley & Sons, 2008.

[19] D. Martin, J. Rooksby, M. Rouncefield, and I. Sommerville, "'Good' organisational reasons for 'bad' software testing: An ethnographic study of testing in a small software company," in *29th International Conference on Software Engineering, ICSE 2007*. IEEE, 2007, pp. 602–611.

[20] M. Catelani, L. Ciani, V.L. Scarano, and A. Bacioccola, "Software automated testing: A solution to maximize the test plan coverage and to increase software reliability and quality in use," *Computer Standards & Interfaces*, Vol. 33, No. 2, 2011, pp. 152–158.

[21] R. Löffler, B. Güldali, and S. Geisen, "Towards model-based acceptance testing for Scrum," *Softwaretechnik-Trends*, Vol. 30, No. 3, 2010.

[22] X. Wang and P. Xu, "Build an auto testing framework based on selenium and fitnesse," in *International Conference on Information Technology and Computer Science, ITCS 2009*, Vol. 2. IEEE, 2009, pp. 436–439.

[23] T. Xie, "Improving effectiveness of automated software testing in the absence of specifications," in *22nd IEEE International Conf. on Software Maintenance, ICSM '06*. IEEE, 2006, pp. 355–359.

[24] N. Nagappan, E.M. Maximilien, T. Bhat, and L. Williams, "Realizing quality improvement through test driven development: results and experiences of four industrial teams," *Empirical Software Engineering*, Vol. 13, No. 3, 2008, pp. 289–302.

[25] A.P. Ress, R. de Oliveira Moraes, and M.S. Salerno, "Test-driven development as an innovation value chain," *Journal of technology management & innovation*, Vol. 8, 2013, p. 10.

[26] L. Madeyski, "The impact of pair programming and test-driven development on package dependencies in object-oriented design—an experiment," in *Product-Focused Software Process Improvement*. Springer, 2006, pp. 278–289.

[27] L. Madeyski, "The impact of test-first programming on branch coverage and mutation score indicator of unit tests: An experiment," *Information and Software Technology*, Vol. 52, No. 2, 2010, pp. 169–184.

[28] J. Winter, K. Rönkkö, M. Ahlberg, and J. Hotchkiss, "Meeting organisational needs and quality assurance through balancing agile and formal usability testing results," in *Software Engineering Techniques*. Springer, 2011, pp. 275–289.

[29] Y. Dubinsky, D. Talby, O. Hazzan, and A. Keren, "Agile metrics at the Israeli air force," in *Agile Conference, Proceedings*. IEEE, 2005, pp. 12–19.

e-Informatica Software Engineering Journal, Volume 10, Issue 1, 2016, pages: 89–123, DOI 10.5277/e-Inf160105

Software Startups – A Research Agenda

Michael Unterkalmsteiner[a], Pekka Abrahamsson[b], XiaoFeng Wang[c], Anh Nguyen-Duc[a], Syed Shah[d], Sohaib Shahid Bajwa[c], Guido H. Baltes[e], Kieran Conboy[f], Eoin Cullina[f], Denis Dennehy[f], Henry Edison[c], Carlos Fernandez-Sanchez[g], Juan Garbajosa[g], Tony Gorschek[a], Eriks Klotins[a], Laura Hokkanen[h], Fabio Kon[i], Ilaria Lunesu[j], Michele Marchesi[j], Lorraine Morgan[k], Markku Oivo[l], Christoph Selig[k], Pertti Seppänen[l], Roger Sweetman[f], Pasi Tyrväinen[m], Christina Ungerer[k], Agustin Yagüe[g]

[a] *Blekinge Institute of Technology, Sweden,* [b] *Norwegian University of Science and Technology, Norway,* [c] *Free University of Bolzano-Bozen, Italy,* [d] *SICS, Sweden,* [e] *Lake Constance University, Germany,* [f] *National University of Ireland Galway, Ireland,* [g] *Technical University of Madrid, Spain,* [h] *Tampere University of Technology, Finland,* [i] *University of São Paulo, Brazil,* [j] *University of Cagliari, Italy,* [k] *National University of Ireland Maynooth, Ireland,* [l] *University of Oulu, Finland,* [m] *Hochschule Konstanz, Germany,* [n] *University of Jyväskylä, Finland*

mun@bth.se, pekkaa@ntnu.no, xiaofeng.wang@unibz.it, anhn@idi.ntnu.no, shah@sics.se, bajwa@inf.unibz.it, guido.baltes@cetim.org, kieran.conboy@nuigalway.ie, eoin.cullina@outlook.com, denis.dennehy@nuigalway.ie, henry.edison@inf.unibz.it, carlos.fernandez@upm.es, jgs@eui.upm.es, tgo@bth.se, ekx@bth.se, laura.hokkanen@tut.fi, fabio.kon@ime.usp.br, ilaria.lunesu@diee.unica.it, michele@diee.unica.it, lorraine.morgan@nuim.ie, markku.oivo@oulu.fi, cselig@htwg-konstanz.de, pertti.seppanen@oulu.fi, roger.sweetman@nuigalway.ie, pasi.tyrvainen@jyu.fi, christina.ungerer@htwg-konstanz.de, ayague@etsisi.upm.es

Abstract

Software startup companies develop innovative, software-intensive products within limited time frames and with few resources, searching for sustainable and scalable business models. Software startups are quite distinct from traditional mature software companies, but also from micro-, small-, and medium-sized enterprises, introducing new challenges relevant for software engineering research. This paper's research agenda focuses on software engineering in startups, identifying, in particular, 70+ research questions in the areas of supporting startup engineering activities, startup evolution models and patterns, ecosystems and innovation hubs, human aspects in software startups, applying startup concepts in non-startup environments, and methodologies and theories for startup research. We connect and motivate this research agenda with past studies in software startup research, while pointing out possible future directions. While all authors of this research agenda have their main background in Software Engineering or Computer Science, their interest in software startups broadens the perspective to the challenges, but also to the opportunities that emerge from multi-disciplinary research. Our audience is therefore primarily software engineering researchers, even though we aim at stimulating collaborations and research that crosses disciplinary boundaries. We believe that with this research agenda we cover a wide spectrum of the software startup industry current needs.

Keywords: software startup, research agenda, software-intensive systems

1. Introduction

Researchers are naturally drawn to complex phenomena that challenge their understanding of the world. Software startup companies are an intriguing phenomenon, because they develop innovative software-intensive[1] products under time constraints and with a lack of resources [2], and constantly search for sustainable and scalable business models. Over the past few years, software startups have garnered increased research interest in the Software Engineering (SE) community.

While one could argue that software startups represent an exceptional case of how software products are developed and brought to the market, several factors suggest a broader impact. From an economical perspective, startups contribute considerably to overall wealth and progress by creating jobs and innovation [3]. Digital software startups[2] are responsible for an astonishing variety of services and products [5]. In the farming sector, venture investment in so-called "AgTech" startups reached $2.06 billion in just the first half of 2015; this figure neared the $2.36 billion raised during the whole of 2014 [6]. From an innovation perspective, startups often pave the way for the introduction of even more new and disruptive innovations [7]. Kickstarter is changing the retail and finance industries, Spotify is offering a new way to listen to and purchase music, and Airbnb is reinventing the hospitality industry [8]. From an engineering perspective, startups must inventively apply existing knowledge in order to open up unexpected avenues for improvement [9]; e.g., they must provide education for full stack engineers, develop techniques for continuous lightweight requirements engineering, or develop strategies to control technical debt.

Despite these promising conditions, software startups face challenges to survival, even in contexts where they play a key role in developing new technology and markets, such as cloud computing [10]. These challenges may arise because, while developing a product can be easy, selling it can be quite difficult [11]. Software startups face other challenges, such as developing cutting-edge products, acquiring paying customers, and building entrepreneurial teams [12]. Such diverse factors underscore the need to conduct research on software startups, which will benefit both scholarly communities and startup leaders.

This paper's research agenda is driven by past and current work on software startups. We outline the various research tracks to provide a snapshot of ongoing work and to preview future research, creating a platform for identifying collaborations with both research and startup environments and ecosystems. This effort is not a one-way path. We have therefore founded a research network, the Software Startup Research Network (SSRN)[3], which enables interactions and collaborations among researchers and interested startups. SSRN envisions to: (1) spread novel research findings in the context of software startups; and (2) inform entrepreneurs with necessary knowledge, tools and methods that minimize threats and maximize opportunities for success. As part of the network initiatives, an International Workshop of Software Startups was established in 2015. The first edition of the workshop was held in Bolzano[4] (Italy) in 2015, and the second took place in Trondheim[5] (Norway) in 2016. This paper provides a research agenda based on the activities carried out by the researchers in the network.

The rest of the paper is organized as follows. After we clarify the meaning of *software startup* and what we know about software startups from

[1] ISO 42010:2011 [1] defines software-intensive systems as "any system where software contributes essential influences to the design, construction, deployment, and evolution of the system as a whole" to encompass "individual applications, systems in the traditional sense, subsystems, systems of systems, product lines, product families, whole enterprises, and other aggregations of interest".

[2] In our article, digital startups refer specifically to startups in which the business value of the solution is created by means of software [4].

[3] https://softwarestartups.org

[4] http://ssu2015.inf.unibz.it/

[5] https://iwssublog.wordpress.com/

prior research in the Background section, Section 3 introduces the research topics on software startups, organized under six main tracks that we have either investigated or envision investigating in the future. Wherever possible, each topic is illustrated and motivated by previous studies. Section 4 highlights the implications of these main tracks for future research. The paper concludes with Section 5, which points out future actions that can establish and consolidate software startups as a research area.

2. Background

2.1. What is a Software Startup?

To understand software startups, we must first clarify what a startup is. According to Ries [13], a startup is a human institution designed to create a new product/service under conditions of extreme uncertainty. Similarly, Blank [14] describes a startup as a temporary organization that creates high-tech innovative products and has no prior operating history. These definitions distinguish startups from established organizations that have more resources and already command a mature market. In addition, Blank [14,15] defines a startup as a temporary organization that seeks a scalable, repeatable, and profitable business model, and therefore aims to grow. Blank's definition highlights the difference between a startup and a small business, which does not necessarily intend to grow, and consequently lacks a scalable business model.

Even though sharing common characteristics with other types of startups, such as resource scarcity and a lack of operational history, software startups are often caught up in the wave of technological change frequently happening in software industry, such as new computing and network technologies, and an increasing variety of computing devices. They also need to use cutting-edge tools and techniques to develop innovative software products and services [16]. All these make software startups challenging endeavours and meanwhile fascinating research phenomena for software engineering researchers and those from related disciplines.

In 1994, Carmel first introduced the term *software startup*, or, to be more precise, *software package startup*, in SE literature [17]. Carmel [17] argued that software was increasingly becoming a fully realized product. Since then, other researchers have offered their own definitions of *software startup*. Sutton [16] considers software startups as organizations that are challenged by limited resources, immaturity, multiple influences, vibrant technologies, and turbulent markets. Hilmola et al. [18] claim that most software startups are product-oriented and develop cutting edge software products. Coleman and Connor [19] describe software startups as unique companies that develop software through various processes and without a prescriptive methodology.

Currently, there is no consensus on the definition of *software startup*, even though many share an understanding that software startups deal with uncertain conditions, grow quickly, develop innovative products, and aim for scalability. Different definitions emphasize distinct aspects, and consequently may have varying implications for how studies that adopt them should be designed, e.g., who qualifies as study subjects, or which factor is worth exploring. For this reason, despite the lack of a single agreed-upon definition of *software startup*, it is important and recommended that researchers provide an explicit characterization of the software startups they study in their work. The research track in Section 3.1.1 is dedicated to develop a software startup context model that would allow for such a characterization.

2.2. What are the Major Challenges of Software Startups?

Software startups are challenging endeavours, due to their nature as newly created companies operating in uncertain markets and working with cutting edge technology. Giardino et al. [20] highlight software startups' main challenges as: their lack of resources, that they are highly reactive, that they are by definition a new company, that they are comprised of small teams with little experience, their reliance on a single product

and innovation, and their conditions of uncertainty, rapid evolution, time pressure, third-party dependency, high risk, and dependency (they are not self-sustained). Further, Giardino et al. [12] apply the MacMillan et al. [21] framework in the software startup context, categorizing the key challenges faced by early stage software startups into four holistic dimensions: product, finance, market, and team. The findings of Giardino et al. [12] reveal that thriving in technological uncertainty and acquiring the first paying customer are the top key challenges faced by many startups. In another study, Giardino et al. [22] discover that inconsistency between managerial strategies and execution could lead to startup failure.

Although research exists on the challenges software startups face, there is no study dedicated to their success factors. Block and Macmillan's [23] study highlights the success factors for any new business, including generating ideas to complete product testing, completing a prototype, and consistently re-designing or making amendments. Researchers have yet to explore these general factors' applicability to the specific software startup context.

2.3. What do We Know about Software Engineering in Software Startups?

Software development comprises a software startup's core activity. However, some initial research studies report a lack of software engineering activities in software startups. A systematic mapping study conducted by Paternoster et al. [2] allows us to start understanding how software startups perform software development. The study reveals that software requirements are often market driven and are not very well documented. Software development practices are only partially adopted; instead, pair programming and code refactoring sessions supported by ad-hoc code metrics are common practices. Testing is sometimes outsourced or conducted through customer acceptance and focus groups, and team members are empowered and encouraged to adapt to several roles. Similarly, Giardino et al. [20] highlight the most common development practices that have been used in software startup companies, such as: using well-known frameworks to quickly change the product according to market needs, evolutionary prototyping and experimenting via existing components, ongoing customer acceptance through early adopters' focus groups, continuous value delivery, focusing on core functionalities that engage paying customers, empowerment of teams to influence final outcomes, employing metrics to quickly learn from consumers' feedback and demand, and engaging easy-to-implement tools to facilitate product development.

Although a few studies provide snapshots of software engineering practices in software startups [9, 24], the state of the art presented in literature is not enough to base an understanding of how software engineering practices could help software startups. Researchers must build a more comprehensive, empirical knowledge base in order to support forthcoming software startups. The research agenda presented in this paper intends to inspire and facilitate researchers interested in software startup related topics to start building such knowledge base.

3. Research Agenda

The Software Startup Research Agenda, initialized in June 2015, was developed by a network of researchers interested in studying the startup phenomenon from different angles and perspectives. This variety of research interests not only opens up new avenues for collaboration, but also sheds light on the complexity of the studied phenomenon. Initially, ten researchers created a mind map of different research areas, aiming to provide an overview of software startup research areas and how they connect to each other. Over a period of six months, more researchers joined the network, added their research tracks, and continuously expanded the map. A working session with twenty researchers at the 1st workshop on software startup research in December 2015 was devoted at discussing the identified areas and finding potential interest overlaps among the participants. After this meeting, the authors of this paper prepared eighteen research track

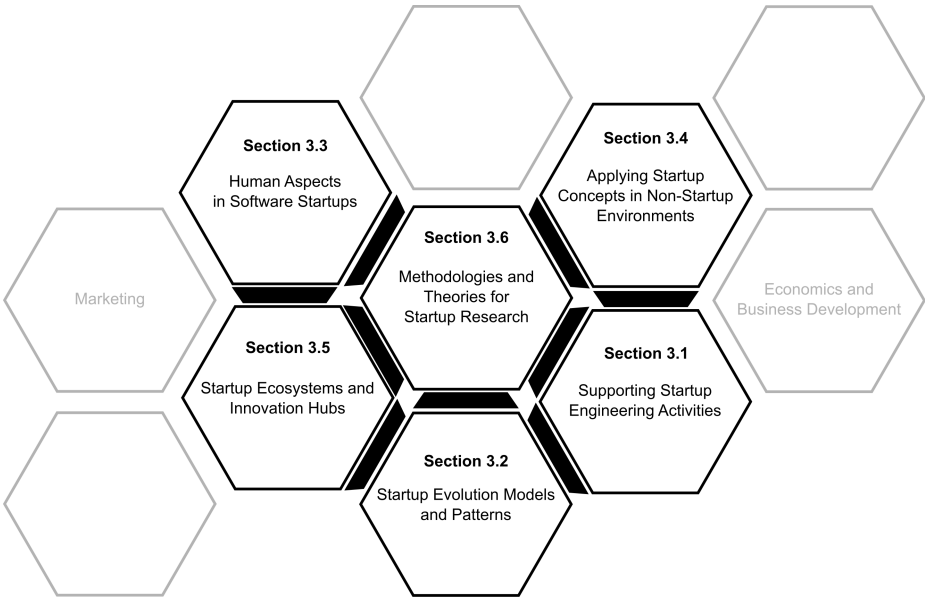


Figure 1. Overview of the Software Startup Research Agenda

descriptions according to the following pattern: background of the area, motivation and relevance for software engineering in startups, research questions, potential impact of answering these research questions on practice and research, potential research methodologies that can be employed to answer the proposed research questions, and related past or ongoing work. Most of the authors interacted in the past or are currently active as advisory board members, mentors, founders or team members of software startups.

The leading authors of this paper grouped the eighteen research tracks into six major clusters, based on the thematic similarities and differences of the tracks. While this grouping is one of the several possible ways to create the clusters, it served the purpose to ease the presentation and discussion of the research agenda, shown in Figure 1. Supporting Startup Engineering Activities (Section 3.1) encompasses research foci that address specific software engineering challenges encountered by startup companies. Startup Evolution Models and Patterns (Section 3.2) focuses on the progression of startups over time, trying to understand the underlying mechanics that drive a company towards success or failure. Human Aspects in Software Startups (Section 3.3) covers research tracks that investigate factors related to the actors involved in startups. The research on Applying Startup Concepts in Non-Startup Environments (Section 3.4) seeks to strengthen innovation by extracting successful software startup practices and integrating them in traditional environments. Startup Ecosystems and Innovation Hubs (Section 3.5), on the other hand, investigates whether and how a thriving environment for software startups can be designed. Finally, all of these areas are connected by research tracks that develop methodologies and theories for software startup research (Section 3.6).

Figure 1's illustration of the research agenda includes reference to research areas outside this paper's current scope. Marketing and Business and Economic Development are directions that are likely relevant for the performance of software startups. These and other areas may be added to the research agenda in later editions when more evidence exists regarding whether and how they interact with software startup engineering, i.e. the "use of scientific, engineering, managerial and systematic approaches with the aim of successfully developing software systems in startup companies" [9].

3.1. Supporting Startup Engineering Activities

The research tracks in this cluster share the theme of studying, identifying, transferring, and evaluating processes, methods, framework, mod-

els, and tools aimed at supporting software startup engineering activities.

3.1.1. The Context of Software Intensive Product Engineering in Startups

Rapid development technologies have enabled small companies to quickly build and launch softwareintensive products with few resources. Many of these attempts fail due to market conditions, team breakup, depletion of resources, or a bad product idea. However, the role of software engineering practices in startups and their impact on product success has not yet been explored in depth. Inadequacies in applying engineering practices could be a significant contributing factor to startup failure.

Studies show that startups use ad-hoc engineering practices or attempt to adopt practices from agile approaches [25, 26]. However, such practices often focus on issues present in larger companies and neglect startup-specific challenges. For example, Yau and Murphy [25] report that test-driven development and pair programming provide increased software quality at an expense of cost and time. Also keeping to a strict backlog may hinder innovation. Since neglecting engineering challenges can lead to sub-optimal product quality and generate waste, engineering practices specific to the startup context are needed. The overarching questions in this research track are:

- RQ1: To what degree is the actual engineering a critical success factor for startups?
- RQ2: How can the startup context be defined such that informed decisions on engineering choices can be made?
- RQ3: What engineering practices, processes and methods/models are used today, and do they work in a startup context?

An answer to RQ1 could help practitioners to decide on what activities to focus on and prioritize allocation of resources. Several studies, e.g Paternoster et al. [2], Giardino et al. [12] and Sutton [16], emphasize the differences between established companies and startups, noting that startups are defined by limited resources and dynamic technologies. However, these characterizations are not granular enough to support a comparison of engineering contexts in different companies, making the transfer of practices from company to company difficult [27]. Thus, understanding the engineering context of startups (RQ2) is an important milestone in developing startup context specific engineering practices (RQ3). While there exists work that provides systematic context classifications for the field of software engineering in general [27–31], these models are not validated and adapted for use within startups. The work in this research track aims to develop such a software startup context model by analysing data from startup experience reports [24]. Provided that engineering contexts among startups and established companies can be compared at a fine level of detail, the context model can be used to identify candidate practices. Moreover, researchers can develop decision support by mapping specific challenges with useful practices, thereby validating the model and helping practitioners select a set engineering practices for their specific context and set of challenges.

3.1.2. Technical Debt Management

The software market changes rapidly. As discussed by Feng et al. [32], in fast changing environments, the product management focus evolves from the more traditional cost or quality orientation to a time orientation. New product development speed is increasingly important for organizations, and a commonly shared belief is that time-to-market of new products can build a competitive advantage [32]. In the software startup context, it may be vital to be the first to market in order to obtain customers. Since software startups also lack resources, quality assurance is often largely absent [2]. However, long-term problems will only be relevant if the product obtains customers in the short term [33]. This short-term vision may produce software code that is low-quality and difficult to change, compelling the company to invest all of its efforts into keeping the system running, rather than increasing its value by adding new capabilities [33]. Scaling-up the system may become an obstacle, which will prevent the company from gaining new customers. Finding a viable trade-off between

time-to-market demands and evolution needs is thus vital for software startups.

One promising approach to performing such a trade-off is technical debt management. Technical debt management consists of identifying the sources of extra costs in software maintenance and analysing when it is profitable to invest effort into improving a software system [33]. Hence, technical debt management could assist startups in making decisions on when and what to focus effort on in product development. Technical debt management entails identifying the technical debt sources, the impact estimation of the problems detected, and the decision process on whether it is profitable to invest effort in solving the detected sources of technical debt [34, 35]. Only those sources of technical debt that provide return on investment should be resolved. More importantly, technical debt should be managed during project development [36] in order to control the internal quality of the developed software. Several research questions need to be answered to successfully manage technical debt in this way:

- RQ1: What kind of evolution problems are relevant in the software startup context? How can we identify them?
- RQ2: How can we prioritize the possible improvements/changes in the context of software startups?
- RQ3. What factors beyond time-to-market and resource availability must be considered in trade-offs?
- RQ4: How can we make decisions about when to implement the improvements/changes within the software startup roadmap?
- RQ5: How can we provide agility to technical debt management, necessary in an environment plenty of uncertainty and changes?

Answering these questions will impact on both practitioners and researchers focused on software startups. Practitioners will be able to make better decisions considering the characteristics of the current software product implementation. The current implementation could make it impossible to reach a deadline (time to market), because of the complexity of the changes to perform to implement a new feature, assuming a given amount (and qualifications) of effort to be deployed. Furthermore, it will be also possible to decide between two alternative implementations, with different costs, but also with different potential for the future, assuming that the "future" has been previously outlined. For researchers, answering these questions could help clarify the role of design decisions in software development in the context of a software product roadmap, similarly to what happens in other engineering disciplines.

Technical debt is context dependent since quality tradeoffs are context dependent [37]. While technical debt is as important to software startups as it is to mature companies, the kind of decisions to take and the consequences of making the wrong decisions are not the same, justifying research on technical debt specifically in software startups.

In general, there is a lack of specific studies on technical debt management in software startups, and current literature reviews on technical debt management do not address this topic [34, 35]. Moreover, there are several specific challenges to managing technical debt that are of special relevance for software startups. For one, very few studies address how to prioritize improvements to solve technical debt problems, especially for commercial software development [35]. In addition, technical debt management literature often refers to time-to-market, but very few studies actually address it [34], perhaps because it is a topic that straddles engineering and economics.

3.1.3. Software Product Innovation Assessment

Startup companies strive to create innovative products. For firms in general, and software startups in particular, it is critical to know as soon as possible if a product aligns with the market, or whether they can increase their chances to lead the market and recruit the highest possible number of customers [38].

The need to invest in infrastructures to measure the impact of innovation in software was highlighted by OECD [39], and more recently by Edison et al. [40]. These measures will enable companies to assess the impact of innovation factors and achieve the expected business

goals, as well as to improve the understanding of success yield high returns on investments in the innovation process [39]. Product innovation assessment is thus very relevant for product developers, and especially for startups, which are more sensitive to market reactions. Product innovation assessment is complex, particularly for software products [41].

Product innovation assessment is reported in literature as the combination of a number of multi-dimensional factors impacting the success or failure of a software product [42]. Factor's measures intend to engage people in the innovation process to think more deeply about factors affecting product innovation. Factors such as time-to-market, perceived value, technology route, incremental product, product liability, risk distribution, competitive environment, life cycle of product, or strength of market could be grouped into dimensions like market, organization, environment, or any other terms of impact on the market and business drivers [43]. These factors can act as innovation enablers or blockers [44].

Since these factors are not always independent, it is critical to identify the existing dependencies and gain a better understanding of each factor's impact. It would be necessary to relate these factors to characteristics specific to software products, such as, but not limited to, software quality attributes proposed by ISO/IEC [45].

There is a lack of specific literature on *software* product innovation assessment; most of the past research refers to products in general, and not specifically to software products [40,46], leading to the following research questions:

- RQ1: What should be the components of a software product innovation assessment/estimation model?
- RQ2: What factors can help measure innovation from a software product and a market perspective?
- RQ3: To what extent are factors that can help measure innovation dependent on the software product and the market perspective?
- RQ4: What is the relation between software product innovation factors and quality factors?
- RQ5: What kind of tools for software product innovation estimation could support software startups in decision making?

While innovation has been widely studied from the process perspective, the product perspective, by nature, has been addressed mainly from the viewpoint of specific products and industries. However, software products are different compared to other kinds of products [47] and innovations in the software industry happen fast. Hence, answers to RQ1-RQ4 would provide a fundamental understanding on software product innovation assessment and be beneficial for both researchers and practitioners. Software startups need to be fast and spend resources in an efficient way. Therefore, to be able to estimate existing products or design new products, considering those characteristics that experience shows that are relevant from an innovation point of view, can be essential for software startups to develop successful products (RQ5).

3.1.4. Empirical Prototype Engineering

Startups often start with a prototype, which serves as a form to validate either a new technology or knowledge about targeted customers [2]. Traditionally, prototyping implies a quick and economic approach to determining final products [48–50]. Defined as a concrete representation of part or all of an interactive system, prototypes has been intensively researched and used in Software Engineering, with well-developed taxonomies, such as horizontal and vertical, low-fidelity and high-fidelity prototypes [50]. The strategy of developing a prototype can greatly vary due to a great variety of prototype types, their development efforts and value they can produce.

While much about prototyping techniques can be learnt from the SE body of knowledge, the discussion about prototyping in the context of business development process is rare. Recent work on startup methodologies, such as Lean Startup [13] and Design Thinking [51] emphasizes the adoption of prototypes to increase chances of success through validated learning. Alternatively, startup prototypes need to be developed to

satisfactorily serve their purposes, i.e. technical feasibility test, demonstration to early customers, and fund raising. We argue that the prevalent Software Engineering practices used by startups to develop their first product inefficiently integrate into startups' dynamic contexts. Hence we call for research in understanding the development and usage of prototypes in startup contexts:

- RQ1: How can prototyping be used to maximize learning experience?
- RQ2: How can prototyping be used for optimization?
- RQ3: How can prototyping be used to support communication with external stakeholders?
- RQ4: How do prototypes evolve under the multiple influences of startups' stakeholders?

Early stage startups are lacking actionable guidelines for making effective prototypes that can serve multiple purposes. We believe that many startups will economically and strategically benefit by having proper practices in prototyping, such as technology evaluation (RQ1), strategic planning (RQ2) and customer involvement (RQ3).

To understand prototype development and its usage in startups, i.e. answering the first three research questions, exploratory case studies can be conducted. Cases would be selected to cover different types of startup prototypes at different phase of startup progress. A large-scale survey can be used to understand the prototype usage patterns, i.e. answering RQ4.

Despite an increasing body of knowledge on software startups [2], empirical research on prototyping processes and practices are rare. A few studies have investigated the adoption of software prototypes in combination with Design Thinking [52] and proposed prototyping techniques [52–54]. However, these studies rely on a very limited number of cases. Moreover, different constraints on prototyping decisions are often neglected. Future work can address antecedence factors, i.e. the involvement of lead-users, available human resources, and technological push, and how they impact prototyping strategies and usages in different startup contexts [55].

3.1.5. Risk Management Tools for Software Startups

The management of risk, namely the risk of failing to meet one's goals within given constraints in budget and/or time, is of paramount importance in every human activity. In the context of software startups, risk management looks unconventional, because startups naturally involve a much higher risk than traditional businesses. Yet, perhaps even more so than in traditional contexts, evaluating and managing risk in the software startup context might be a key factor for success.

Risk factors can be identified as a check-list of the incidents or challenges to face. Each of them could be categorized and prioritized according to its probability and the impact level of its consequences. This research track aims to study, model, and quantify various aspects related to risk management in software startups, with the goal of providing tools, based on process simulation, that control risk. Being able to efficiently model and simulate the startup process and its dynamics, would support startups in timely decision making. While numerous other approaches to risk control exist [56], we have found in our previous work [57, 58] that process simulations can be effective in risk management. Therefore, the overarching questions in this research track are:

- RQ1: To what extent do software startups explicitly manage risk?
- RQ2: To what degree is it feasible to model software development processes in startups?
- RQ3: To what extent can these models be used to quantify the risk of exceeding project budget or time?
- RQ4: What systematic ways exist to understand when to pivot or persevere [13], and what might be the cost of a wrong or untimely decision?

Following our previous experiences in software process modelling and simulation, to gain a better understanding is necessary to identify and analyse significant activities, not limited to the software development phase, of a software

startup (RQ1). This is necessary to be able to identify the critical aspects of startup development risks that are suitable for simulation. In our previous work we studied the application of Event-Driven models and/or System Dynamics to the software development processes. From this work we know that it is possible analyse project variations in time and budget with a Monte Carlo approach, by performing several simulations of the same project, varying the unknown parameters according to given distributions, and calculating the resulting distributions of cost and time of the simulated projects. Such analysis allows one to compute the Value At Risk (VAR) of these quantities, at given VAR levels. While Cocco et al. [57] and Concas et al. [58] provide exemplar studies of the application of these techniques in mature (agile) software development contexts, the question is whether such an approach is suitable and beneficial for software startups, and under what conditions (RQ2). By simulating the evolution of a startup as a process, we might be able to make predictions on its future development. Such predictions, or a result that can be rapidly be drawn from simulations, might be crucial for startups to understand which decisions are less costly and/or risky (RQ3). This is particularly true for decisions related to fields such as market strategies, team management, financial issues or product development (RQ4).

3.1.6. Startup Support Tools

Support tools can help software startups get their business off the ground with less pain and more guidance. These tools generally embed crucial knowledge regarding startup processes and activities. A plethora of tools (mostly software tools) exist for meeting the different needs of entrepreneurs and supporting various startup activities. For example, the web-page[6] by Steve Blank, a renowned entrepreneurship educator, author, and researcher from Stanford University, contains a list of more than 1000 tools. Well-designed portals such as Startupstash.com ease access to these supporting tools.

However, due to the lack of time, resources, and/or necessary knowledge, entrepreneurs cannot easily find the tools that best suit their needs, or cannot effectively utilize these tools to their potential. Existing studies provide limited insights on how entrepreneurial teams could find, use and benefit from support tools. Hence, the overarching questions in this research track are:

- RQ1: What are the needs of software startups that can be supported by software tools?
- RQ2: What are the tools that support different startup activities?
- RQ3: How can support tools be evaluated with respect to their efficiency, effectiveness, and return-on-investment?
- RQ4: How can support tools be effectively recommended to entrepreneurs and used by them?

RQ1 and RQ2 are targeted at identifying a match between the needs of software startups and the available tool support. To enable robust recommendations, both the individual startups and the software tools need to be objectively characterized allowing for their evaluation w.r.t. certain quality criteria (RQ3). There are potential synergies with the research track looking at the context characterization of software startups (Section 3.1.1). Answers to these research questions can be also valuable input for software tool vendors to develop the right tools that are needed by startups. In addition, the findings can be useful for future studies that develop proof-of-concept prototypes to support startup activities.

To investigate the proposed questions, various research methods can be applied, including survey of software startups regarding their needs and usage of support tools, in-depth case study of adoption and use of support tools, and design science approach to develop recommender systems of support tools (RQ4).

Research on tooling aspects in the software startup context is scarce. Edison et al. [59] argue that, despite the fact that different startup supporting tools have been developed and published over the Internet, new entrepreneurs might not have sufficient knowledge of what tools they need when compared to experienced entrepreneurs. In

[6] http://steveblank.com/tools-and-blogs-for-entrepreneurs/

addition, not all tools will help entrepreneurs in certain tasks or situations. Entrepreneurs' experiences using the tools can serve as the basis for evaluating and recommending appropriate tools. Besides suggesting a new categorization of existing startup support tools, Edison et al. [59] propose a new design of a tool portal that will incorporate new ways to recommend tools to entrepreneurs, especially to those who engage for the first time in a software startup endeavour.

3.1.7. Supporting Software Testing

Testing software is costly and often compromised in startups [60], as it is challenging for startups to fulfil customer needs on time, while simultaneously delivering a high quality product. In many software startups there is a common slogan that says "done is better than perfect", which indicates a general tendency toward a lack of testing and quality assurance activities [61]. However, it is sometimes also observed that startups do not know how and what to test; they lack expertise to test requirements as they do not have knowledge about their customers and users [61]. Therefore considering testing in software startups poses the following research questions:

- RQ1: To what extent does software testing in startup companies differ from traditional companies?
- RQ2: To what extent does testing evolve over time in software startup companies?
- RQ3: What is an optimal balance between cost/time spent on testing and development activities?
- RQ4: How can a software startup leverage customers/users for testing?

Answering RQ1 would provide insights on the aspects that differentiate the software testing process in startups from mature companies. For example, integration testing is likely very important for startups due to the fast paced product development. At the same time however, startups tend to work with cutting edge technologies, requiring a robust and flexible test integration platform. Connected to this is the question whether testing needs change over time, while the software startup matures. Answers to RQ2 and RQ3 would be particularly valuable for practitioners who could then better allocate resources. Users of software could be used for different testing purposes. On one hand, users provide valuable feedback in testing assumptions on customers needs. On the other hand, early adopters that are more robust towards deficiencies can help to improve product quality before targeting a larger market. Answers to RQ4 would provide strategies to harvest these resources.

In order to answer these research questions, various empirical research methods could be utilized. The studies would be devised in a way that "contrasting results but for anticipatable reasons" could be expected [62], i.e. different software startup companies would be taken into account to acquire a broad view of testing in software startups.

To the best of our knowledge, software testing in software startups has been scarcely researched. Paternoster et al. [2] highlighted the quality assurance activities in software startups in their mapping study. They found that it is important to provide software startups effective and efficient testing strategies to develop, execute, and maintain tests. In addition, they highlighted the importance of more research to develop practical, commercial testing solutions for startups.

3.1.8. User Experience

User experience (UX) is described as "a person's perceptions and responses that result from the use or anticipated use of a product, system or service" [63]. Good UX can be seen as providing value to users, as well as creating a competitive advantage. UX is important for software startups from their earliest stages. Firstly, human-centred design methods such as user research and user testing can help startups better understand how they can provide value to users and customers, as well as what features and qualities need testing for users to be satisfied with their product. Combined with business strategy, this human-centred approach helps startups move towards successful, sustainable business creation. Secondly, providing an initially strong UX in the first product versions can create positive word of mouth [64],

as well as keep users interested in the product for a longer time [65]. Genuine interest from users for the product idea while the product is still a prototype helps gain meaningful feedback [65]. Compared to more established businesses, software startups may pivot resulting in new target markets and user groups. This means efforts put into designing UX need to be faster and less resource consuming. Furthermore, failing to deliver satisfying UX can be fatal to small startups that can not cover the costs of redesigning. The overarching questions in this research track are:

- RQ1: What useful methods and practices exist for creating UX in startups?
- RQ2: What is UX's role during different phases of a startup's life-cycle?
- RQ3: To what extent are UX and business models connected in customer value creation?

An answer to RQ1 can provide software startups methods for developing strong UX in the first product versions which can keep users interested in the product for a longer time [65]. Genuine interest from users for the product idea while the product is still a prototype helps to gain meaningful feedback [65]. For business creation, understanding the value of UX for startups (RQ2) helps assigning enough resources for creation of UX while not wasting resources where there is no value to be gained (RQ3).

Research on startups and UX has been very limited. Some case studies report UX's role in building successful startups [66, 67]. Practices and methods for UX work in startups have been reported in [65, 68, 69]. A framework for creating strong early UX was presented by Hokkanen et al. [70]. These provide some results on feasible and beneficial UX development in startups, but more generalizable results are needed.

3.2. Startup Evolution Models and Patterns

The research tracks in this cluster share the theme of studying, identifying, and differentiating the transformation of startups in different stages. This also includes studies about different business and technical decision-making practices.

3.2.1. Pivots in Software Startups

It is very difficult for software startups to understand from start what are the real problems to solve and what are the right software solutions and suitable business models. This is evidenced by the fact that many successful software startups are different from what they started with. For example, Flickr, a popular online photo sharing web application, originally was a multiplayer online role playing game [71]. Twitter, a famous microblogging application, was born from a failed attempt to offer personal podcast service [71].

Due to their dynamic nature, software startups must constantly make crucial decisions on whether to change directions or stay on the chosen course. These decisions are known as *pivot* or *persevere* in the terms of Lean Startup [13]. A pivot is a strategic decision used to test fundamental hypothesis about a product, market, or the engine of growth [13]. Software startups develop technology intensive products in nature. Due to this, these are more prone to the rapidly changing technology causing pivots. Similarly, certain types of pivots are more relevant to software startups e.g. zoom in pivot: a pivot where one feature of a product become the whole product as in the case of Flickr. Pivot is closely linked to validated learning, another key concept from Lean Startup. The process to test a business hypothesis and measure it to validate its effect is called validated learning [13], whereas pivot is often the outcome of validated learning. A recent study [22] reveals that startups often neglect the validated learning process, and neglect pivoting when they need to, which leads to failure. This shows the importance of pivoting for a startup to survive, grow, and eventually attain a sustainable business model. In order to better understand and explore the pivoting process in the software startup context, the following fundamental research questions can be formed:

- RQ1: To what extent is pivoting crucial for software startups?
- RQ2: How do software startups pivot during the entrepreneurial/startup process?

- RQ3: What are the existing process/strategies/methods to make a pivoting decision in a startup context?
- RQ4: How do pivots occur during different product development and customer development life cycles?

Answering RQ1–RQ2 is necessary to understand pivoting in the context of software startups, building a fundamental framework on reasons for pivoting and their types. RQ3–RQ4, on the other hand, are targeted at understanding pivoting decisions and mechanisms. The overall contribution of answering the stated research questions has implications for both researchers and practitioners. The answers would provide an empirically validated conceptual and theoretical basis for the researchers to conduct further studies regarding the pivot phenomenon. For the practitioners, it would help them to make informed decision regarding when and how to pivot in order to increase the chances of success.

Due to the nascent nature of software startup research area, exploratory cases studies is a suitable approach to answer the research questions. Followed by the case studies, quantitative surveys can also be conducted to further generalize the results regarding pivoting in software startups.

Recently, there were some studies conducted on pivots in software startups. A study by Van der Van and Bosch [72] compares pivoting decisions with software architecture decisions. Another study by Terho et al. [73] describes how different types of pivots may change business hypothesis on lean canvass model. However, these studies lack the sufficient detail to understand different types of pivots and the factors triggering pivots. A study by Bajwa et al. [74], presents an initial understanding of different types of pivots occurred at different software development stages, however it lacks the deeper understanding of the pivoting decision that can only be achieved by a longitudinal study.

3.2.2. Determination of Software Startup Survival Capability through Business Plans

Software startups are highly specialized from a technological point of view. Focusing on the economic exploitation of technological innovations [75], they belong to the group of new technology-based firms. Literature suggests that one of their major challenges is the transformation of technological know-how into marketable products [76, 77]. New technology-based firms often struggle with unlocking the product-market fit [78] and commercializing their technological products [76]. Applying a resource-based view does thus not suffice for explaining survival and growth of software startups [79, 80]: a crucial success factor is the ability of new technology-based firms to understand and interact with the market environment to position their products accordingly [81, 82].

Particularly in early lifecycle stages, new technology-based firms need to build network relations with the market. Network theory literature suggests that with increasing network maturity, the chances for survival and growth increase [83–85]. The ability to transform resources in response to triggers resulting from market interactions can be described as a dynamic capability [86–89] which helps software startups commercialize their products. This transformation process captures the evolution of new technology-based firms in their early-stages. Current research is based on the construct of "venture emergence", which provides a perspective on the evolutionary change process of new technology-based firms [81, 90]. Venture emergence reflects the interaction process with agents and their environments [91]. Business plans of new technology-based firms are used as the artefact for measuring the status of venture emergence. They contain descriptions of transaction relations [92–94] new technology-based firms build in four market dimensions: customer, partner, investor, and human resources [95]. This research track intends to answer a number of research questions:

- RQ1: How reliably can annotated transaction relations from business plan texts determine the venture emergence status of technology-based startups?
- RQ2: To what extent are the number and strength ("level") of identified transaction re-

lationships useful as an indicator of survival capability?
– RQ3: How can patterns of transaction relations be used as an indicator for evaluating strengths and weaknesses of new technology-based firms, and thus be used to more effectively direct support measures?

While it is possible to measure the venture emergence status even in a software startup's very early stages, the predictive strength of transaction relations needs to be evaluated (RQ1–RQ2). This use of network theory to operationalize the venture emergence construct is a new approach, which adds to network theory literature in the context of the survival of new technology-based firms. It further confirms the business plans of new technology-based firms as a valuable source of information on startup potential. Finally, the resource-based approach to explain venture survival is enriched by applying a process-oriented perspective: we analyse resource transformation, rather than only looking at the initial resource configuration (RQ3). Furthermore, the research can contribute to the effectiveness of the innovation system by investigating indicators that reveal strengths and weaknesses of new technology-based firms. These can be used to direct support measures to software startups more effectively.

To answer the stated research questions, one can use content analysis [96, 97], combining human and computer-based coding of business plans, to determine the number and strength of transaction relations [98, 99].

Initial statistical tests that have been performed on a sample of 40 business plans of new technology-based firms confirm the relationship between the status of venture emergence of new technology-based firms and venture survival [99]. Earlier work led to the development of the concept for analysing early-stage startup networks and the relevance for survival [95]. Based on this concept, a coding method for transaction relations in business plans has been developed and validated with 120 business plans [98].

3.3. Cooperative and Human Aspects in Software Startups

The research tracks in this cluster address challenges and practices related to how people cooperate and work is software startups.

3.3.1. Competencies and Competency Needs in Software Startups

Software startups set different competency requirements on their personnel than more established companies. The biggest differences occur in two phases of the evolution of startups which have an impact on the nature of software development and competence needs: (1) in the early stages of rapid software development when there is a lack of resources and immature competencies in many key areas, and (2) when the rapid business growth of successful startups requires management of a fast growing personnel and amount of software with limited management resources and competencies. In the early phases strong competition requires the software startup to innovate and react quickly [2], and deployment of systematic software engineering processes is many times replaced by light-weight ad-hoc processes and methods [2,26]. The nature of software makes it possible for successful startups to scale fast [2]. Rapid software-driven growth requires fast scaling of the software production, distribution, and maintenance. The required competences also quickly evolve when software development moves from rapid greenfield prototyping to professional software development and management. Mastering this demanding situation often requires a broad prior skill basis from the startup team, including an ability to adjust to changes, and learn quickly.

Research on specific skills and competency needs in software startups broadens not only the knowledge on software startups themselves, but also broadens the knowledge on software engineering conducted under the challenging circumstances of startups. Focusing the research on the early stages and on the growth period of the software startups, when the challenges of the software startups are the great-

est [12, 22], brings the most valuable knowledge to both academia and practitioners. Competency research also brings human factors into focus [100, 101], and reinforces the results of existing software startup research towards a more comprehensive modelling and understanding. The research questions for studies on competencies and competency needs in software startups include:

- RQ1: Software startup challenges and competency needs – what software development knowledge and skills are needed to overcome the challenges?
- RQ2: What are the competency needs specific for software startups compared to the more established software companies?
- RQ3: How do the competency needs change over the evolution of software startups?
- RQ4: How do the competency needs map onto the roles and responsibilities of the startup teams in software startups?
- RQ5: How can the growth of software startups be managed in terms of competency needs for software development practices, processes and recruitment?

Research on software startups, including research on competency needs, provides the research and development of software engineering with new knowledge and viewpoints on how to direct the work in order to best address the specific challenges of the software startups (RQ1). In particular, differences to mature software companies are interesting to study (RQ2) considering software startups evolve, if they survive, to established companies. Knowing how competency needs change might turn out as one key factor for this transition (RQ3). Theoretical models describing the evolution paths of software startups have been created [13, 102], but competency needs and how they map to roles and responsibilities have been to a large degree ignored (RQ4). Similarly, while software development work [2] and software engineering practices [26] have also been studied, it is unclear how competency needs can be managed in growing software startups (RQ5).

3.3.2. Teamwork in Software Startups

The importance of human aspects in software development is increasingly recognized by software engineering researchers and practitioners. Teamwork effectiveness is crucial for the successes of any product development project [103]. A common definition of a team is "a small number of people with complementary skills who are committed to a common purpose, set of performance goals, and approach for which they hold themselves mutually accountable" [104]. A startup team is special in the wide range of variety, including both technicians and entrepreneurs.

While an innovative idea is important for the formation of a startup, startup success or failure ultimately rests on the ability of the team to execute. Entrepreneurship research showed that over 80 percent of startups that survive longer than two years were founded by a group of two or more individuals [105]. The dynamic and intertwined startups activities require the close collaboration not only among startup team members, but also with external stakeholders, such as mentors and investors. Given the diversity in mindsets and skill sets among founders, it is essential that they can work well together along with the startup life-cycle. The movement with recent methodology in Lean startup introduces an opportunity to look at startup teams from various angles, i.e. pivoting, startup culture, team formation, and decision-making. The overarching questions in this research track are:

- RQ1: Is there a common cultural/organizational/team characteristic among successful software startups?
- RQ2: How can a software startup team effectively communicate with other stakeholders, i.e. mentors and investors?
- RQ3: How can a software startup manage team internal relationships?
- RQ4: What are the common patterns of competence growth among software startup teams?

Understanding software startup team behaviour to internal and external environments and relating them to startup success measures would help

to identify characteristics and teamwork patterns of successful startups. Answering RQ1 would provide practitioners some guidance on how to form startup teams while answers to RQ2–RQ3 would provide an understanding how internal end external team dynamics work and can improved. An answer to RQ4 would also support the work in Section 3.3.1, looking however specifically at competence growth patterns that could be valuable for practitioners when deciding on what to focus on in competence development. Empirical studies, i.e. case studies, surveys and action research are all suitable to investigate the stated research questions. Among them, comparative case studies would be the first option to discover the difference in startup teamwork patterns.

There exists a large body of literature in business management, entrepreneurship, and small ventures about entrepreneurial teams' characteristics and their relationship to startup outcomes [105–107]. In Software Engineering, few empirical studies identified team factors in the failure of software startups. Giardino et al. found that building entrepreneurial teams is one of the key challenges for early-stage software startups from idea conceptualization to the first launch [12]. Crowne et al. described issues with founder teamwork, team commitment and skill shortages [108]. Ensley et al. investigated the relative influence of vertical versus shared leadership within new venture top management teams on the performance of startups [109]. Other team dimensions are explored in the business and engineering management domain in specific geographies. E.g., Oechslein analysed influencing variables on the relational capital dimension trust within IT startup companies in China [105]. How generalizable these influencing variables to other geographies is yet to be seen.

3.4. Applying Startup Concepts in Non-Startup Contexts

One of the Lean Startup principles claims that entrepreneurs are everywhere, and that entrepreneurial spirits and approaches may be applied in any size company, in any sector or industry [13]. On the other hand, established organizations face the challenge of innovation dilemma and inertia caused by the organization's stability and the maturity of markets [110]. Therefore, applying startup concepts in non-startup contexts seems an promising avenue for established organizations to improve their innovation potential.

3.4.1. Internal Software Startups in Large Software Companies

The internal software startup concept has been promoted as a way to nurture product innovation in large companies. An internal software startup operates within the corporation and takes responsibility for everything from finding a business idea to developing a new product and introducing it to market [111]. Internal software startups can help established companies master the challenge of improving existing businesses, while simultaneously exploring new future business that sometimes can be very different from existing ones [112]. Usually, this involves a conflict of interest in terms of learning modes [113] or risk propensity [114], which can be prevented by establishing dual structures within the organization for implementing internal software startups [115]. Compared to the traditional R&D activities of larger companies, an internal software startup develops products or services faster [2] and with higher market orientation [116]. This helps established companies maintain their competitiveness in volatile markets [117].

Besides the fact that the successful implementation of internal software startups faces various barriers, such as cultural conflicts [118] or the fear of cannibalization of existing businesses [119], internal software startups can also benefit from being part of established companies. Shared resources, such as capital, human resources [120, 121], and the access to the corporates' internal and external network [122] are just some benefits.

Earlier research on analysing the results of startups' value creation cycle has taken place in the context of the evolution of the enterprise [123]. However, this occurs over too long of a time period to be useful for guiding software development. Measuring the cycle time of the software

engineering process to the completion of a software feature is also insufficient. The Lean startup approach [13] has been commonly adopted to new business creation in software intensive ventures. They use the learning loop to discover the customer value and potential of the new product concept, as well as to find new means to produce software. Tyrväinen et al. [124] propose that measuring the cycle time from development to analysis of customer acceptance of the feature enables faster learning of market needs. In addition, receiving fast feedback from users makes changing the software easier for the programmers who have not yet forgotten the code. Relevant research questions regarding internal software startups can be formulated as follows:

- RQ1: How can Lean startup be adopted and adapted for software product innovation in large software companies?
- RQ2: What are the challenges and enablers of Lean startup in large software companies?
- RQ3: How should internal software startups be managed/lead?
- RQ4: What metrics can be used to evaluate software product innovation in internal startups?
- RQ5: To what extent do internal startups have a competitive advantage compared to independent startups (through shared resources, etc.)?

Lean startup approach gains more interest from scholars and academics as a new way to foster innovation since it helps to avoid building products that nobody wants [125]. Some evidence shows that mature software companies and startups differ in applying Lean startup approach [126]; e.g. mature firms start the cycle by collecting data from existing users and then generating a hypothesis based on that data, whereas software startups generate ideas and collect data from new users to validate the ideas. However, it seems that, to a large extent, the approach can be used both in startups and established enterprises. By answering RQ1–RQ3 we aim at defining structured guidelines on how to introduce Lean startup in large software companies, supporting practitioners, while answering RQ4-RQ5 would provide a motivation for this approach, allowing to compare effectiveness on a quantitative level.

Due to the complex nature of the research phenomenon and the intention to achieve an in-depth understanding of it, we consider multiple case studies [62] as a suitable research approach. The case organizations can be selected based on the following criteria: (1) the organization develops software in-house, (2) a dedicated team is responsible from ideation to commercialization of a new software, and (3) the software falls out of the current main product line. The unit of analysis in this study would be a development team.

Very few studies have investigated how the Lean startup [13] can leverage internal startups in large software companies to improve their competency and capabilities of product innovation. Initial steps have been taken and some of the results have been published to fill this observed gap (e.g. [119, 127]). Marijarvi et al. [128] report on Finnish large companies' experience in developing new software through internal startups. They also discuss the lifecycle phases of innovation work in large companies. The authors argue that different types of internal organization may take place in each stage of new product development. For example, problem/solution fit can be done in an internal startup or company subsidiary.

3.4.2. Lean Startup for Project Portfolio Management and Apen Innovation

Building on the challenges proposed in Section 3.4.1, we propose that Lean startup could also be applied within both (i) project portfolio management (PPM), to co-ordinate multiple startup initiatives within an organization, and (ii) open innovation, wherein internal startups involve multiple organizations, individuals, or even unknown participants. Both PPM and open innovation and their main challenges are briefly introduced below, followed by research questions that require investigation before Lean startup principles can be successfully applied in these new contexts.

Software engineering PPM describes the ongoing identification, selection, prioritization, and management of the complete set of an organization's software engineering projects, which share common resources in order to maximize returns to the organization and achieve strategic business objectives [44, 129–131]. Open innovation is defined as the use of "purposive inflows and outflows of knowledge to accelerate internal innovation and to expand the markets for external use of innovation, respectively" [132]. Popular examples of open innovation include open source software development, crowd-sourcing, and inner source.

Effective PPM is critical to achieving business value [133, 134], improving cost and time savings, and eliminating redundancies [135, 136]. Unfortunately, existing portfolio management practices, which are based on the effective completion of individual projects with only episodic portfolio level reviews [134], fail to manage either the dynamic nature of contemporary projects, or problems associated with portfolios comprising too many projects [134, 137]. Indeed, many portfolios report an unwillingness to cancel projects that no longer contribute to the achievement of strategy [134].

Open innovation (OI) presents numerous advantages for organizations, such as access to a requisite variety of experts, a prospective reduction in overall R&D spending, reduced time-to-market, improved software development processes, and the integration of the firm into new and collaborative value networks [132, 138, 139]. Nonetheless, adopting open innovation processes can be significantly challenging. For example, adopters often lack internal commitment, in addition to challenges associated with aligning innovation strategies to extend beyond the boundaries of the firm. Moreover, there are concerns regarding intellectual property and managing unknown contributors/contributions, as well as managing the higher costs and risks associated with managing both internal and external innovations [140–142]. The role of Lean startup principles in addressing these challenges in both PPM and OI is worthy of further research:

- RQ1: How can Lean startup be implemented within a portfolio management or open innovation context?
- RQ2: How can Lean startup initiatives drive or accelerate open innovation?
- RQ3: What Lean startup concepts could be adapted to facilitate open innovation processes in an organization?
- RQ4: How can one ensure Lean startup initiatives conducted across multiple projects or organizations align with strategy?
- RQ5: How do you reconcile potential conflicts between portfolio / open innovation processes and Lean startup processes?
- RQ6: How do you achieve consensus in defining the minimum viable product (MVP) in networks comprised of multiple autonomous (and sometime anonymous) agents?

The successful application of Lean startup principles (RQ1–RQ3) has the potential to reduce the costs arising from the poor implementation of PPM and OI practices and increase the value achieved from these initiatives. However, because such approaches are often practice led, it is necessary for academic research to develop effective theory to underpin practice and provide empirical data to support, or refute claims of effectiveness (RQ4–RQ6). Rich human interactions are at the heart of software engineering PPM and open innovation. Accordingly, phenomena in these domains can be examined using interpretive, qualitative methods such as semi-structured interviews, case studies and ethnography.

While the principles of lean have been applied to PPM (e.g. [143, 144], there is little research looking at the application of Lean startup principles to PPM. Similarly, while there is interest in the application of Lean startup principles in open innovation contexts, to date, such applications have predominantly been driven by practice.

3.5. Software Startup Ecosystems and Innovation Hubs

Successful software startups do not live in isolation. Normally, they are inserted in a rich environment that includes a number of relevant

players, such as entrepreneurs, developers, investors, scientists, as well as business and intellectual property consultants. To support these players, a number of support programs from the private and public sectors are required to provide funding, incubation, acceleration, training, networking, and consulting. All these elements combine into what scholars and practitioners have called Startup Ecosystems [145]. In our software startups research agenda, we focus on Software Startup Ecosystems (SSE) and the elements that are relevant for startups that have software as a key part of their products or services.

By studying how SSEs are created, their main characteristics, and how they can evolve, one can better understand the environments that favour, or not, the birth and development of successful software startups. Research in this field can provide, to the relevant stakeholders, the concrete actions (e.g., public policies, private activities) that will establish a fruitful and vibrant environment for the execution of high-growth innovative projects within nascent software companies. The main research questions that need to be answered are the following:

- RQ1: What are the key elements of a fruitful SSE?
- RQ2: Are there different types of SSEs, e.g. differentiated by size, technology sectors, country economy or other factors?
- RQ3: How do SSEs evolve over time?
- RQ4: How can one measure the output and qualities of an SSE?

By answering RQ1, researchers will provide a better understanding of the way how SSEs and innovation hubs work, instrumenting key stakeholders in taking actions to improve their ecosystems. By identifying what factors promote or hinder the development of successful startups within a certain SSE, policy makers will get support in decision making (RQ2). Entrepreneurs will also be able to better understand what are the environmental factors and forces that can help or hinder the success of their enterprises.

Researchers from Brazil, Israel, and the USA have developed a methodology to map a specific software startup ecosystem; this methodology has been applied to Israel [145], São Paulo [146] and New York [147]. Currently, with the help of dozens of experts worldwide, they are developing a maturity model for SSEs [145, 148], addressing RQ3 and RQ4. This maturity model needs further research and validation before it can be applied in real scenarios to help practitioners and policy makers.

The Global Startup Ecosystem Ranking [149] is crafted by a group of experts that have been proposing metrics to evaluate regional ecosystems around the world and compare them according to multiple criteria. Frenkel and Maital [150] have developed a methodology to map national innovation ecosystems and use this map to propose policies to promote improvement. Jayshree has studied the influence of environmental factors on entrepreneurial success [151]. Finally, Sternberg [152] researched the role of regional government support programs and the regional environment as success factors for startups.

3.6. Theory and Methodologies for Software Startup Research

The tracks in this cluster direct their research towards identifying means to better study and understand software startups.

3.6.1. Overview of the Possible Theoretical Lenses for Studying Software Startups

Theories are important in any scientific field, as they form the foundation to understand a contemporary phenomenon better. Theories provide answers to the "why" questions, and are therefore useful for explaining why certain events occur while others do not. Software startup research does not operate in a vacuum, but rather can borrow theories from both the software engineering and information systems fields, business and management literature, as well as from the fields of organizational and social sciences.

We have identified a few potential theories that can be meaningfully applied in the context of software startup companies. The proposed theories are the hunter-gatherer model [153], Cynefin model [154], Effectuation theory [155] and Bound-

ary Spanning theory [156]. These theories are briefly outlined in this section.

Although 90% of human history was occupied by hunters and gatherers, who forged for wild plants and killed wild animal to survive, only recently was the hunter-gatherer model re-discovered by Steinert and Leifer [153] to explain how designers pursue their endeavours in search of the best design outcome. The model shows the changes in the design process, as well as subsequently in the design outcome. The model portrays a distinction between a hunter who aims to find an innovative idea, and a gatherer who aims to implement the idea. Both are needed to achieve concrete results. While hunting the idea through ambiguous spaces has a change-driven, analytical, and qualitative nature; gathering the idea across predetermined paths has a plan-oriented, manageable, and quantitative nature. The model has recently been applied in software startup research to explain startups' evolutionary paths [157].

Complexity theory has been used as a frame of reference, by analysing its implications on software design and development (e.g. Pelrine [158], Rikkilä et al. [159]). Software projects can be characterized as endeavours wherein a dynamic network of customers, software designers, developers, 3rd party partners, and external stakeholders interact and can be seen as a Complex Adaptive System (CAS). To reason about decision-making in different situations, Snowden et al. [154] proposed a sense-making framework for such systems. The model has five sub-domains and divides the world in two parts – ordered and unordered main domains. The ordered domain is the one in which cause-effect (CE) relationships are known (the Known domain), or at least knowable after analysis (the Complicate domain). In contrast, the unordered domain includes a complexity situation, wherein the CE relationship can only be perceived in retrospect, but not in advance (the Complex domain), and a chaotic situation, wherein behaviours are completely random, lacking any expected consequence when acted upon. Depending on the problem domain, suitable approaches include categorizing, analysing, probing or acting [154]. The Cynefin model provides a framework that can be used to analyse the decisions made by software startuppers in developing their products. Often they find themselves in the unordered domain, attempting to make sense out of the current situation and navigate to the ordered domain.

Effectuation theory is a simple model, rooted in entrepreneurship, of decision-making under uncertainty. The effectual thinking is in the opposite of causal reasoning which starts from desired ends to necessary means (top-down). Experienced entrepreneurs reason from means to ends (bottom-up), trying to work out meanings and goals based on the resources they have at hand. The theory is embodied by five principles: the bird-in-hand principle, the affordable loss principle, the crazy quilt principle, the lemonade principle, and the pilot-in-the-plane principle [155]. The effectuation theory can help to make better sense of entrepreneurs' decision-making process in the evolution of software startups, such as problem validation, value proposition definition, design of MVPs, and pivoting processes. Good practices could be discovered using the effectuation theory as a theoretical lens.

Startups operate in a dynamic environment and face expectations and influences from many directions. In order to survive, they need to effectively collaborate within their team, but also outside it. Boundary spanning is a concept that deals with the structures of organizations that are transitioning from a rigid hierarchical structure towards a network-based expert organization, which gives rise to informal boundaries rather than structural ones [156]. Boundary spanners are those people and entities who bridge these boundaries and opportunities. In the software engineering context, boundary spanning has been studied in the context of global software development [160]. Startuppers can be seen as boundary spanners when they need to bridge between various stakeholders. While boundaries are always unavoidable, but also necessary and useful, knowledge is required on how they can be crossed, rearranged, or even dissolved when considered harmful [161]. Startuppers should see boundaries as tools that facilitate and support making sense out of the environment. Boundary

spanning helps in discovering how to overcome the challenges of distributed global work, where motivations, work styles, and knowledge domains vary across boundaries. Startuppers can become knowledge brokers, transferring and sharing their knowledge.

There are other theoretical lenses that can be used to study software startups. Startups deal with innovative services and products, often for new or emerging markets. Birkinshaw et al. [162] analyse the innovation theories presented and propose a framework for management innovation process. This could be applied to the startup innovation process context to explore how product development moves from problem-driven search through trial and error to a finished prototype. The analysis can be complemented with Van de Ven and Poole's [163] four views into organizational changes, in which they present alternate processes for organizations to transform.

Theorizing software startups is important, since there is a current lack of understanding of the dynamics in startups. Theoretical advancements need to be achieved so that researchers can make better sense out the diverse contexts, situations, and places where startuppers strive for success.

3.6.2. Defining the Lean Startup Concept and Evaluating Practice

Many positive drivers underpin the Lean Startup movement. The literature is abound with claims of reduced risk [13, 125], the benefits of evidenced-based trials [13, 164], and shorter time-to-market [13]. We certainly know that these benefits are needed, given the challenges experienced by early stage software startups [12,22] and the percentage that fail [13]. Indeed, many software startups fail [108, 165] because they waste too much time and money building the wrong product before realising too late what the right product should have been [102, 166]. These challenges coupled with high uncertainty make the Lean Startup Methodology attractive to software startups as it supposedly offers an integrated approach to creating products and services that fit the market [167]. This research builds on previous research conducted by Dennehy, Kasraian, O'Raghallaigh, and Conboy [168], which identified a significant absence of frameworks that assisted startups to efficiently and effectively progress their Minimum Viable Products (MVP) to a Product Market Fit (PMV). The theoretical advancement of the lean concept in contemporary software engineering and software development literature has been arrested, mainly because the academic research community has followed "fads and fancies" which characterize academic research. The implications for the arrested theoretical development of lean concept, listed next, are the motivation for this research.

As is often the case with new and emerging phenomena, Lean Startup practice has led research, with the creation, promotion, and dissemination of these methods almost completely due to the efforts of practitioners and consultants. Now, Lean Startup research is beginning to gain momentum, as is evident from the increasing number of dedicated journal special issues, conferences, conference tracks, and workshops. While there are merits to adopting such a practice-oriented focus, little if any research effort has focused on the conceptual development of Lean Startup and its underlying components. As practice has lead research, the definition of Lean Startup has emerged through how it is used in practice. As a result, Lean Startup adoption is often defined by how the practices are adhered to, rather than the value gleaned from their use, adaptation, or, in some cases, abandonment. We see this in many other methods such as in agile, where many define "being agile" as how many Scrum or XP practices are used, rather than the value obtained by their use [169]. As a result, the current body of software startup knowledge suffers from a number of limitations, including:

1. Lack of clarity: While there is broad agreement in principle regarding what constitutes key concepts such as MVP, assumptions regarding the specific definitions, interpretations, use, and evaluations are often unclear in many existing Lean Startup studies. This makes critical appraisal, evidence-based evaluation, and comparison across studies extremely difficult.

2. Lack of cohesion and cumulative tradition: A good concept or theory should cumulatively build on existing research. Very little academic research has examined Lean Startup using concepts that have more mature and substantive bodies of research with theories, frameworks and other lenses that have been thoroughly tested over time. The lean concept has been applied in manufacturing since WW1, and yet in Lean Startup research we see very myopic and limited use of the broad lean frameworks available. Other concepts that influence Lean Startup include agility, flow, and innovation.

3. Limited applicability: Adherence-based measures of Lean Startup inhibit the ability to apply Lean Startup in domains other than that originally intended. Research now attempts to apply Lean Startup in other environments, such as large organizations and regulated environments, and so this will become a more prevalent issue as this trend continues. Therefore, questions relevant for this research track include:

- RQ1: What are the core concepts that underpin Lean Startup?
- RQ2: What are the components of a higher abstract Lean Startup that allows the concept to be applied and evaluated in a value-based manner?
- RQ3: What theories, frameworks, metrics, and other instruments from these existing related bodies of knowledge can be applied to Lean Startup?
- RQ4: How can these be effectively applied to improve the use of Lean Startup in practice, and the study and improvement of Lean Startup in research?
- RQ5: How can Lean Startup then be tailored to suit environments it was not originally designed to support, e.g. large organizations, regulated environments, or peer production?
- RQ6: Does Lean Startup enable or inhibit fundamental leaps in business and software business ideas? For example, does MVP place an invisible ceiling, wherein once you reach MVP you subconsciously stop looking for the truly significant innovation?

As there is reciprocal relationship between practice and academia, where academic research is informed by practice and practice is informed by academic research, this research would impact on research and on practice. By answering RQ4–RQ6, this research track would provide practice with empirical evidence on the utility of lean practices in diverse environments, while also positioning the lean method at the core of academic research (RQ1–RQ3). As case study research is an empirical inquiry that "investigates a contemporary phenomenon in depth and within its real-life context" [62], it would be highly suited to addressing the theoretical limitations of lean and for answering the questions listed above. Specifically, the use of a multiple-case design would allow a cross-case pattern to develop more sophisticated descriptions and powerful explanations [170] of the lean concept.

The challenges of new product development are not confined to software startups. Therefore, software engineering teams working in distributed or regulated environments such as financial services and within multinational companies would provide rich insights to the advancement of the lean concept.

3.6.3. Research Collaboration Strategies with Software Startups

Empirical research in the area of software engineering normally requires access to organizations and artefacts from companies developing software intensive products and services [171]. In the case of startups, such access is very limited, due to several challenges:

1. startups have limited resources both in terms of person hours and calendar time for anything but working on their MVP,
2. startups want all investments to yield almost immediate results, thus investments in long-term potential are not prioritized, and
3. artefacts and actual products are often very sensitive, as the startup is very vulnerable.

These and other reasons limit empirical research, as reflected in both academic knowledge about startups overall, but also in the superficial nature of what is available. For this reason, any initiative to seriously collect empirical data as well as conduct research on core challenges facing

startups has to originate with a strategy that overcomes these obstacles. One possible strategy is to pool resources and access to startups, in essence sharing empirical data and coordinating research into startup software engineering. Coordination should be seen as equally central, as it enables researchers to limit the impact and costs as each study and project part can be focused and small, and several larger issues can be tackled through coordination. Concrete examples of joint activities include, but are not limited to:

1. joint surveys at the superficial level (pooling resources to collect many data points),
2. complementary surveys and case studies where each partner does a part only, but the results can be combined in analysis and synthesis,
3. formulating a complementary research agenda with clear interfaces and joint research questions, and
4. pooling resources in relation to testing "solutions" emerging from the collaboration.

While this strategy opens the possibility to share the resource requirements among the studied startups, there are open questions regarding its implementation:

– RQ1: To what extent is data from different startups and startup ecosystems comparable? In other words, which techniques exist to perform meta-analysis of the gathered heterogeneous data?
– RQ2: How can we efficiently transfer technology between researchers and startups, and how can we measure the impact of transferred solutions?

We conjecture that the software startup context model discussed in Section 3.1.1 would be an enabler for answering RQ1. Confounding variables [172] could then be easier identified, allowing for sample stratification and robust statistical analyses [173]. In particular, data collected from different researchers could be aggregated and increase the strength of the conclusions drawn from the analysis, i.e. enabling meta-analysis [174].

Answering RQ2 would allow us to actually support software startups on a broad basis with the knowledge gained from the research proposed in this agenda. While different approaches exist to transfer knowledge from academia to industry [175,176], they are mostly targeted at mature companies that have the resources to collaborate with researchers over a longer period of time. We think that software startup ecosystems, discussed in Section 3.5, can contribute to technology transfer if researchers are active in these structures and can create a win-win situation where both startups and researchers benefit.

4. Discussion

In this section we give a brief overview of the research tracks in relation to other work in software engineering and their potential impact on the field. We conclude this section with a discussion on the study's limitations.

Software startup engineering research centers around the core knowledge base in Software Engineering [177]. This is illustrated by the research tracks proposed in Section 3.1 that encompass providing support for startup engineering activities. Noticing what is considered "good" software engineering practice [177], and the challenges that software startups encounter [12, 24], we see potential in directing research towards efficient and effective requirements for engineering practices in startups. Klotins et al. [24] studied 88 experience reports from startups and identified lack of requirements validation, classification (to enable prioritization), and identification of requirements sources (to identify a relevant value proposition) as causes for engineering uncertainty, which maps to the early-stage startup challenges of technology uncertainty and delivering customer value, identified by Giardino et al. [12]. Unlike large companies, software startups have unique time and resource constraints and thus cannot afford to develop features and services that will not be used or valued by the customers. We believe that lightweight practices to identify, and, most importantly, analyse requirements for their business value can help software startups in their decision process. Looking at the research tracks in Section 3.1, several of them touch upon requirements engineering aspects. Prototypes can be used to communicate with customers to elicit

requirements (Section 3.1.4), while product innovation assessment (Section 3.1.3) is relevant in the context of analysing the customers' perceived value of the offered solutions. Even optimizing the effort spent in requirements engineering and quality assurance, for example by using test cases as requirements [178], involving product users for testing (Section 3.1.7), addresses requirements engineering aspects.

The focus on requirements in software startup engineering research directly relates to the research tracks presented in Section 3.2, startup evolution models and patterns, as the cost of pivoting could be reduced by earlier and less ad-hoc analysis of requirements and value propositions of the envisioned products. The patterns emerging from the research on survival capabilities of software startups, proposed in Section 3.2.2, could provide valuable heuristics leading to a lightweight analysis of product value propositions. The research on pivoting and survival capabilities is likely to affect software startup practitioners on a strategic level by providing them managerial decision support that draws from models rooted in software engineering practice. An example where such a cross-discipline approach has been very successful is value-based software engineering [179].

The research tracks described in Section 3.3 were grouped under the name "cooperative and human aspects in software startups", borrowed from the research area in software engineering that is interested in studying the impact of cognitive abilities, team composition, workload, informal communication, expertise identification and other human aspects on software construction [180]. We conjecture that studying and understanding these aspects better has a large potential as software startups are driven by motivated individuals rather than a corporate agenda. Lessons from this research can both benefit startup practitioners, in particular in conjunction with the work on software startups ecosystems (Section 3.5), and more mature companies, for example by applying models of competency needs that could emerge from the work presented in Section 3.3.1.

The remaining research tracks described in Sections 3.4 - 3.5 take a step back from what happens *inside* a software startup. The research tracks in Section 3.4 propose to apply startup concepts in non-startup contexts. The idea of extracting a concept from one context and applying it in another has proven successful in other areas, such as in systematic literature reviews [181, 182] and open source principles [183–185]. The premise of internal startups is that the positive traits of "startups in the wild" can be transferred to a corporate environment, fostering innovation and faster product development. The overall aim of the research tracks described in Section 3.4 is to evaluate whether the traits of startups can actually produce thriving environments within mature companies. In comparison, the research on startup ecosystems and innovation hubs (Section 3.5) takes a broader and higher level view of software startup phenomenon. Neither independent startups nor mature companies adopting internal startup initiatives live in isolation. A better understanding of startup ecosystems and innovation hubs might thereby provide key insights into the factors that create a fruitful software startup environment.

Finally, the research tracks in Section 3.6 look at aspects relevant for implementing the research agenda described in this paper. In particular, theories that can be used to better understand the dynamics in and around software startups are of value when attempting to construct a more holistic understanding of software startups in their various contexts. For the research on defining the Lean Startup concept, parallels to and lessons from similar endeavours around research on agile software development [186] should be taken into consideration. In this paper, we followed a recommendation by Dybå and Dingsøyr to develop a research agenda on the phenomenon of interest [186]. However, in order to implement this research agenda, we need to also answer the questions about how to enable efficient and effective research collaborations with software startups (Section 3.6.3).

4.1. Limitations

The research agenda presented in this paper was developed "bottom-up", i.e. the areas of interest were proposed and described by a sample of soft-

ware startup researchers without any restriction on covering certain aspects of the software engineering body of knowledge but guided by their past, current and future work in the field. Often, these researchers have both a leg in academia and in the startup community, either as mentors, founders, or simply as part of the development team. This approach to develop a research agenda is not uncommon (see e.g. [187–189]), but is threatened by a potential bias towards the preferences of individual researchers. This is why we invited a large number of our peers to contribute to the agenda. Even though the research tracks cover many software engineering aspects and beyond, the agenda is only a sample of the potentially relevant future research on software startups. This means that potentially interesting and relevant research topics, such as use of open source software, business model development, legal issues and intellectual property rights, are not discussed in this paper. However, we expect that the agenda will grow together with the research community as soon as the work on the proposed research tracks bears fruits, leading to new research questions.

5. Outlook and Conclusions

Software startups are an interesting and stimulating phenomenon in the modern economy and are of paramount importance for the societies of today. Despite of high failure rates, communities, cities and countries are investing on stimulating the creation of software startups. While these startups may not solve the unemployment problems of many countries they stimulate a new type of positive dynamism in societies encouraging people to collaborate and develop their personal skills in novel ways. The emergence of the software startup research area reflects the fact that we need to better understand this phenomenon to learn valuable lessons and accumulate valid knowledge to benefit future entrepreneurial initiatives. The research agenda described in this paper is one of the first attempts to establish the software startup as a nascent, yet fast growing research area, and to depict its landscape by highlighting the interesting research topics and questions to explore.

It is worth emphasizing again that software engineering is only one of the multiple disciplines that are relevant and can inform software startup practice. Other disciplines include Economics, Entrepreneurship, Design, Finance, Sociology, and Psychology. Therefore, there is a need to collaborate with researchers from these disciplines in order to increase the potential of achieving relevant and useful research results that can benefit practice.

Due to the emerging nature of the field, there is still much to be done to establish software startups as a research area. Relevant concepts need clear definitions, substantive theories need to be developed, and initial research findings need to be validated by future studies. Software startups are very diversified in terms of entrepreneurs' varying approaches to their startup endeavours. Without the sound foundation mentioned above for this research area, there are risks of asking irrelevant research questions and not being able to attain rigorous results.

Last but not least, this research agenda is not meant to be exhaustive, and we are aware that we may exclude some important Software Engineering topics relevant to software startups. The research agenda is open to additions of new tracks, topics, and research questions by other researchers interested in the research area. With contributions and commitments from researchers from different institutions and backgrounds, collectively we can establish software startup as a promising and significant research area that attracts more exciting discovery and contribution. We welcome those interested in joining the Software Startup Research Network in fostering the collaboration between researchers and taking the research agenda further.

References

[1] *42010-2011ISO/IEC/IEEE Systems and software engineering – Architecture description*, ISO Std., 2011.

[2] 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.

[3] "Frequently asked questions about small business," U.S. Small Business Administration, Tech. Rep., 2014.

[4] S. Nambisan, K. Lyytinen, A. Majchrzak, and M. Song, "Digital Innovation Management: Reinventing Innovation Management Research in a Digital World," *MIS Quarterly*, 2016, in press.

[5] "A cambrian moment cheap and ubiquitous building blocks for digital products and services have caused an explosion in startups. special report: Tech startups," *The Economist*, 18/01 2014.

[6] WMF, "Intelligent assets unlocking the circular economy potential," World Economic Forum, Tech. Rep., December 2015.

[7] S. Srinivasan, I. Barchas, M. Gorenberg, and E. Simoudis, "Venture Capital: Fueling the Innovation Economy," *Computer*, Vol. 47, No. 8, 2014, pp. 40–47.

[8] A. Shontell, "The 11 most disruptive startups," *Business Insider*, 12/07 2012. [Online]. http://www.businessinsider.com/disruptive-startups-2012-7?op=1&IR=T

[9] C. Giardino, N. Paternoster, M. Unterkalmsteiner, T. Gorschek, and P. Abrahamsson, "Software Development in Startup Companies: The Greenfield Startup Model," *Transactions on Software Engineering*, Vol. 42, No. 6, 2016, pp. 585–604.

[10] "Progress without profits. A flock of startups is making cloud computing faster and more flexible, but most of them will not survive," *The Economist*, 19/09 2015.

[11] "Testing, testing," *The Economist*, 18/01 2014.

[12] C. Giardino, S.S. Bajwa, X. Wang, and P. Abrahamsson, "Key Challenges in Early-Stage Software Startups," in *Proceedings 16th International XP Conference (XP)*. Helsinki, Finland: Springer, 2015, pp. 52–63. [Online]. http://link.springer.com/chapter/10.1007/978-3-319-18612-2_5

[13] E. Ries, *The lean startup: How today's entrepreneurs use continuous innovation to create radically successful businesses*. Crown Books, 2011.

[14] S. Blank, *The Four Steps to the Epiphany: Successful Strategies for Products that Win*. Cafepress.com, 2005. [Online]. http://www.amazon.com/The-Four-Steps-Epiphany-Successful/dp/0976470705

[15] S. Blank and B. Dorf, *The Startup Owner's Manual: The Step-By-Step Guide for Building a Great Company*. K & S Ranch, 2012.

[16] S.M. Sutton, "The Role of Process in a Software Start-up," *IEEE Softw.*, Vol. 17, No. 4, 2000, pp. 33–39. [Online]. http://dx.doi.org/10.1109/52.854066

[17] E. Carmel, "Time-to-completion in software package startups," in *Proceedings 27th Hawaii International Conference on System Sciences (HICSS)*. IEEE, 1994, pp. 498–507. [Online]. http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=323468

[18] O.P. Hilmola, P. Helo, and L. Ojala, "The value of product development lead time in software startup," *System Dynamics Review*, Vol. 19, No. 1, 2003, pp. 75–82. [Online]. http://doi.wiley.com/10.1002/sdr.255

[19] G. Coleman and R.V. O'Connor, "An investigation into software development process formation in software start-ups," *Journal of Enterprise Information Management*, Vol. 21, No. 6, 2008, pp. 633–648. [Online]. http://www.emeraldinsight.com/10.1108/17410390810911221

[20] 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.

[21] I.C. Macmillan, L. Zemann, and P. Subbanarasimha, "Criteria distinguishing successful from unsuccessful ventures in the venture screening process," *Journal of Business Venturing*, Vol. 2, No. 2, 1987, pp. 123–137.

[22] C. Giardino, X. Wang, and P. Abrahamsson, "Why early-stage software startups fail: A behavioral framework," in *Proceedings 5th International Conference on Software Business (ICSOB)*. Paphos, Cyprus: Springer, 2014, pp. 27–41. [Online]. http://link.springer.com/10.1007/978-3-319-08738-2{_}3

[23] Z. Block and I.C. MacMillan, "Milestones for successful venture planning," *Harvard Business Review*, Vol. 63, No. 5, 1985, pp. 184–196.

[24] E. Klotins, M. Unterkalmsteiner, and T. Gorschek, "Software Engineering in Start-up Companies: an Exploratory Study of 88 Startups," *Empirical Software Engineering*, 2016, in Submission.

[25] A. Yau and C. Murphy, "Is a Rigorous Agile Methodology the Best Development Strategy for Small Scale Tech Startups?" Tech-

nical Report MS-CIS-13-01, 2013. [Online]. http://repository.upenn.edu/cis_reports/980

[26] E. Klotins, M. Unterkalmsteiner, and T. Gorschek, "Software engineering practices in start-up companies: A mapping study," in *6th International Conference on Software Business*. Springer, 2015, pp. 245–257.

[27] K. Petersen and C. Wohlin, "Context in Industrial Software Engineering Research," in *Proceedings 3rd International Symposium on Empirical Software Engineering and Measurement (ESEM)*. Orlando, USA: IEEE, 2009, pp. 401–404.

[28] P. Clarke and R.V. O'Connor, "The situational factors that affect the software development process: Towards a comprehensive reference framework," *Information and Software Technology*, Vol. 54, No. 5, 2012, pp. 433–447. [Online]. http://www.sciencedirect.com/science/article/pii/S0950584911002369

[29] T. Dybå, D.I. Sjøberg, and D.S. Cruzes, "What Works for Whom, Where, When, and Why?: On the Role of Context in Empirical Software Engineering," in *Proceedings International Symposium on Empirical Software Engineering and Measurement (ESEM)*. Lund, Sweden: ACM, 2012, pp. 19–28. [Online]. http://doi.acm.org/10.1145/2372251.2372256

[30] D. Kirk and S.G. MacDonell, "Investigating a Conceptual Construct for Software Context," in *Proceedings 18th International Conference on Evaluation and Assessment in Software Engineering (EASE)*. London, UK: ACM, 2014, pp. 27:1–27:10. [Online]. http://doi.acm.org/10.1145/2601248.2601263

[31] D. Kirk and S. MacDonell, "Categorising Software Contexts," in *Proceedings 2014 Americas Conference on Information Systems (AMCIS)*. Savannah, USA: AIS Electronic Library, 2014. [Online]. http://aisel.aisnet.org/amcis2014/Posters/ITProjectManagement/8

[32] T. Feng, L. Sun, C. Zhu, and A.S. Sohal, "Customer orientation for decreasing time-to-market of new products IT implementation as a complementary asset," *Industrial Marketing Management*, Vol. 41, No. 6, 2012, pp. 929–939.

[33] 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. [Online]. http://www.sciencedirect.com/science/article/pii/S0164121213000022

[34] C. Fernández-Sánchez, J. Garbajosa, and A. Yagüe, "A framework to aid in decision making for technical debt management," in *Proceedings 7th International Workshop on Managing Technical Debt (MTD)*. Bremen, Germany: IEEE, 2015, pp. 69–76.

[35] Z. Li, P. Avgeriou, and P. Liang, "A systematic mapping study on technical debt and its management," *Journal of Systems and Software*, Vol. 101, 2015, pp. 193–220. [Online]. http://www.sciencedirect.com/science/article/pii/S0164121214002854

[36] E. Lim, N. Taksande, and C. Seaman, "A balancing act: What software practitioners have to say about technical debt," *IEEE Software*, Vol. 29, No. 6, 2012, pp. 22–27.

[37] F. Shull, D. Falessi, C. Seaman, M. Diep, and L. Layman, "Technical Debt: Showing the Way for Better Transfer of Empirical Results," in *Perspectives on the Future of Software Engineering*. Springer, 2013, pp. 179–190.

[38] F. Johne and P.A. Snelson, "Success factors in product innovation: A selective review of the literature," *Journal of Product Innovation Management*, Vol. 5, No. 2, 1988, pp. 114–128.

[39] D. Lippoldt and P. Stryszowski, "Innovation in the software sector," Organisation for Economic Co-operation and Development, Tech. Rep., 2009.

[40] H. Edison, N. bin Ali, and R. Torkar, "Towards innovation measurement in the software industry," *Journal of Systems and Software*, Vol. 86, No. 5, 2013, pp. 1390–1407. [Online]. http://www.sciencedirect.com/science/article/pii/S0164121213000058

[41] W. Eversheim, *Innovation Management for Technical Products: Systematic and Integrated Product Development and Production Planning*. Springer Science & Business Media, 2008.

[42] M.M. Crossan and M. Apaydin, "A multi-dimensional framework of organizational innovation: A systematic review of the literature," *Journal of Management Studies*, Vol. 47, No. 6, 2009, pp. 1154–1191. [Online]. http://doi.wiley.com/10.1111/j.1467-6486.2009.00880.x

[43] R. Balachandra and J. Friar, "Factors for success in R/D projects and new product innovation: a contextual framework," *IEEE Transactions on Engineering Management*, Vol. 44, No. 3, 1997, pp. 276–287.

[44] R.G. Cooper, "From experience – the invisible success factors in product innovation," *Journal of Product Innovation Management*, Vol. 16, No. 2, 1999, pp. 115–133.

[45] "Software engineering – software product quality requirements and evaluation (SQuaRE) – guide to SQuaRE – ISO/IEC 25000:2005," In-

ternational Organization for Standardization, Tech. Rep., 2005.

[46] A. Yagüe, J. Garbajosa, J. Pérez, and J. Díaz, "Analyzing Software Product Innovation Assessment by Using a Systematic Literature Review," in *Proceedings 47th Hawaii International Conference on System Sciences (HICSS)*. Waikoloa, USA: IEEE, 2014, pp. 5049–5058.

[47] M. Pikkarainen, W. Codenie, N. Boucart, and J.A. Heredia Alvaro, Eds., *The Art of Software Innovation*. Berlin, Germany: Springer, 2011.

[48] H. Lichter, M. Schneider-Hufschmidt, and H. Züllighoven, "Prototyping in Industrial Software Projects–Bridging the Gap Between Theory and Practice," in *Proceedings 15th International Conference on Software Engineering (ICSE)*. Baltimore, USA: IEEE, 1993, pp. 221–229.

[49] M. Beaudouin-Lafon and W.E. Mackay, "Prototyping development and tools," *Handbook of Human-Computer Interaction*, 2002, pp. 1006–1031.

[50] I. Sommerville, *Software Engineering*, 9th ed. Boston: Pearson, 2010.

[51] T. Brown, *Change by Design: How Design Thinking Transforms Organizations and Inspires Innovation*. New York: HarperBusiness, 2009.

[52] A. Efeoğlu, C. Møller, and M. Sérié, "Solution Prototyping with Design Thinking – Social Media for SAP Store: A Case Study," in *Proceedings European Design Science Symposium (EDSS)*. Dublin, Ireland: Springer, 2013, pp. 99–110.

[53] P. Newman, M.A. Ferrario, W. Simm, S. Forshaw, A. Friday, and J. Whittle, "The role of design thinking and physical prototyping in social software engineering," *37th IEEE International Conference on Software Engineering*, 2015.

[54] C. Grevet and E. Gilbert, "Piggyback prototyping: Using existing, large-scale social computing systems to prototype new ones," in *Proceedings 33rd Annual ACM Conference on Human Factors in Computing Systems*. Seoul, Korea: ACM, 2015, pp. 4047–4056. [Online]. http://doi.acm.org/10.1145/2702123.2702395

[55] A. Nguyen Duc and P. Abrahamsson, "Minimum viable product or multiple facet product? The Role of MVP in software startups," in *Proceedings 17th International XP Conference*. Edinburgh, UK: Springer, 2016.

[56] T. Raz and E. Michael, "Use and benefits of tools for project risk management," *International Journal of Project Management*, Vol. 19, No. 1, 2001, pp. 9–17. [Online]. http://www.sciencedirect.com/science/article/pii/S0263786399000368

[57] L. Cocco, K. Mannaro, G. Concas, and M. Marchesi, "Simulating Kanban and Scrum vs. Waterfall with System Dynamics," in *Proceedings 12th Internation XP Conference (XP)*. Madrid, Spain: Springer, 2011, pp. 117–131. [Online]. http://link.springer.com/10.1007/978-3-642-20677-1_9

[58] G. Concas, M.I. Lunesu, M. Marchesi, and H. Zhang, "Simulation of software maintenance process, with and without a work-in-process limit," *Journal of Software: Evolution and Process*, Vol. 25, No. 12, 2013, pp. 1225–1248. [Online]. http://onlinelibrary.wiley.com/doi/10.1002/smr.1599/abstract

[59] H. Edison, D. Khanna, S.S. Bajwa, V. Brancaleoni, and L. Bellettati, "Towards a Software Tool Portal to Support Startup Process," in *Proceedings 1st International Workshop on Software Startups*. Bolzano, Italy: Springer, 2015, pp. 577–583. [Online]. http://link.springer.com/chapter/10.1007/978-3-319-26844-6_43

[60] J. Zettel, F. Maurer, J. Münch, and L. Wong, "LIPE: a lightweight process for e-business startup companies based on extreme programming," in *Proceedings 3rd International Conference on Product-Focused Software Process Improvement (PROFES)*. Kaiserslautern, Germany: Springer, 2001, pp. 255–270.

[61] M.D. Kelly, "Lessons Learned from Software Testing at Startups," in *EuroStar-Software Testing Conference*, Amsterdam, The Netherlands, 2012.

[62] R.K. Yin, *Case Study Research: Design and Methods*, 3rd ed. Sage Publications, 2003.

[63] *Ergonomics of Human-system Interaction: Part 210: Human-centred Design for Interactive Systems*, ISO Std., 2010.

[64] J. Füller, R. Schroll, and E. von Hippel, "User generated brands and their contribution to the diffusion of user innovations," *Research Policy*, Vol. 42, No. 6–7, 2013, pp. 1197–1209. [Online]. http://www.sciencedirect.com/science/article/pii/S004873331300053X

[65] L. Hokkanen and K. Väänänen-Vainio-Mattila, "UX work in startups: current practices and future needs," in *Proceedings 16th International XP Conference (XP)*. Helsinki, Finland: Springer, 2015, pp. 81–92. [Online]. http://link.springer.com/chapter/10.1007/978-3-319-18612-2_7

[66] B. May, "Applying Lean Startup: An Experience Report – Lean & Lean UX by a UX Veteran: Lessons Learned in Creating & Launching a Complex Consumer App," in *Proceedings Agile Conference (AGILE)*. Dallas, USA: IEEE, 2012, pp. 141–147.

[67] M. Taipale, "Huitale–A Story of a Finnish Lean Startup," in *Proceedings 1st International Conference on Lean Enterprise and Software Systems (LESS)*. Helsinki, Finland: Springer, 2010, pp. 111–114. [Online]. http://www.springerlink.com/index/w48n06l04712621p.pdf

[68] L. Hokkanen, K. Kuusinen, and K. Väänänen, "Early Product Design in Startups: Towards a UX Strategy," in *Proceedings 16th International Conference on Product-Focused Software Process Improvement (PROFES)*. Bolzano-Bozen, Italy: Springer, 2015, pp. 217–224. [Online]. http://link.springer.com/chapter/10.1007/978-3-319-26844-6_16

[69] L. Hokkanen and M. Leppänen, "Three Patterns for User Involvement in Startups," in *Proceedings 20th European Conference on Pattern Languages of Programs (EuroPLoP)*. Kloster Irsee, Germany: ACM, 2015, pp. 51:1–51:8. [Online]. http://doi.acm.org/10.1145/2855321.2855373

[70] L. Hokkanen, K. Kuusinen, and K. Väänänen, "Minimum viable user experience: A framework for supporting product design in startups," in *Proceedings 17th International XP Conference (XP)*. Edinburgh, Scotland: Springer, 2016, in press.

[71] J. Nazar, "14 Famous Business Pivots," *Forbes*, 2013. [Online]. http://www.forbes.com/sites/jasonnazar/2013/10/08/14-famous-business-pivots/

[72] J. Bosch, V.D. Veen, and J. Salvador, "Pivots and Architectural Decisions: Two Sides of the Same Medal?" in *Chalmers Publication Library (CPL)*, 2013, pp. 310–317. [Online]. http://publications.lib.chalmers.se/publication/189719-pivots-and-architectural-decisions-two-sides-of-the-same-medal

[73] H. Terho, S. Suonsyrjä, A. Karisalo, and T. Mikkonen, *Ways to Cross the Rubicon: Pivoting in Software Startups*. Cham: Springer International Publishing, 2015, pp. 555–568. [Online]. http://dx.doi.org/10.1007/978-3-319-26844-6_41

[74] S. Shahid Bajwa, X. Wang, A. Nguven Duc, and P. Abrahamsson, "How do software startups pivot? empirical results from a multiple case study," in *7th International Conference on Software Business (ICSOB 2016)*, Ljubljana, Slovenia, 2016, pp. 169–176.

[75] H. Löfsten and P. Lindelöf, "Science Parks and the growth of new technology-based firms–academic-industry links, innovation and markets," *Research Policy*, Vol. 31, No. 6, 2002, pp. 859–876. [Online]. http://www.sciencedirect.com/science/article/pii/S0048733301001536

[76] J.S. Gans and S. Stern, "The product market and the market for "ideas": commercialization strategies for technology entrepreneurs," *Research Policy*, Vol. 32, No. 2, 2003, pp. 333–350. [Online]. http://www.sciencedirect.com/science/article/pii/S0048733302001038

[77] A. Brem and K.I. Voigt, "Integration of market pull and technology push in the corporate front end and innovation management – Insights from the German software industry," *Technovation*, Vol. 29, No. 5, 2009, pp. 351–367. [Online]. http://www.sciencedirect.com/science/article/pii/S0166497208000898

[78] A. Maurya, *Running Lean: Iterate from Plan A to a Plan That Works*. O'Reilly Media, Inc., 2012.

[79] K. Klyver and M.T. Schenkel, "From Resource Access to Use: Exploring the Impact of Resource Combinations on Nascent Entrepreneurship," *Journal of Small Business Management*, Vol. 51, No. 4, 2013, pp. 539–556. [Online]. http://onlinelibrary.wiley.com/doi/10.1111/jsbm.12030/abstract

[80] J. Levie and B.B. Lichtenstein, "A Terminal Assessment of Stages Theory: Introducing a Dynamic States Approach to Entrepreneurship," *Entrepreneurship Theory and Practice*, Vol. 34, No. 2, 2010, pp. 317–350. [Online]. http://onlinelibrary.wiley.com/doi/10.1111/j.1540-6520.2010.00377.x/abstract

[81] F. Giones and F. Miralles, "Strategic Signaling in Dynamic Technology Markets: Lessons From Three IT Startups in Spain," *Global Business and Organizational Excellence*, Vol. 34, No. 6, 2015, pp. 42–50. [Online]. http://onlinelibrary.wiley.com/doi/10.1002/joe.21634/abstract

[82] B. Clarysse, J. Bruneel, and M. Wright, "Explaining growth paths of young technology-based firms: structuring resource portfolios in different competitive environments," *Strategic Entrepreneurship Journal*, Vol. 5, No. 2, 2011, pp. 137–157. [Online]. http://onlinelibrary.wiley.com/doi/10.1002/sej.111/abstract

[83] S.L. Newbert and E.T. Tornikoski, "Supporter networks and network growth: a contingency model of organizational emergence," *Small Business Economics*, Vol. 39, No. 1, 2010, pp. 141–159. [Online]. http://link.springer.com/article/10.1007/s11187-010-9300-9

[84] T. Semrau and S. Sigmund, "Networking Ability and the Financial Performance of New Ventures: A Mediation Analysis among Younger and More Mature Firms," *Strategic Entrepreneurship Journal*, Vol. 6, No. 4, 2012, pp. 335–354. [Online]. http://onlinelibrary.wiley.com/doi/10.1002/sej.1146/abstract

[85] P. Witt, "Entrepreneurs' networks and the success of start-ups," *Entrepreneurship & Regional Development*, Vol. 16, No. 5, 2004, pp. 391–412. [Online]. http://dx.doi.org/10.1080/0898562042000188423

[86] K.M. Eisenhardt and J.A. Martin, "Dynamic capabilities: what are they?" *Strategic Management Journal*, Vol. 21, No. 10-11, 2000, pp. 1105–1121. [Online]. http://onlinelibrary.wiley.com/doi/10.1002/1097-0266(200010/11)21:10/11<1105::AID-SMJ133>3.0.CO;2-E/abstract

[87] D.J. Teece, G. Pisano, and A. Shuen, "Dynamic capabilities and strategic management," *Strategic Management Journal*, Vol. 18, No. 7, 1997, pp. 509–533. [Online]. http://onlinelibrary.wiley.com/doi/10.1002/(SICI)1097-0266(199708)18:7<509::AID-SMJ882>3.0.CO;2-Z/abstract

[88] S.L. Newbert, S. Gopalakrishnan, and B.A. Kirchhoff, "Looking beyond resources: Exploring the importance of entrepreneurship to firm-level competitive advantage in technologically intensive industries," *Technovation*, Vol. 28, No. 1–2, 2008, pp. 6–19. [Online]. http://www.sciencedirect.com/science/article/pii/S0166497207000910

[89] B.B. Lichtenstein, K.J. Dooley, and G.T. Lumpkin, "Measuring emergence in the dynamics of new venture creation," *Journal of Business Venturing*, Vol. 21, No. 2, 2006, pp. 153–175. [Online]. http://www.sciencedirect.com/science/article/pii/S0883902605000376

[90] C.G. Brush, T.S. Manolova, and L.F. Edelman, "Properties of emerging organizations: An empirical test," *Journal of Business Venturing*, Vol. 23, No. 5, 2008, pp. 547–566. [Online]. http://www.sciencedirect.com/science/article/pii/S0883902607000602

[91] J. Katz and W.B. Gartner, "Properties of Emerging Organizations," *Academy of Management Review*, Vol. 13, No. 3, 1988, pp. 429–441. [Online]. http://amr.aom.org/content/13/3/429

[92] B. Honig and T. Karlsson, "Institutional forces and the written business plan," *Journal of Management*, Vol. 30, No. 1, 2004, pp. 29–48. [Online]. http://jom.sagepub.com/content/30/1/29

[93] D. Kirsh, B. Goldfarb, and A. Gera, "Firm or substance: the role of business plans in venture capital decision making process," *Strategic Management Journal*, No. 30, 2009, pp. 487–515.

[94] T. Karlsson and B. Honig, "Judging a business by its cover: An institutional perspective on new ventures and the business plan," *Journal of Business Venturing*, Vol. 24, No. 1, 2009, pp. 27–45. [Online]. http://www.sciencedirect.com/science/article/pii/S0883902607000791

[95] M. König, G. Baltes, and B. Katzy, "On the role of value-network strength as an indicator of technology-based venture's survival and growth: Increasing innovation system efficiency by leveraging transaction relations to prioritize venture support," in *Proceedings International Conference on Engineering, Technology and Innovation/ International Technology Management Conference (ICE/ITMC)*. IEEE, 2015, pp. 1–9. [Online]. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6904149

[96] K. Dovan, "Reliability in content analysis: Some common misconceptions and recommendations," *Human Communication Research*, Vol. 30, No. 3, 1998, pp. 411–433.

[97] S. Elo and H. Kyngäs, "The qualitative content analysis process," *Journal of Advanced Nursing*, Vol. 62, No. 1, 2008, pp. 107–115. [Online]. http://onlinelibrary.wiley.com/doi/10.1111/j.1365-2648.2007.04569.x/abstract

[98] M. König, C. Ungerer, R. Büchele, and G. Baltes, "Agreement on the Venture's Reality Presented in Business Plans," in *Proceedings 22nd International Conference on Engineering, Technology and Innovation (ICE)*. Trondheim, Norway: IEEE, 2016.

[99] C. Ungerer, M. König, F. Giones, and G. Baltes, "Measuring Venture Emergence and Survival by Analyzing Transaction Relations in Business Plans," in *Proceedings 22nd International Conference on Engineering, Technology and Innovation (ICE)*. Trondheim, Norway: IEEE, 2016.

[100] S. Marlow, "Human resource management in smaller firms: A contradiction in terms?" *Human Resource Management Re-*

view, Vol. 16, No. 4, 2006, pp. 467–477. [Online]. http://www.sciencedirect.com/science/article/pii/S1053482206000684

[101] S. Jack, J. Hyman, and F. Osborne, "Small entrepreneurial ventures culture, change and the impact on HRM: A critical review," *Human Resource Management Review*, Vol. 16, No. 4, 2006, pp. 456–466. [Online]. http://www.sciencedirect.com/science/article/pii/S1053482206000672

[102] J. Bosch, H.H. Olsson, J. Björk, and J. Ljungblad, "The Early Stage Software Startup Development Model: A Framework for Operationalizing Lean Principles in Software Startups," in *Proceedings 4th International Conference on Lean Enterprise Software and Systems (LESS)*. Galway, Ireland: Springer, 2013, pp. 1–15.

[103] T. Dingsøyr and Y. Lindsjørn, "Team performance in agile development teams: Findings from 18 focus groups," in *Proceedings 14th International Conference on Agile Software Development*, Vienna, Austria, 2013, pp. 46–60. [Online]. http://link.springer.com/10.1007/978-3-642-38314-4

[104] J.R. Katzenbach and D.K. Smith, "The discipline of teams." *Harvard Business Review*, Vol. 71, No. 2, 1993, pp. 111–120.

[105] O. Oechslein and A. Tumasjan, "Examining trust within the team in it startup companies–an empirical study in the people's republic of china," in *Proceedings 45th Hawaii International Conference on System Science (HICSS)*, Maui, USA, 2012, pp. 5102–5111. [Online]. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6149511

[106] J. Kamm, J. Shuman, J. Seeger, and A. Nurick, "Entrepreneurial teams in new venture creation: A research agenda," *Entrepreneurship Theory and Practice*, Vol. 14, No. 4, 1990, pp. 7–17.

[107] D. Francis and W. Sandberg, "Friendship within entrepreneurial teams and its association with team and venture performance," *Entrepreneurship: Theory and Practice*, Vol. 25, No. 2, 2000, pp. 5–21. [Online]. http://go.galegroup.com/ps/i.do?id=GALE%7CA74524630&sid=googleScholar&v=2.1&it=r&linkaccess=fulltext&issn=10422587&p=AONE&sw=w

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

[109] M.D. Ensley, K.M. Hmieleski, and C.L. Pearce, "The importance of vertical and shared leadership within new venture top management teams: Implications for the performance of startups," *The Leadership Quarterly*, Vol. 17, No. 3, 2006, pp. 217–231. [Online]. http://www.sciencedirect.com/science/article/pii/S1048984306000051

[110] R.G. Cooper, "Perspective: The innovation dilemma: How to innovate when the market is mature," *Journal of Product Innovation Management*, Vol. 28, No. SUPPL. 1, 2011, pp. 2–27.

[111] C.K. Bart, "New venture units: use them wisely to manage innovation," *Sloan Management Review*, Vol. 29, No. 4, 1988, pp. 35–43.

[112] S.A. Hill and J. Birkinshaw, "Ambidexterity and survival in corporate venture units," *Journal of Management*, Vol. 40, No. 7, 2014, pp. 1899–1931. [Online]. http://jom.sagepub.com/cgi/doi/10.1177/0149206312445925

[113] J.J.P. Jansen, M.P. Tempelaar, F.A.J. van den Bosch, and H.W. Volberda, "Structural differentiation and ambidexterity: The mediating role of integration mechanisms," *Organization Science*, Vol. 20, No. 4, 2009, pp. 797–811. [Online]. http://pubsonline.informs.org/doi/abs/10.1287/orsc.1080.0415

[114] R. Nanda and M. Rhodes-Kropf, "Investment cycles and startup innovation," *Journal of Financial Economics*, Vol. 110, No. 2, 2013, pp. 403–418.

[115] D. Lavie, U. Stettner, and M.L. Tushman, "Exploration and exploitation within and across organizations," *The Academy of Management Annals*, Vol. 4, No. 1, 2010, pp. 109–155. [Online]. http://www.tandfonline.com/doi/abs/10.1080/19416521003691287

[116] J. Lerner, "Corporate venturing," *Harvard Business Review*, Vol. 91, No. 10, 2013, pp. 86–94.

[117] C.A. O'Reilly and M.L. Tushman, "Organizational ambidexterity: Past, present, and future," *Academy of Management Perspectives*, Vol. 27, No. 4, 2013, pp. 324–338. [Online]. http://amp.aom.org/cgi/doi/10.5465/amp.2013.0025

[118] D.A. Garvin and L.C. Levesque, "Meeting the challenge of corporate entrepreneurship." *Harvard business review*, Vol. 84, No. 10, 2006, pp. 102–12, 150. [Online]. http://www.ncbi.nlm.nih.gov/pubmed/17040043

[119] H. Edison, X. Wang, and P. Abrahamsson, "Lean startup," in *Scientific Workshop Proceedings of the XP conference*. Helsinki, Finland: ACM, 2015, pp. 1–7. [Online]. http://dl.acm.org/citation.cfm?doid=2764979.2764981

[120] C.J. Chen, "Technology commercialization, incubator and venture capital, and new venture performance," *Journal of Business Research*, Vol. 62, No. 1, 2009, pp. 93–103.

[121] S. Coleman, C. Cotei, and J. Farhat, "A resource-based view of new firm survival: new perspectives on the role of industry and exit route," *Journal of Developmental Entrepreneurship*, Vol. 18, No. 01, 2013, pp. 1–25. [Online]. http://www.worldscientific.com/doi/abs/10.1142/S1084946713500027

[122] G.G. Dess, R.D. Ireland, S.A. Zahra, S.W. Floyd, J.J. Janney, and P.J. Lane, "Emerging issues in corporate entrepreneurship," *Journal of Management*, Vol. 29, No. 3, 2003, pp. 351–378. [Online]. http://jom.sagepub.com/cgi/doi/10.1016/S0149-2063{_}03{_}00015-1

[123] A. Croll and B. Yoskovitz, *Lean Analytics: Use Data to Build a Better Startup Faster*, 1st ed. Sebastopol, USA: O'Reilly Media, 2013.

[124] P. Tyrväinen, M. Saarikallio, T. Aho, T. Lehtonen, and R. Paukeri, "Metrics Framework for Cycle-Time Reduction in Software Value Creation," in *Proceedings 10th International Conference on Software Engineering Advances (ICSEA)*. Barcelona, Spain: IARIA, 2015. [Online]. https://jyx.jyu.fi/dspace/handle/123456789/47980

[125] T.R. Eisenmann, E. Ries, and S. Dillard, "Hypothesis-driven entrepreneurship: The lean startup," *Harvard Business School*, 2012.

[126] J. Järvinen, T. Huomo, T. Mikkonen, and P. Tyrväinen, "From Agile Software Development to Mercury Business," in *Proceedings 5th International Conference on Software Business (ICSOB)*. Paphos, Cyprus: Springer, 2014, pp. 58–71. [Online]. http://link.springer.com/10.1007/978-3-319-08738-2_5

[127] H. Edison, "A conceptual framework of lean startup enabled internal corporate venture," in *Proceedings 1st International Workshop on Software Startups*. Bolzano-Bozen, Italy: Springer, 2015, pp. 607–613. [Online]. http://link.springer.com/10.1007/978-3-319-26844-6_46

[128] J. Märijärvi, L. Hokkanen, M. Komssi, H. Kiljander, Y. Xu, M. Raatikainen, P. Seppänen, J. Heininen, M. Koivulahti-Ojala, M. Helenius, and J. Järvinen, *The Cookbook for Successful Internal Startups*. DIGILE and N4S, 2016.

[129] S. Meskendahl, "The influence of business strategy on project portfolio management and its success – a conceptual framework," *International Journal of Project Management*, Vol. 28, No. 8, 2010, pp. 807–817.

[130] B.S. Blichfeldt and P. Eskerod, "Project portfolio management – there's more to it than what management enacts," *International Journal of Project Management*, Vol. 26, No. 4, 2008, pp. 357–365.

[131] J.R. Turner, *The handbook of project-based management*. McGraw-Hill, 2014.

[132] H.W. Chesbrough, *Open innovation: The new imperative for creating and profiting from technology*. Harvard Business Press, 2006.

[133] T. Hatzakis, M. Lycett, and A. Serrano, "A programme management approach for ensuring curriculum coherence in is (higher) education," *European Journal of Information Systems*, Vol. 16, No. 5, 2007, pp. 643–657.

[134] B.D. Reyck, Y. Grushka-Cockayne, M. Lockett, S.R. Calderini, M. Moura, and A. Sloper, "The impact of project portfolio management on information technology projects," *International Journal of Project Management*, Vol. 23, No. 7, 2005, pp. 524–537.

[135] B. Kersten and C. Verhoef, "IT portfolio management: A banker's perspective on IT," *Cutter IT Journal*, 2003. [Online]. http://www.few.vu.nl/~x/bp/bp.pdf

[136] R. LeFave, B. Branch, C. Brown, and B. Wixom, "How sprint nextel reconfigured it resources for results," *MIS Quarterly*, 2008. [Online]. http://misqe.org/ojs2/index.php/misqe/article/view/213

[137] J. Krebs, *Agile portfolio management*, 1st ed. Microsoft Press, 2008.

[138] Y.Z. Anbardan and M. Raeyat, "Open innovation: Creating value through co-creation," in *Proceedings 7th World Conference on Mass Customization, Personalization, and Co-Creation (MCPC 2014)*. Aalborg, Denmark: Springer, 2014, pp. 437–447.

[139] W. Vanhaverbeke and M. Cloodt, *Open Innovation: Researching a New Paradigm*. Oxford University Press, 2008, ch. Open innovation in value networks, pp. 258–284.

[140] V. Van de Vrande, J.P. De Jong, W. Vanhaverbeke, and M. De Rochemont, "Open innovation in smes: Trends, motives and management challenges," *Technovation*, Vol. 29, No. 6, 2009, pp. 423–437.

[141] J. West and S. Gallagher, "Challenges of open innovation: the paradox of firm investment in open-source software," *R&D Management*, Vol. 36, No. 3, 2006, pp. 319–331. [On-

line]. http://dx.doi.org/10.1111/j.1467-9310.2006.00436.x

[142] H. Chesbrough and A. Crowther, "Beyond high tech: early adopters of open innovation in other industries," *R&d Management*, Vol. 36, No. 3, 2006, pp. 229–236. [Online]. http://onlinelibrary.wiley.com/doi/10.1111/j.1467-9310.2006.00428.x/full

[143] G. Hu, L. Wang, S. Fetch, and B. Bidanda, "A multi-objective model for project portfolio selection to implement lean and Six Sigma concepts," *International Journal of Production Research*, Vol. 46, No. 23, 2008, pp. 6611–6625. [Online]. http://dx.doi.org/10.1080/00207540802230363

[144] M.A. Cusumano and K. Nobeoka, *Thinking Beyond Lean: How Multi-project Management is Transforming Product Development at Toyota and Other Companies*. Simon and Schuster, 1998.

[145] F. Kon, D. Cukier, C. Melo, O. Hazzan, and H. Yuklea, "A conceptual framework for software startup ecosystems: the case of Israel," Department of Computer Science, University of São Paulo, Tech. Rep. RT-MAC-2015-01„ 2015. [Online]. http://www.ime.usp.br/~kon/papers/SoftwareStartupsConceptualFramework-TR.pdf

[146] M.C. Fonseca, "O ecossistema de startups de software da cidade de são paulo," Master's thesis, University of São Paulo, 2016.

[147] D. Cukier, F. Kon, and L.S. Thomas, "Software startup ecosystems evolution: The New York City case study," in *Proceedings 2nd International Workshop on Software Startups*. Trondheim, Norway: IEEE, 2016.

[148] D. Cukier, F. Kon, and N. Krueger, "Designing a maturity model for software startup ecosystems," in *Proceedings 1st International Workshop on Software Startups*. Bolzano, Italy: Springer, 2015, pp. 600–606.

[149] B.L. Herrmann, J.F. Gauthier, D. Holtschke, R. Berman, and M. Marmer, "The global startup ecosystem ranking 2015," Tech. Rep. August, 2015.

[150] A. Frenkel and S. Maital, *Mapping National Innovation Ecosystems: Foundations for Policy Consensus*. London, UK: Edward Elgar Publishing, 2014.

[151] S. Jayshree and R. Ramraj, "Entrepreneurial ecosystem: Case study on the influence of environmental factors on entrepreneurial success," *European Journal of Business and Management*, Vol. 4, No. 16, 2012, pp. 95–102.

[152] R. Sternberg, "Success factors of university-spin-offs: Regional government support programs versus regional environment," *Technovation*, Vol. 34, No. 3, 2014, pp. 137–148. [Online]. http://www.sciencedirect.com/science/article/pii/S0166497213001399

[153] M. Steinert and L.J. Leifer, "'Finding One's Way': Re-Discovering a Hunter - Gatherer Model based on Wayfaring," *International Journal of Engineering Education*, Vol. 28, No. 2, 2012, pp. 251–252.

[154] D.J. Snowden and M.E. Boone, "A leader's framework for decision making," *Harvard Business Review*, Vol. 85, No. 11, 2007, pp. 69–76.

[155] S.D. Sarasvathy, "Causation and effectuation: Toward a theoretical shift from economic inevitability to entrepreneurial contingency," *Academy of Management*, Vol. 26, No. 2, 2001, pp. 243–263. [Online]. http://www.jstor.org.proxy.lib.ul.ie/stable/info/259121

[156] P. Williams, "The Competent Boundary Spanner," *Public Administration*, Vol. 80, No. 1, 2002, pp. 103–124. [Online]. http://onlinelibrary.wiley.com/doi/10.1111/1467-9299.00296/abstract

[157] A. Nguyen Duc, P. Seppänen, and P.K. Abrahamsson, "Hunter-gatherer cycle: a conceptual model of the evolution of software startups," in *Proceedings 2015 International Conference on Software and System Process*. Tallin, Estonia: ACM, 2015, pp. 199–203.

[158] J. Pelrine, "On Understanding Software Agility: A Social Complexity Point Of View," *Emergence: Complexity & Organization*, Vol. 13, No. 1/2, 2011, pp. 26–37.

[159] J. Rikkila, P. Abrahamsson, and X. Wang, "The Implications of a Complexity Perspective for Software Engineering Practice and Research," *Journal of Computer Engineering & Information Technology*, 2012. [Online]. http://www.scitechnol.com/2324-9307/2324-9307-1-e103.pdf

[160] A. Johri, "Boundary spanning knowledge broker: An emerging role in global engineering firms," in *Proceedings 38th Annual Frontiers in Education Conference*. IEEE, 2008, pp. 7–12.

[161] E. Wenger, *Communities of Practice: Learning, Meaning, and Identity*. Cambridge University Press, 1998.

[162] J. Birkinshaw, G. Hamel, and M.J. Mol, "Management innovation," *Academy of management Review*, Vol. 33, No. 4, 2008, pp. 825–845. [Online]. http://amr.aom.org/content/33/4/825.short

[163] A.H.V.D. Ven and M.S. Poole, "Explaining Development and Change in Organizations," *Academy of Management Review*, Vol. 20, No. 3, 1995, pp. 510–540. [Online]. http://amr.aom.org/content/20/3/510

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

[165] J.W. Mullins and R. Komisar, *Getting to Plan B: Breaking Through to a Better Business Model*. Harvard Business Press, 2009.

[166] C. Nobel, "Teaching a 'Lean Startup' Strategy," *HBS Working Knowledge*, 2011. [Online]. http://hbswk.hbs.edu/item/teaching-a-lean-startup-strategy

[167] Y. Harb, C. Noteboom, and S. Sarnikar, "Evaluating Project Characteristics for Selecting the Best-fit Agile Software Development Methodology: A Teaching Case," *Journal of the Midwest Association for Information Systems (JMWAIS)*, Vol. 1, No. 1, 2015. [Online]. http://aisel.aisnet.org/jmwais/vol1/iss1/4

[168] D. Dennehy, L. Kasraian, O. O'Raghallaigh, and K. Conboy, "Product Market Fit Frameworks for Lean Product Development," in *Proceedings R&D Management Conference 2016 "From Science to Society: Innovation and Value Creation"*, Cambridge, UK, 2016.

[169] K. Conboy, "Agility from first principles: Reconstructing the concept of agility in information systems development," *Information Systems Research*, Vol. 20, No. 3, 2009, pp. 329–354. [Online]. http://pubsonline.informs.org/doi/abs/10.1287/isre.1090.0236

[170] M.B. Miles and A. Huberman, *Qualitative data analysis: An expanded sourcebook*. Thousand Oaks, US: Sage Publications, Inc, 1994.

[171] C. Wohlin, A. Aurum, L. Angelis, L. Phillips, Y. Dittrich, T. Gorschek, H. Grahn, K. Henningsson, S. Kagstrom, G. Low *et al.*, "The success factors powering industry-academia collaboration," *IEEE software*, Vol. 29, No. 2, 2012, p. 67.

[172] M. Unterkalmsteiner, T. Gorschek, A. Islam, C. Cheng, R. Permadi, and R. Feldt, "A conceptual framework for SPI evaluation," *Journal of Software: Evolution and Process*, Vol. 26, No. 2, 2014, pp. 251–279.

[173] B. Kitchenham, L. Madeyski, D. Budgen, J. Keung, P. Brereton, S. Charters, S. Gibbs, and A. Pohthong, "Robust Statistical Methods for Empirical Software Engineering," *Empirical Software Engineering*, 2016, pp. 1–52. [Online]. http://link.springer.com/article/10.1007/s10664-016-9437-5

[174] W. Hayes, "Research synthesis in software engineering: a case for meta-analysis," in *Proceedings 6th International Software Metrics Symposium*. Boca Raton, USA: IEEE, 1999, pp. 143–151.

[175] T. Gorschek, C. Wohlin, P. Carre, and S. Larsson, "A Model for Technology Transfer in Practice," *IEEE Software*, Vol. 23, No. 6, 2006, pp. 88–95.

[176] R. Wieringa, "Empirical research methods for technology validation: Scaling up to practice," *Journal of Systems and Software*, Vol. 95, 2014, pp. 19–31.

[177] P. Bourque and R.E. Fairley, Eds., *Guide to the Software Engineering Body of Knowledge*, 3rd ed. IEEE, 2014.

[178] E. Bjarnason, M. Unterkalmsteiner, E. Engström, and M. Borg, "A multi-case study of agile requirements engineering and using test cases as requirements," *Information and Software Technology*, Vol. 77, 2016, pp. 61–79.

[179] B. Boehm, "Value-based Software Engineering," *SIGSOFT Softw. Eng. Notes*, Vol. 28, No. 2, 2003, pp. 1–12.

[180] C.R.B.d. Souza, H. Sharp, J. Singer, L.T. Cheng, and G. Venolia, "Guest Editors' Introduction: Cooperative and Human Aspects of Software Engineering," *IEEE Software*, Vol. 26, No. 6, 2009, pp. 17–19.

[181] C.D. Mulrow, "Rationale for systematic reviews." *BMJ: British Medical Journal*, Vol. 309, No. 6954, 1994, pp. 597–599. [Online]. http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2541393/

[182] B.A. Kitchenham, T. Dybå, and M. Jørgensen, "Evidence-Based Software Engineering," in *Proceedings 26th International Conference on Software Engineering (ICSE)*. Edinburgh, UK: IEEE, 2004, pp. 273–281.

[183] R. Torkar, P. Minoves, and J. Garrigós, "Adopting free/libre/open source software practices, techniques and methods for industrial use," *Journal of the Association for Information Systems*, Vol. 12, No. 1, 2011, p. 88. [Online]. http://search.proquest.com/openview/beaf4af76e081e0d38c0c1c574218df2/1

[184] E. von Hippel, "Innovation by User Communities: Learning from Open-Source Software," *MIT Sloan Management Review*, Vol. 42, No. 4, 2001, pp. 82–82. [Online]. http://adaptknowledge.com/wp-content/uploads/rapidintake/PI_CL/media/InnArticle.pdf

[185] J. West, "How open is open enough?: Melding proprietary and open source platform strategies," *Research Policy*, Vol. 32, No. 7, 2003, pp.

1259–1285. [Online]. http://www.sciencedirect.com/science/article/pii/S0048733303000520

[186] 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. [Online]. http://www.sciencedirect.com/science/article/pii/S0950584908000256

[187] M. Broy, "Challenges in Automotive Software Engineering," in *Proceedings 28th International Conference on Software Engineering (ICSE)*. Shanghai, China: ACM, 2006, pp. 33–42. [Online]. http://doi.acm.org/10.1145/1134285.1134292

[188] S. Chandra, V.S. Sinha, S. Sinha, and K. Ratakonda, "Software Services: A Research Roadmap," in *Future of Software Engineering (FOSE)*. Hyderabad, India: ACM, 2014, pp. 40–54. [Online]. http://doi.acm.org/10.1145/2593882.2593892

[189] J. Cleland-Huang, O.C.Z. Gotel, J. Huffman Hayes, P. Mäder, and A. Zisman, "Software Traceability: Trends and Future Directions," in *Proceedings Future of Software Engineering*. Hyderabad, India: ACM, 2014, pp. 55–69. [Online]. http://doi.acm.org/10.1145/2593882.2593891

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 *Polish Academy of Sciences, Committee of Computer Science, Software Engineering Section*.

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** (see 1, 2, 3, 4 and 5) 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 and external reviewers.

English is the only accepted publication language. To submit an article please enter our online paper submission site.

Subsequent issues of the journal will appear continuously according to the reviewed and accepted submissions.

http://www.e-informatyka.pl/wiki/e-Informatica

e-Informatica

ISSN 1897-7979