Advances in Intelligent Systems and Computing 504

Lech Madeyski
Michał Śmiałek
Bogumiła Hnatkowska
Zbigniew Huzar *Editors*

Software Engineering: Challenges and Solutions

Results of the XVIII KKIO 2016 Software Engineering Conference 2016 held at September 15–17 2016 in Wroclaw, Poland

Springer

Advances in Intelligent Systems and Computing

Volume 504

More information about this series at http://www.springer.com/series/11156

Lech Madeyski · Michał Śmiałek
Bogumiła Hnatkowska · Zbigniew Huzar
Editors

Software Engineering: Challenges and Solutions

Results of the XVIII KKIO 2016 Software Engineering Conference 2016 held at September 15–17 2016 in Wroclaw, Poland

Springer

Editors

Lech Madeyski
Faculty of Computer Science
and Management
Wrocław University of Science
and Technology
Wrocław
Poland

Bogumiła Hnatkowska
Faculty of Computer Science
and Management
Wrocław University of Science
and Technology
Wrocław
Poland

Michał Śmiałek
Faculty of Electrical Engineering
Warsaw University of Technology
Warsaw
Poland

Zbigniew Huzar
Faculty of Computer Science
and Management
Wrocław University of Science
and Technology
Wrocław
Poland

ISSN 2194-5357 ISSN 2194-5365 (electronic)
Advances in Intelligent Systems and Computing
ISBN 978-3-319-43605-0 ISBN 978-3-319-43606-7 (eBook)
DOI 10.1007/978-3-319-43606-7

Library of Congress Control Number: 2016947021

Preface

Back in 1968, the first NATO Conference on Software Engineering has announced the "software crisis". Today we can see this as a "chronic affliction" of missed deadlines, extended budgets (poor productivity) and unsatisfied clients (poor quality). The cause of this affliction is the inherent complexity of software systems. Today, we can identify several major areas of research and practice that aim at overcoming this complexity and thus achieving an increase in productivity and quality of software in various application domains. First of the areas consists in increasing levels of abstraction for programming constructs. Recent approaches in this field include domain-specific languages, various requirements languages and domain modelling. Second of the areas consists in assuring a better quality of produced software. It includes various approaches to predict and eliminate defects by analysing code or by creating new testing methods. Third of the areas consists in optimising software development and operation cycles. This direction comprises such elements as continuous product delivery, frequent product releases, high end-user involvement and feedback, as well as tight integration of teams. The above three areas of software engineering are reflected in three parts of this volume.

The first part—Requirements and Domain Modelling—treats the problems concerning initial phases of software development, which are domain or business modelling, and requirements specification. The basic questions asked in this part are: "what requirements specification languages should be used?" and "how to represent requirements specifications in these languages to guarantee that they are complete and are equally understandable by all stakeholders?" The paper "Business Rule Patterns Catalog for Structural Business Rules basing on the OMG recommendation Semantics of Business Vocabulary and Business Rules" proposes a catalogue of patterns for structural business rules. The catalogue is intended to contain a limited number of patterns, which cover the most typical cases that can be found by a business analyst during the business rule identification process. The paper "Sketching Use-case Scenarios Based on Use-Case Goals and Patterns" is in some sense similar to the previous one but focuses on use cases. It describes and evaluates the idea of a catalogue containing use-case patterns as a basis for a semi-automatic approach to generate use-case scenarios, and a programming tool supporting this generation. The next two papers are complementary to one another. The first one, "Domain Modelling Based on Requirements Specification and Ontology", presents an approach to UML class diagram construction on the basis of a given domain ontology expressed using the SUMO language and a given user requirements model. The second one, "Semantic Validation of UML Class Diagrams with the Use of Domain Ontologies Expressed in OWL 2", considers how to check compliance of a given UML class diagram with a given domain ontology expressed in OWL. The last paper in this part, "RSL-DL: Representing Domain Knowledge for the Purpose of Code Generation", proposes a declarative specification language to define domain logic, being a new extension to the existing RSL language. The new language facilitates construction of precise requirements specifications and their further generation into code structured using the MVP architectural pattern.

In the next part—Quality Assurance—the main issues under consideration are prediction of defects, mutation testing and practical problems of measuring the quality of software. The paper "Assessment of the Software Defect Prediction Cost Effectiveness in an Industrial Project" deals with a real problem found in the industry: how the proposed method for predicting defects, assisted by an open source software measurement framework, affects the cost of software development. This preliminary investigation is encouraging, indicating the possibility of significant financial savings. The paper "Dynamic Stylometry for Defect Prediction" hypothesises that the indirect measure of software defects can be estimated by the measures defined for the software development process. The proposed model of defects prediction is evaluated experimentally on a group of computer science students. A new idea for mutation testing of Web Services is presented in the paper "Mutant generation for WSDL". A tool called Web Services Mutant Generator that supports mutation testing of Web Services and examples of its usage are presented. The paper "Improving Measurement Certainty by Using Calibration to Find Systematic Measurement" states a very practical question: how to reduce the uncertainty of measurement results obtained from instruments with an unknown measurement method? When answering this question, the authors postulate to replace standard statistical analysis by an appropriate calibration procedure. The last paper in this part also refers to practical issues. The aim of the paper "Performance comparison of CRM systems dedicated to reporting failures to IT department" is to compare efficiency of a client-side architecture system vs. a server-side architecture system, under the condition of their significant load. The obtained results confirm the importance of choosing the client-side architecture for small and medium-sized systems.

The third part of this volume—Software Process Improvement—discusses selected aspects and mechanisms conducive to improving quality of software development. The first paper, "Software Engineering Needs Agile Experimentation: a New Practice and Supporting Tool", meets the needs of collecting high-quality data during lightweight experimentation in real-world software development. It proposes and assesses a new practice called "agile experimentation", together with a dedicated tool supporting the practice. The next paper, "An Industrial Survey on Business Analysis Problems and Solutions", analyses a set of business analysis techniques used in selected industrial projects and their impact on the most often reported problems encountered by business analysts. The main outcome of the paper is a recommendation regarding ways in which these techniques should be applied. The paper "Beyond Software Architecture Knowledge Management Tools" is a reflection on architecture decision-making tools in isolation from a particular practical context. It postulates the need for an architecture knowledge management infrastructure integrated with popular tools that are used for software development.

The Rational Unified Process is a widely accepted formal software development methodology. It can be used as a pattern for assessment of process maturity in software development organisations. The paper "Model of RUP processes maturity assessment in IT organisations" presents the design of a system that analyses application of RUP-compliant processes in real organisations within actual projects. The last paper, "A collaborative approach to developing a part-time Ph.D. program in IT Architecture with industry cooperation", concerns the important problem of professional education at the Ph.D. level. A motivation model for establishing Ph. D. studies is proposed. Its leading idea is to make an agreement between academy and industry based on the SWOT analysis.

The above 15 papers constitute a selection from 59 submissions, which gives the total acceptance rate of less than 26 %. The editors would like to express their cordial thanks to all the authors for their significant contributions. We are very grateful to all reviewers for their excellent reviews and comments supporting the selection process and leveraging the quality of the selected papers.

Wrocław Lech Madeyski
Warsaw Michał Śmiałek
Wrocław Bogumiła Hnadkowska
Wrocław Zbigniew Huzar
May 2016

Reviewers

Zbigniew Banaszak, Warsaw University of Technology, Poland
Włodzimierz Bielecki, West Pomeranian University of Technology, Poland
Miklós Biró, Software Competence Center Hagenberg, Austria
Přemek Brada, University of West Bohemia, Czech Republic
David Budgen, Durham University, United Kingdom
Krzysztof Cetnarowicz, AGH University of Science and Technology, Poland
Zbigniew Czech, Silesia University of Technology, Poland
Włodzimierz Dąbrowski, Warsaw University of Technology, Poland
Mariusz Flasiński, Jagiellonian University, Poland
Jozef Goetz, University of La Verne, USA
Adam Grzech, Wroclaw University of Science and Technology, Poland
Janusz Górski, Gdańsk University of Technology, Poland
Piotr Habela, Polish-Japanese Institute of Information Technology, Poland
Bogumiła Hnatkowska, Wroclaw University of Science and Technology, Poland
Zbigniew Huzar, Wroclaw University of Science and Technology, Poland
Stanisław Jarząbek, Bialystok University of Technology, Poland
Hermann Kaindl, TU Wien, Austria
Barbara Kitchenham, Keele University, United Kingdom
Ludwik Kuźniarz, Blekinge Institute of Technology, Sweden
Piotr Kosiuczenko, Military University of Technology, Poland
Kevin Lano, King's College London, United Kingdom
Laszlo Lengyel, Budapest University of Technology and Economics, Hungary
Leszek Maciaszek, Wroclaw University of Economics, Poland
Lech Madeyski, Wroclaw University of Science and Technology, Poland
Marek Majchrzak, Wroclaw University of Science and Technology, Poland
Zygmunt Mazur, Wroclaw University of Science and Technology, Poland
Erika Nazaruka, Riga Technical University, Latvia
Ngoc-Thanh Nguyen, Wroclaw University of Science and Technology, Poland
Mirosław Ochodek, Poznań University of Technology, Poland
Cezary Orłowski, Wyższa Szkoła Bankowa Gdańsk
Aneta Poniszewska-Marańda, Lódź University of Technology, Poland

Krzysztof Sacha, Warsaw University of Technology, Poland
Klaus Schmid, University of Hildesheim, Germany
Michał Smiałek, Warsaw University of Technology, Poland
Miroslaw Staron, University of Gothenburg, Sweden
Andrzej Stasiak, Military University of Technology, Poland
Krzysztof Stencel, University of Warsaw, Poland
Jakub Swacha, University of Szczecin, Poland
Tomasz Szmuc, AGH University of Science and Technology, Poland
Lech Tuzinkiewicz, Wroclaw University of Science and Technology, Poland
Bartosz Walter, Poznań University of Technology, Poland
Krzysztof Wnuk, Blekinge Institute of Technology, Sweden
Andrzej Wąsowski, IT University of Copenhagen, Denmark
Janusz Zalewski, Florida Gulf Coast University, USA
Jaroslav Zendulka, Brno University of Technology, Czech Republic
Mel Ó Cinnéide, University College Dublin, Ireland

Additional Reviewers

Dubielewicz, Iwona
Majchrzak, Marek
Kuźniarz, Ludwik

Contents

Part I
Requirements and Domain Modeling

Business Rule Patterns Catalog for Structural Business Rules

Bogumiła Hnatkowska and Jose María Alvarez-Rodriguez

Abstract Business rules are substantial citizens of requirement world. Similarly to software requirements they can be specified with the use of patterns. Business rules patterns increase the quality of a specification assuring business rules representation consistency and helping in revealing missing or overlapping rules. The aim of the paper is to propose a catalog of business rules patterns for structural rules. Such rules extend the definition of the domain expressing mandatory constraints. The catalog is constructed with two main assumptions in mind: it should be flexible enough to cover most of the business rules from that category, and it should not put too strong constraints on its users. The catalog has been developed as a result of literature overview including investigation of existing pattern catalogs. The usefulness of proposed catalog is then checked by a case study when the business rules patterns are used to express business rules from existing sources.

Keywords Business rules • Structural business rules • Patterns • Catalog of patterns

B. Hnatkowska (✉)
Faculty of Computer Science and Management, Wrocław University
of Science and Technology, Wyb. Wyspiańskiego 27, 50-370 Wrocław, Poland
e-mail: Bogumila.Hnatkowska@pwr.edu.pl

J.M. Alvarez-Rodriguez
Faculty of Computer Science and Management, Department of Computer Science
and Engineering, Carlos III University of Madrid, Avd. de la Universidad 30,
Leganés, Spain
e-mail: josemaria.alvarez@uc3m.es

L. Madeyski et al. (eds.), *Software Engineering: Challenges and Solutions*,
Advances in Intelligent Systems and Computing 504,
DOI 10.1007/978-3-319-43606-7_1

1 Introduction

Business rules can be perceived as citizens of requirement world [1, 2] or, at least, statements that imply requirements [3]. Not every business rule is implemented in software, but if it is, it is strongly connected with functional requirements that must behave according to it [3].

Business rules, similarly to software requirements, can be described with the use of patterns or statement patterns. Application of patterns is of particular importance at the business level when business rules are formulated by business people in the language, which is commonly understood by all interested parties, i.e. natural language, or controlled natural language. Application of patterns can bring several potential benefits, e.g. [1, 4]:

- Better requirements—with more details, precision (including completeness), and clarity, and with less ambiguity.
- Better efficiency—you will be able to write requirements more quickly and with less effort.
- Better management—your requirements will be better structured.

Business Rule Pattern is here defined as an approach to specifying a particular type of business rules, which aims at better business rules quality, understood as uniformity of vocabulary, and grammar as well as better readability and analyzability of business rules.

The goal of this research is to propose a catalog of patterns specified in an implementation-independent language that is focused on structural (definitional) business rules. This kind of business rules is especially interesting as they represent invariants that always must be true, and prevent performing actions that violate this state.

The catalog was designed on the base of literature overview and verified against its completeness with the use of the freely available set of business rules [5]. The catalog was prepared with two major assumptions. The intention was to propose a narrow but representative set of rule patterns that will be sufficient to cover most of the possible rules from that group (Pareto rule). On the other side, the catalog should support the process of business rule specification without imposing too intense demands on the way it is created. To the knowledge of the author such a catalog for structural business rules does not exist, but one can find other examples of catalogs for structural rules or their subtypes.

Related works, especially those used for catalog construction, are described in Sect. 2. The method of catalog construction and validation together with the patter catalog itself is presented in Sect. 3. Section 4 concludes the paper and shows further works.

2 Related Work

2.1 Business Rules Definitions and Classifications

There are many definitions of a business rule notion. About 20 years ago business rules were defined as statements that define or constrain some aspect of the business [6]. Currently, the elements that define some aspect of the business (business vocabulary) are separated from those that influence the behavior of the business (business rules), e.g. [3, 7]. According to [3], "a business rule is a criterion used to guide operational business behavior, shape operational business judgments, or make operational business decisions". According to [7], a business rule is a "proposition that is a claim of obligation or of necessity", and "is under business jurisdiction".

The Business Rules Group (BRG) formulated so-called "Business Rules Manifesto", which contains a set of 10 articles about business rules. The most important, in the context of this research, are those stating that: business rules are a first-class citizens of the requirement world (1.1); they are not processes and not procedures, and should not be contained in none of these (2.2); are built on facts, and facts are built on concepts as expressed by terms (3.1); should be expressed declaratively in natural-languages understood by business people (4.1), in such a way that they can be validated for correctness by business people (5.2).

There exist many different classifications of business rules. One of the simplest is proposed in [7], which splits business rules into two categories: structural/definitional business rules ("claim of necessity"), and operative/behavioral business rules ("claim of obligation"). In the context of that paper, we are interested in the first group. "Necessity" means that something must be true, and it relates to the way the concepts are understood [7]. "Structural rules often, but not always, propose necessary characteristics of concept" [7]. Those rules are also able to express some demands about quantities of instances or mandatory constraints. Structural rules supplement definitions.

On the other side, operational business rules state what should or shouldn't happen in particular circumstances, but one can't guarantee it is true. These rules can be contravened or ignore by people, so it is why they have so-called "enforcement level" (e.g. strict—"if you violate the rule, you cannot escape the penalty").

The Semantics of Business Vocabulary and Rules (SBVR) uses another syntax to distinguish between structural and operational rules, either those based on SBVR Structured English ("It is necessary …"—structural rules, "It is obligatory …"—operative rule), or those based on RuleSpeak (" … always …"—structural rules, "… must …"—operational rules).

Barbara von Halle defines three categories of rules: derivations, action-enabler rules, and constraints split into two subcategories: mandatory constraints, and suggested guidelines [8]. We are mainly interested in derivations (divided into inference rules and calculations) and mandatory constraints, which are further subdivided into data integrity constraints, and business-oriented constraints (those imposed by the business on its own, based on business policies).

In [9] the following categories belonging to mandatory constraints are defined:

- Referential integrity constraints, e.g. It is impossible to delete a class if it has any student enrolled.
- Optionality constraints (at least one occurrence of an entity must be related in a specified way to an occurrence of another entity) e.g. Each contract must be composed of at least 1 line item.
- Cardinality constraints (no more than one or some other specified number of occurrences of an entity must be related in a specified way to an occurrence of another entity), e.g. Each contract must be assigned to only one party.
- An attribute is required to have value when an occurrence of an entity is created, e.g. Description requires an attribute of asset type.
- An occurrence of a super type must be an occurrence of one and only one subtype, e.g. a party must be either a person or an organization, but cannot be both.

Referential integrity constraints in our opinion are unnecessary, as the constraint can be expressed declaratively, e.g. each student must be enrolled in at least one class.

Business Rules Group in [6] defines three types of business rules: structural assertions, action assertions, and derivations. Action assertions specify constraints on the results that action can produce. Action assertions are divided into three types:

- Integrity constraint—something that must be true about entity, relationship, or attribute by definition; the rules prevent actions that would violate the constraint.
- Condition—defines a condition under which another rule is applied.
- Authorizations (business rules, what they are really)—only x can do y.

Integrity constraints can be understood as a subset of structural business rules while others subtypes belong to operative business rules according to SBVR classification.

Witt, in his book [10], proposes another classification and maps it to existing ones (e.g. [6, 7]). He introduces four types of business rules: definitional rules, data rules, activity rules, party rules. His definitional rules include many subtypes, e.g. formal definitions of particular terms (include derivations, but that name is not used), enumerations of mutually exclusive set of categories, rules describing the structure of complex concepts or algorithms for calculating values and definitions of conversion factors. Data rules impose some constraints on data, e.g. about the cardinality of data items, mandatory elements, etc. Activity and party rules are out of the scope of our interest. According to [10], action assertions defined in BRG classification [6] belong to operative rules, what, in our opinion is only partially true (see the comment about integrity rules). What is interesting, Witt criticizes the syntax convention proposed both in SBVR and RuleSpeak for definitional rules, and defines its own which uses "… by definition …" phrase.

Table 1 References to categories of business rules that are considered as structural business rules

Classification	Business rules category or subcategory
SBVR [7]	Structural business rules
Witt [10]	Definitional rules Data rules
Von Halle [8]	Derivations Data integrity constraints
Hay [9]	Mandatory constraints
BRG [6]	Derivations Integrity constraints

Table 1 shows subcategories proposed by different authors that in this paper are treated as structural rules.

2.2 *Business Rules Patterns*

The notion of (business rule) pattern is defined differently by different authors, e.g. as "a collection of model elements relating to a particular situation. It is a fragment of a model that's already available, ready for you to incorporate—either as it stands or with modifications—into your own complete model" [11] or "a pattern is a structured way of presenting key elements which can be used to create or identify statements" [12].

We define business rule pattern as an approach to specifying a particular type of business rule that aims at the better quality (uniformity of vocabulary, grammar, better readability, and analyzability). It defines a pre-defined structure of elements involved what limits the way a business rule can be expressed.

The idea of patterns for business rules can be implemented using different approaches [13]:

- Approach 1: a complete syntax describing all valid patterns.
- Approach 2: valid patterns that allow for fill-in-the-blanks.
- Approach 3: free sentence formulation combined with guidelines about the patterns.

An example of the approach 1 is presented in [14], where authors present a controlled language called RuleCNL for Business Rules Specification. The syntax of the language was defined with the use of Extended-Backus-Naur Form (EBNF). Particular rules can reference to the entries of existing business vocabulary. The solution is supported by business rule editor that assists end-users in business rules writing process and automatic mappings into SBVR standard.

The complexity of proposed solution is quite big as you need to define a complete grammar, implement it in a parser, build supporting tools, e.g. editors or consistency checkers. On the other hand, the controlled language constraints the way of language usage, what, according to [13], influences negatively the effectiveness of work: "It forces the author to write complete and correct sentences, which distracts him from the real content—the knowledge—that he would like to record".

Another interesting example of the approach 1 is described in [15] which defines a catalog of patterns for derivation business rules as well as describes a process of its construction. First of all, the author identified the fundamental constructs of derivation business rules, and next described their interrelations using a meta-model. This meta-model formulated the foundation for the creation of patterns, which are formulated with the use of controlled natural language. The author defined 19 basic patterns that can be employed together to constitute more complex business rules. As it can be observed the number of patterns is high as they very often represent small chunks of the business rule statement, e.g. "comparison with value".

The application of the approach 1 can also be found in Graham Witt's book [10]. The author defined a large number of sentence templates for all types of business rules, among others also structural ones. His approach is more prescriptive than the others (e.g. Ross [6]), what means "that any two rules of the same type will be expressed using statements having the same syntax" [16]. To effectively apply his approach one needs to establish the proper type of rule first. What was mentioned in the previous chapter, he proposed a new syntax to express definitional rules, other than RuleSpeak [17] or SBVR Structure English.

An example of the approach 2 is presented in [18]. There is a hybrid approach in which authors define a formal model to produce patterns "that help domain experts familiar with the specific domain to express additional knowledge" [18]. These patterns take a form of sentences of a controlled language with formally defined grammar and mappings to the elements of formal model. From the user perspective the patterns have placeholders which can be freely filled, e.g. There is a class of products named <ProductClass> [18] As it can be seen from the example, the patterns serve mainly to express facts, not structural business rules. The main benefit of proposed approach is syntax and semantics separation. In consequence different languages can be used to express the same knowledge, each one dedicated to specific audience. The problems encountered during approach implementation include limitations in terms of the grammatical quality of generated text (proper expression of multiplicity and gender), what leads to the large number of patterns, and, in consequence, a significant amount of work. One way of using this approach is first to select a pattern, next to fill it with proper elements. According to [13] that may lead to incorrect rules where the patterns are selected prematurely.

Software Requirement Patterns [1] can be referenced as another example of the approach 2, which is much less formal than the previously described. No controlled language is here in use. The structure of requirement patterns resembles the structure of design pattern: we have a pattern name, basic details, applicability, discussion, content, template(s), examples, extra requirements, considerations for development, considerations for testing. Patterns belong to different categories, and, in the context of this paper (patterns for business rules) the most interesting group is Information Requirement Patterns, which contains following patterns: Data Type

Requirement Pattern (field level), Data Structure Requirement Pattern (class level), ID Requirement Pattern, and Calculation Formula Requirement Pattern. An exemplary template of the later pattern is shown below [1]:

<<Value description>> shall be calculated as follows:
<<Value name>> = <<Formula>>
Where <<Variable 1 name>> is <<Variable 1 description>>
<<Variable 2 name>> is <<Variable 2 description>>

Tags in "<<>>" can be freely filled.

RuleSpeak Sentence Forms [17] can be referenced as an example of the approach 3. RuleSpeak is a set of sentence forms for specifying natural-language business rules in English. The main idea behind is that every type of business rule contains a proper key word, e.g. must, must not, may only, which points out the fact that a business rule always removes a degree of freedom. The pattern, except the keyword, contains almost nothing ("…always …", "… must …", "… may … only …"). The sentence forms "do not represent a formal language or syntax for implementing business rules using a software platform" [8]. Except a key word, a business rule should follow some guidelines, e.g. every statement should be a complete sequence, or every statement should have a subject at the beginning. RuleSpeak uses the same pattern for all constraints—we can't distinguish easily operational from structural rules.

In this paper a catalog that follows the approach 2 is proposed. Its definition is similar to that used in [1]. The authors' intention was not to cover all the possible structural rules or to define a complete syntax (a part of controlled language) for them. Instead, she aim to offer a tool, which should help in revealing and specifying typical structural business rules expressed according to RuleSpeak guidelines [19].

3 Definition of a Business Rule Patterns Catalog

3.1 Description of Catalog Design Method

This chapter presents how the catalog of business rule patterns was designed. The method consisted of three main stages:

1. Literature review to find out:
   (a) Definition of structural business rules and their subtypes together with examples
   (b) Definition of business rule patterns for structural business rules and their subtypes
   (c) Examples of structural business rules
2. Definition of business rule patterns catalog for structural business rules
3. Validation of business rule patterns catalog

3.2 Results of Literature Overview

To define the structural business rule catalog one had to investigate definitions of structural business rules together with their examples. In this subchapter, the main findings of literature overview together with resulting specifications of catalog entries are described. The specifications serve later as a basis for pattern catalog definition. The wording used in the specification does not follow any particular notation (will be later changed). The primary sources taken into consideration were those, referenced in Table 1. The specification relates to notions used in UML (Unified Modeling Language) class and object diagrams, which are often used for visualization of business glossary, what makes it more readable.

Definition of structural business rules and their subtypes together with examples

- Necessary characteristics of a class (concept):
  - (a) Some feature(s) of related class *C* must be present (under specific condition (s))—all instances of that class must have these features defined.
  - (b) Some feature(s) of related class *C* must be unique (under specific condition (s)).
  - (c) Some feature(s) of related class *C* must be constant/not changed (under specific condition(s)).
  - (d) Some feature(s) of related class *C* must satisfy condition *cond(C.f1*, …, *C. fn*), where *f1*….*fn* are features of *C*, and the condition defines some constraints about features' ranges, actual values, min, max values, equality, etc.
  - (e) Each *C1* instance must be in relation *R* with *C2* instance(s) if cond(*C1.f1*, *C1.f2*, …, *C2.f1'*, *C2.f2'*), where *f1*, …, *fk* are features of class *C1*, and *f1'*, …, *fn'* are features of class *C2*.
- Constraints on quantity of instances:
  - (f) There is exactly/at most/at least *n* instances of class *C* (n = 0 … k).
  - (g) Each instance of class *C* has exactly/at most/at least *n* instances of a specific feature.
  - (h) Each instance of class C has, at least, *m* and most *n* instances of a specific feature.
  - (i) Each instance of class *C1* is in relation *R* with exactly/at most/at least *n* other instances of class *C2* (n = 0 … k).
  - (j) Each instance of class *C1* is in relation *R* with at least *m* and at most *n* other instances of class *C2* (m = 0 … k, n = 0 … k, $m < n$, note: *C1* and *C2* don't need to be different).
- Inference rules:
  - (k) Each instance of class *C1* must be considered to be a role of class *C2* if condition cond(*C1.f1*, …, *C1.fn*) where *f1*….*fn* are features of *C1*.

(l) Some feature of class C must be considered to be a characteristic if condition cond(*C1.f1*, …, *C1.fn*) where *f1….fn* are features of *C1*.
(m) Some feature of class *C* must be calculated according to a specific formula.

- Constraints on instances (power sets):

(n) None instance of *C1* is an instance of *C2* (*C1* and *C2* should have the same parent).
(o) Each instance of *C* must be an instance of *C1* or *C2* (*C1* and *C2* should be subclasses of *C*).

3.2.1 Definition of business rule patterns for structural business rules and their subtypes

Three sources with business rule patterns were considered to be a source for new specifications of structural business rule patterns, namely: [1, 10, 15]. However, the author did not manage to find out such types of rules that haven't been covered yet (at least at the general level). Some patterns proposal were irrelevant, e.g. Data Structure Requirement Pattern that is used "to define a compound data item (one that comprises multiple individual pieces of information)" [1]; according to SBVR this is a part of vocabulary definition.

3.2.2 Structural business rule examples

The last source that was used to establish business rule patterns were structural business rules examples defined in [7]. The document itself contains plenty of structural business rules which were used as "training set". Structural business rules defined in [7] were grouped according to their types what allows discovering some new groups of rules: Conditional constraints on instances quantity, and Complex invariants. The group called Conditional constraints on instances quantity contains extended versions of rules (f)–(i), with condition part.

(i') Each instance of class *C1* satisfying condition cond(*C1.f1*, …, *C1.fk*) is in relation *R* with exactly/at most/at least *n* instances of class *C2* (n = 0 … k).

Example: Each proposition that *is true corresponds to* exactly one actuality [7].

Complex invariants contain implications and complicated invariants that traverse through navigational paths in a complex manner. It is not possible to define a supporting pattern for that type of rules, or the pattern would be defined at very general level. Below you have some examples of the rules from this category: "If a concept1 *is coextensive with* a concept2 then the extension *of* the concept1 *is* the extension *of* the concept2". "The set *of* characteristics that *are incorporated by* a general concept *is* not the set *of* characteristics that *are incorporated by* another general concept" [7]. Proposed pattern catalog does not cover such rules.

3.2.3 Summary

Four groups of structural rule patterns specifications were identified. Each of them contains from 2 to 5 patterns (15 patterns in total). For those who are familiar with UML and OCL (Object Constraint Language), structural business rules are those which can be described either directly on a UML class diagram (e.g. multiplicity constraints, power set constraints) or by OCL expressions (invariants, derivations, and queries—for calculations).

3.3 Catalog Definition

This chapter brings the definition of catalog for structural business rule patterns (See Table 2). It is presented with the assumption that the vocabulary is expressed in an object-oriented manner (e.g. as a simplified version of a class diagram, without constraints), or a user is familiar with concepts from object-oriented paradigm.

The sentence patterns follow RuleSpeak guidelines [19]. They use tags ("<<>>") representing gaps to be filled with elements from a vocabulary. The tag itself either defines the type of vocabulary entry: i.e. <<Class>> (name of concept at the higher level), <<Feature>> (name of concept being an attribute or available through navigational path), <<Role>> (name of associated concept), or is defined as <<Condition>>—any condition accessible in the context formulated by other tags, can be both simple and complex (and, or, not).

The pattern uses a simplified BNF-notation, where "[]" is used to describe an optional element, and "|" is used to describe the alternative. It should be noted than the patterns are not part of controlled language, so the grammar elements, e.g. plural names are not taken into consideration.

For each pattern at least one example is given.

3.4 Catalog Validation

The catalog was constructed to help in revealing different types of constraints put on defined vocabulary. The primary intention was to propose a limited number of patterns that is expressive enough to cover as many of business rules as possible. A simple case study was conducted to verify if this aim was achieved.

There exist only a few complete sets of business rules sets (at least those containing structural rules). One of them is [5]. It was used for validation purposes. It includes 59 structural rules in total, but 27 define inheritance hierarchy, which could be directly expressed at glossary level (e.g. "The concept corporate renter is included in Renters by Affiliation"). The rest of rules were classified to pattern groups (if possible) and rewritten with the usage of patterns, e.g. (Each rental

Table 2 Definition of sentence patterns for structural business rules

Group 1: Concept (Class) Necessary Characteristics		
Pattern no	Pattern	Example(s)
1.1.	[Each] <<Class>> always has <<Feature>> [[if | when | if and only if] <<Condition>>]	Each student always has student number A rental always has a business currency (feature) if and only if the business currency is the currency of the operating country of the operating company that includes the local area that includes the pickup branch of the rental [SBVR]
1.2.	[The] <<Feature>> of [each] <<Class>> always is unique [if <<Condition>>]	Student number of each Student always is unique
1.3.	[The] <<Feature>> of [each] <<Class>> never change [[if | when] <<Condition>>]	PESEL of Employee never change Dean of Faculty never change during holidays
1.4.	[The] <<Feature>> of [each] <<Class>> always <<Condition>>	PESEL of Employee always is 11 characters long Gender of Person always belongs to the set {male, female} Shipment date of Order is always after order date of Order
1.5.	[Each] <<Class>> always is in relation <<Relation>> with <<Class>> [if | when | if and only if] <<Condition>>	Each University always is in relation "located" with City
Group 2: Instances Quantity Constraints		
2.1.	There always is [exactly | at most | at least] *N* <<Class>> [if <<Condition>>]	There always is exactly 1 Dean
2.2.	[Each] <<Class>> always has [exactly | at most | at least] *N* [instances of] <<Feature>>	Each Employee always has at least 1 Phone number
2.3.	[Each] <<Class>> always has at least *M* and at most *N* <<Feature>> [if <<Condition>>]	Each Employee always has at least 1 and at most 2 surnames
2.4.	[Each] <<Class>> [< <Condition>>] always is in relation <<Relation>> with [exactly | at most | at least] *N* [instances of] <<Class>>	Each Student always is in relation "enroll" with at most 5 Courses Each Student that is distinctive always is in relation "awarded" with at least 1 Award
2.5.	[Each] <<Class>> [satisfying <<Condition>>] always is in relation <<Relation>> with at least *M* and at most *N* [instances of] <<Class>> [if <<Condition>>]	Each Student always is in relation "enroll" with at least 1 and at most 10 Courses

(continued)

Table 2 (continued)

Group 1: Concept (Class) Necessary Characteristics		
Pattern no	Pattern	Example(s)
Group 3: Inference Rules		
3.1	[Each ∣ A ∣ An] <<Class>> is to be considered [a] <<Role>> [if <<Condition>>]	A location is to be considered a third-party location if located at EU-Rent site that is owned by a third party
3.2	[The] <<Feature>> of [a] <<Class>> is to be considered <<Characteristic>> [if <<Condition>>	The car storage capacity *of* a branch is to be considered minimal if *less than* 10
3.3	[The] <<Feature>> of [a] <<Class>> is to be computed as <<Formula>>	The age of a Person is to be computed as a difference between the current year and the year of the birthday of that Person
Group 4: Power Set		
4.1.	None [instance of] <<Concept>> is [an instance of] <<Concept>>	None Student is an instance of Teacher
4.2.	[Each [instance of]] <<Concept 1>> always is [an instance of] <<Concept 2>> or <<Concept 3>> or … (note: <<Concept 2>>, <<Concept 3>>, …. must be subclasses of <<Concept 1>>)	Each Person always is an instance of Student or Teacher

Table 3 Verification of structural rule patterns catalog

Pattern group	Instance no/Percentage (of 57) (%)
Group 1	28/49
Group 2	21/36
Group 3	4/7
Group 4	0/0
Sum	53/92

booking always has exactly one booking date/time). After then the number of rules from each group was counted. Table 3 presents the results.

52 from 57 rule statements from [5] could be expressed with the use of defined patterns (92 %), what is a satisfied result. 6 rules were not covered, e.g. “A cash rental price of a rental that is calculated because of EU-Rent price changes and that is less than the lowest rental price of the rental replaces the lowest rental price of the rental” [5]. Only one group from defined pattern catalog was not represented.

Some key limitations of the presented work should be outlined. The first is that the proposed catalog must not be considered complete. It was constructed on the base on literature overview with the intention to address the typical cases. The second is that the business rules patterns have been defined as a kind of boilerplates, a sequence of slot restrictions depending on some business vocabulary. Although these patterns are implementation-independent, they could be transformed into different languages such as UML and OCL; in such a case the usage of boilerplates could also lead to a lack of expressivity regarding necessary notions for transformation purpose, e.g. cardinality of relationships. The third is that the implementation of the business rules patterns within tools such as e.g. knowledgeMANAGER is still an open issue. It will help to validate both, the patterns and possible realizations of such patterns, i.e. business rules. It will also help to identify and extract business rules from different sources such as specification documents.

4 Conclusions and Further Work

The paper describes a catalog for structural business rules together with a method of its construction. The intention of the catalog is to support business analyst during the business rules identification process as well as assuring the rules are written in a consistent manner. The catalog was developed on the base on literature overview including investigation of existing catalogs. One of the main aims was catalog minimalism, i.e. the catalog should contain only a limited number of patterns, only those that cover the most obvious cases. According to the case study, described in Sect. 3, the catalog met this requirement. The proposed catalog consists of only 15 patterns, while, e.g. the catalog that covers only derivation rules [15] contains 19 patterns.

In the proposed catalog some of the business rules patterns are quite restrictive, preventing the matching of complex business rules. However, it is possible to extend the patterns with sub-patterns which represent more complicated cases (restrictions). It is also possible to customize the restrictions for a particular domain or situation but keeping the original set of high-level business rules patterns.

In the future, the flexibility of proposed pattern catalog will be checked with other existing sets of structural business rules, e.g. [10]. The catalog will also be recommended to students taking part in Software Engineering courses. That will allow answering the questions about practical quality aspects of catalog application: its readability, completeness, and the real help in reviling and documenting existing business rules.

We also consider to prepare a formal version of the patterns catalog on top of existing ontology, and to implement it in the knowledgeMANAGER tool [20], part of the Requirements Quality Suite (RQS). That would make possible to automatically perform processes for quality analysis (consistency, completeness and correctness) or recovery traceability links between business rules and any other software or system artifact. It should also have the potential of being used to automatically generate source code (e.g. Object Constrained Language constraints), test cases and to enable cross-domain reuse of business rules (e.g. for certification purposes).

Acknowledgments This work was partially supported by the European Commission under the 7th Framework Programme, Coordination and Support Action, Grant Agreement Number 316097, ENGINE - European research centre of Network intelliGence for INnovation Enhancement.

References

1. Withall, S.: *Software Requirement Patterns*. Microsoft Press (2007)
2. The Business Rules Manifesto. http://www.businessrulesgroup.org/brmanifesto.htm
3. Ross, R.G., Lam, G.S.: *Building Business Solutions: Business Analysis with Business Rules, Business Rule Solutions*, 2nd edn. (2015)
4. Dick J., Llorens, J.: Using statement-level templates to improve the quality of requirements, White paper (2012). http://www.integrate.biz/downloads/white_paper-templating.pdf
5. http://www.kdmanalytics.com/sbvr/vocabulary.pdf
6. The Business Rules Group: Defining Business Rules—What Are They Really? Final Report, revision 1.3, (2000). http://www.businessrulesgroup.org/first_paper/BRG-whatisBR_3ed.pdf
7. Object Management Group: Semantics of Business Vocabulary and Business Rules (SBVR), v. 1.3 (2015)
8. Von Halle, B.: *Business Rules Applied: Building Better Systems Using the Business Rules Approach*, pp. 29–30. Wiley, New York (2001)
9. Hay, D.C.: *Requirements Analysis: From Business Views to Architecture*, Chapter 8. Prentice Hall (2002)
10. Witt, G.: *Writing Effective Business Rules, A Practical Method*, Chapter 4. Elsevier (2012)
11. Morgan, T.: Business Rules and Information Systems: Aligning IT with Business Goals. Addision-Wesley, London (2002)
12. Zoet, M.: Methods and Concepts for Business Rules Management. Doctoral Thesis. Universiteit Utrecht, Utrecht (2014)
13. Spreeuwenberg S.: Patterns and the capture of business rules. Bus. Rule J. **14**(10) (2013)
14. Njonko, F., Brillant, P., Cardey, S., Greeneld, P., El Abed W.: RuleCNL: a controlled natural language for business rule specifications in controlled natural language. In: 4th International Workshop, CNL 2014, Galway, Ireland, August 20–22, 2014. Proceedings. Springer International Publishing (2014)
15. de Haan, E.Y.: Patterns for Derivation Business Rules, Master Thesis, Utrecht University (2015)
16. http://dataqualitypro.com/data-quality-pro-blog/how-to-create-effective-business-rules-graham-witt
17. Ross, R.: Rule speak sentence forms. Specifying natural-language business rules in English. Bus. Rules J. **10**(4) (2009)
18. Spreeuwenberg, S., van Grondelle, J., Heller R., Grijzen, G.: Using CNL techniques and pattern sentences to involve domain experts in modeling in controlled natural language. In: Second International Workshop, CNL 2010, Marettimo Island, Italy, September 13–15, 2010. Revised Papers. Springer, Berlin (2012)
19. Object Management Group: Semantics of Business Vocabulary and Business Rules (SBVR), v. 1.3. Annex H—The RuleSpeak Business Rule Notation (2015)
20. The Reuse Company Inc., knowlegeMANAGER (KM). *knowledgeMANAGE* (2014)

Sketching Use-Case Scenarios Based on Use-Case Goals and Patterns

Mirosław Ochodek, Krystian Koronowski, Adam Matysiak, Piotr Miklosik and Sylwia Kopczyńska

Abstract Use cases are a scenario-based technique used to express functional requirements from the user perspective. They are often elicited using a top-down approach by first identifying their goals and later documenting their scenarios. In the paper, we investigate the possibility of supporting analysts in progressing from definitions of use-case goals to full documentation of their scenarios. We propose a semi-automatic approach to generate use-case scenarios based on use-case patterns. The proposed approach is a result of an empirical analysis of 217 use cases from 12 projects. The analysis revealed that a notion of use-case transaction could be used to organize use-case patterns into a catalog of patterns. We have implemented a prototype tool called UC-Sketch to illustrate the proposed idea. The acceptance of the proposed approach by its potential users was assessed through the use of Technology Acceptance Model.

Keywords Use cases · Use-case patterns · Use-case transactions · Technology acceptance model

1 Introduction

Use cases are a scenario-based technique for expressing functional requirements from the user perspective. They have gained a lot of attention from the software engineering community [1, 2] since their introduction in the 1980s [3].

Several authors recommended to elicit use cases according to a top-down and breadth-before-depth strategy [4, 5]. When following such an approach, firstly, business-level goals are identified [6], which further gives a basis for determining user-level goals and sub-functional goals. As a result, a hierarchy of use cases is created which forms a kind of "unfolding story." The breadth-before-depth strategy

M. Ochodek (✉) · K. Koronowski · A. Matysiak · P. Miklosik · S. Kopczyńska
Faculty of Computing, Institute of Computing Science, Poznan University of Technology, Ul. Piotrowo 2, 60-965 Poznań, Poland
e-mail: Miroslaw.Ochodek@cs.put.poznan.pl

L. Madeyski et al. (eds.), *Software Engineering: Challenges and Solutions*,
Advances in Intelligent Systems and Computing 504,
DOI 10.1007/978-3-319-43606-7_2

means that one should elicit use cases at a given level of abstraction before decomposing them to into a set of more precise ones. Consequently, the elicitation process of user-level use cases is often divided into two stages: *identification* of use cases, and *documentation* of their scenarios. The outcome of the first stage is a list of use-case names, usually visualized using UML use-case diagrams. In the second stage, use cases are analyzed and documented with scenarios.

Taking into account existing similarities between scenarios of use cases [7, 8], it would be beneficial to investigate if it is possible to use historical data concerning use cases to support the process of transitioning between use-case names (which express goals of use cases) and use-case scenarios. In this context, the following problem can be formulated:

Problem 1 Given the name of a use case (a definition of its goal), provide a plausible documentation of its scenarios.

A reasonable approach to address the above-stated problem is to identify and document patterns of commonly appearing use cases (or directly copy and modify previously created ones). Several catalogs of use-case patterns have been proposed so far [9, 10]. They provide guidance on how to model frequently appearing types of user goals with use cases, and usually provide some examples or templates. These examples might be used as a basis for authoring new use cases. However, there are at least two problems with such an approach. Firstly, (1) the examples need to be manually rewritten to match the context, and secondly (2) the aforementioned patterns usually focus on very specific, domain-oriented goals which limits their generalizability.

In the paper, we propose a different approach to organizing catalogs of use-case patterns. It allows structural patterns of use cases to be defined independently of the business domain at three levels of abstraction: use-case actions, use-case transactions, and scenarios, and associates them with use-case goals. In addition, we propose a method and prototype tool that enables semi-automatic generation of use-case scenarios based on use-case patterns. We call this approach "sketching a use case" because it allows the creation of an initial version of use-case documentation quickly. To develop the proposed approach we investigate the following research questions:

- **RQ1**: How to organize a catalog of use-case patterns that binds use-case goals and scenarios, but is independent of business domain?
- **RQ2**: How to store information about use-case patterns and generate use case scenarios in a semi-automatic way?

The paper is organized as follows. In Sect. 2, we discuss the related work concerning use-case patterns. Section 3 concerns RQ1 and describes the organization of a use-case patterns catalog. The following Sect. 4 addresses RQ2 and presents the process of sketching use cases and prototype tool called UC-Sketch. A preliminary evaluation of the usefulness of the proposed approach is presented in Sect. 5. Section 6 discusses limitations and generalizability of the study. The paper is concluded in Sect. 7.

2 Related Work

Use-case patterns could be defined at different levels of abstraction, i.e., for a set of use cases, a single use case, scenario, or step [4]. Also, the form used for expressing use-case patterns differs visibly between the existing catalogs of patterns. Some authors provide exemplary fragments of use-case models that solve some problems; others define use-case patterns as a set of guidelines.

Probably the most recognized catalog of use-case patterns was proposed by Övergaard and Palmkvist [10]. They documented several patterns of use cases and provided a catalog of blueprints that guides how to compose use-case models for some common features, e.g., access control, future task, report generation. A similar approach to documenting use-case patterns was proposed by Issa et al. [7, 9]. From the perspective of the considered Problem 1, these kinds of patterns could be used by copying the examples or templates they provide and adjusting them to a particular context.

Saeki [11] proposed a different approach to derive and organize use-case patterns. He suggested using the UML generalization relationship to derive a specific use case from a pattern, or to use the Decorator pattern to adjust existing use-case patterns to a given context. This approach makes it possible to maintain associations between use-case patterns and use cases. However, the resulting use-case model might be difficult to understand by the readers who are IT-laymen, not familiar with the UML notation.

A different approach to solving the considered problem is to find similarities between previously created requirements and the ones that are currently being analyzed. For instance, the problem of similarity analysis between textual requirements has been considered before by och Dag et al. [12]. In the context of use cases, Kaindle et al. [13] proposed to use similarity measures to find commonality between currently developed and historical use cases. Because the approach relies on previously-created use cases, it might have even wider applicability than use-case patterns. Unfortunately, to calculate similarity measures (and find similar use cases) one has to document at least a part of a new scenario.

There were also several studies focused on generating scenarios. These studies are not directly related to use cases. However, because use cases are a scenario-based technique, they seem applicable also in this context. For instance, Rolland et al. [14] proposed an approach to composing scenarios from so-called requirements chunks—a pair <Goal, Scenario>, and to use three types of relationships: composition, alternative, and refinement. However, they mainly focused on guiding goal modeling rather than documenting and re-using patterns. Ridao et al. [15] studied different types of episodes in scenarios. They formulated four main scenario patterns, i.e., production, collaboration, service, and negotiation. They also consider the possibility of automatically generating sets of candidate scenarios. However, the proposed approach requires the existence of a language extended lexicon (LEL), thus, it is not directly applicable to use-case names. Watahiki and Saeki [8] focused on generating scenarios from case frames (an abstract representation of a sentence characteristic for a given domain, e.g., [reserve, actor, object]) that fulfill order constraints.

At the level of scenario actions, the research focused mainly on linguistic patterns. For instance, the most recognized are the language patterns proposed by Rolland et al. in the CREWS project [16]. These studies seem complementary to ours because the proposed linguistic patterns might be used to formulate action templates in the proposed approach.

We might conclude that there are at least several approaches that might support solving Problem 1. However, it seems that they do not provide a holistic approach that would cover the possibility of defining patterns at different levels of abstraction: actions, episodes of scenarios, use-case scenarios, and use-case goals. In addition, most of the existing approaches rely on copying and modifying the proposed exemplary use cases or templates to create new instances of use cases.

3 Use-Case Patterns Catalog

To answer RQ1, we have to find a way to associate use-case goals and appropriate patterns of scenarios. The classification of goals should be, as much as possible, independent of business domain.

We decided to investigate this question empirically, by analyzing a set of 217 use cases coming from 12 software projects that developed web applications [17]. In particular, we aimed at investigating the relationship between goals of use cases and use-case transactions forming their scenarios.

Use-case transaction is defined as "the smallest unit of activity that is meaningful from the actor's point of view that is self-contained and leaves the business of the application being sized in a consistent state." [18] In our previous studies [17, 19], we proposed a categorization scheme of transactions that consist of twelve types of use-case transactions: Create (C), Retrieve (R), Update (U), Delete (D), Link (L), Delete Link (DL), Asynchronous Retrieve (AR), Dynamic Retrieve (DR), Transfer (T), Check Object (CO), Complex Internal Activity (CIA), and Change State (CS). These types refer to general tasks and are not bound to any specific business domain.

During the analysis of the sample of use cases, we observed that more than half of user-level use cases (55 %) consisted of a single transaction that corresponded to the goal of a use case. We also observed that use cases consisting of more than one transaction usually had a *dominant transaction* that reflected the goal of a use case and a set of auxiliary transactions. For instance, in a use case entitled "modify a product," the modification of the product could be preceded by the presentation of the products that could be modified. In such a case, a *dominant* transaction would be Update and Retrieve would play a supportive role. Therefore, we assume that *the type of dominant transaction might be used to summarize the goal of a use case*. The only exceptions from this rule were the CRUD (Create, Retrieve, Update, Delete) use cases that have more than one type of dominating transaction. Thus, we extended the

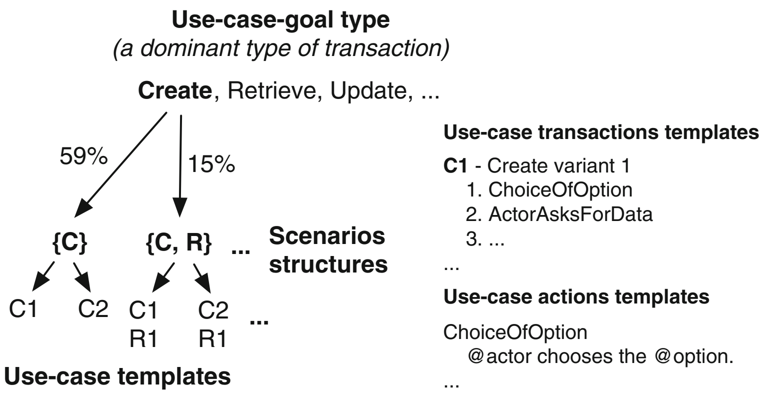


Fig. 1 Organization of the use-case patterns catalog

set of goals categories with CRUD. In fact, the set of categories can be extended if new types of goals are identified.

The resulting structure of a use-case catalog is shown in Fig. 1. It is organized into three layers. The first layer consists of categories of use-case goals. In the second layer, the goals are associated with the structures of use cases. We define structure by providing a set of acronyms of types of transactions it includes (e.g., {C, R} consists of Create and Retrieve). The additional benefit is that we could augment the catalog with the information about the probability of a given structure appearing in the use case of a given type of goal. This information might be used to suggest analyst the most probable pattern. In the third layer, each structure of scenario is associated with a set of *use-case templates*. Such templates are defined at three levels. The smallest unit is called *action template*. These templates are used to compose *transaction templates* that are used to define scenarios in use-case templates.

In the analyzed set of use cases, we identified 57 different structures of use-case scenarios. The most frequently appearing ones are presented in Table 1. They seem to be good candidates for use-case patterns.

3.1 Action Templates

Use-case action represents the smallest unit of activity in a use-case scenario perform by an actor. Each action template has a *unique identifier*, *name*, *parameters*, and *language transforms*.

Parameters are used to determine the fragments of a template that can be configured while generating the text based on the template. A parameter can have a set of default values (one per supported natural language). For instance, a parameter @actor in the action "select object" might have a default value "uytkownik" in Polish and "user" in English.

Table 1 The most frequently appearing of configurations of transactions among the analyzed set of 217 use cases from 12 software projects

Goal of use case	Structure of scenarios	Frequency	Goal of use case	Structure of scenarios	Frequency
C	{C}	0.59	L	{L}	0.60
	{C,R}	0.15		{DL,L,R}	0.16
R	{R}	0.47		{L,R}	0.08
	{DR,R,R}	0.13	DL	{DL}	1.00
	{R,R}	0.08	AR	{AR}	1.00
U	{U}	0.73	DR	{DR}	0.86
	{R,U}	0.15		{DR,R}	0.14
D	{D}	1.00	CIA	{CIA}	1.00
T	{T}	0.60	CS	{CS}	0.55
	{R,T}	0.20		{CS,R}	0.27
	{R,T,U}	0.20	C,D,R,U	{C,D,R,U}	0.41
CO	{CO}	0.80		{C,D,U}	0.31
	{CO,R}	0.20		{C,D,L,U}	0.10

Language transforms are boilerplates that are used to generate instances of actions in a given language. They may include references to parameters with the specification of their linguistic forms (e.g., the number—singular, plural; form—normative, accusative, etc.).

Parameters and transforms are important in the context of RQ2 because they provide a simple natural language generation (NLG) mechanism.

The execution of an action might be interrupted by events, which are described in the extensions section of a use case. Event templates are defined similarly to action templates.

3.2 Transaction Templates

Transaction template is defined by a *unique identifier*, *name*, name of *type*, and *description*. Similarly to actions, they can define *parameters*.

The structure of transaction is defined by a sequence of *actions* and corresponding *extensions*. Actions templates are included by referring to their unique identifiers discussed before. Also, values of parameters can be passed between transaction and action templates, which is similar to invoking a programming function with parameters. One can also bind the values of parameters defined at the level of transactions with the parameters in the action template. For instance, if a parameter @actor is defined at the level of transaction, one can bind its value with the parameter @actor defined by the action “select object” that is included in the transaction’s template.

This mechanism allows the flow of data between templates defined at the higher level of abstraction and the ones at the lower levels.

Extensions consists of an *event* that may occur while performing an action, and a sequence of *actions*/*transactions* describing the alternative scenario, and a set of *parameters*. Optionally, one can state which transaction should be performed after executing the alternative scenario.

3.3 Use-Case Templates

Use-case template has a *unique identifier*, *name*, definition of *goal*, and *description* giving the purpose of the use-case pattern.

Similarly to definitions of templates at other levels, it can also define a set of *parameters*. The scenario of a use case is a sequence of *transactions* and optional *extensions*. The parameters at the level of a use case can be bound with the parameters of included transactions and extensions (in the same way as for transactions and actions).

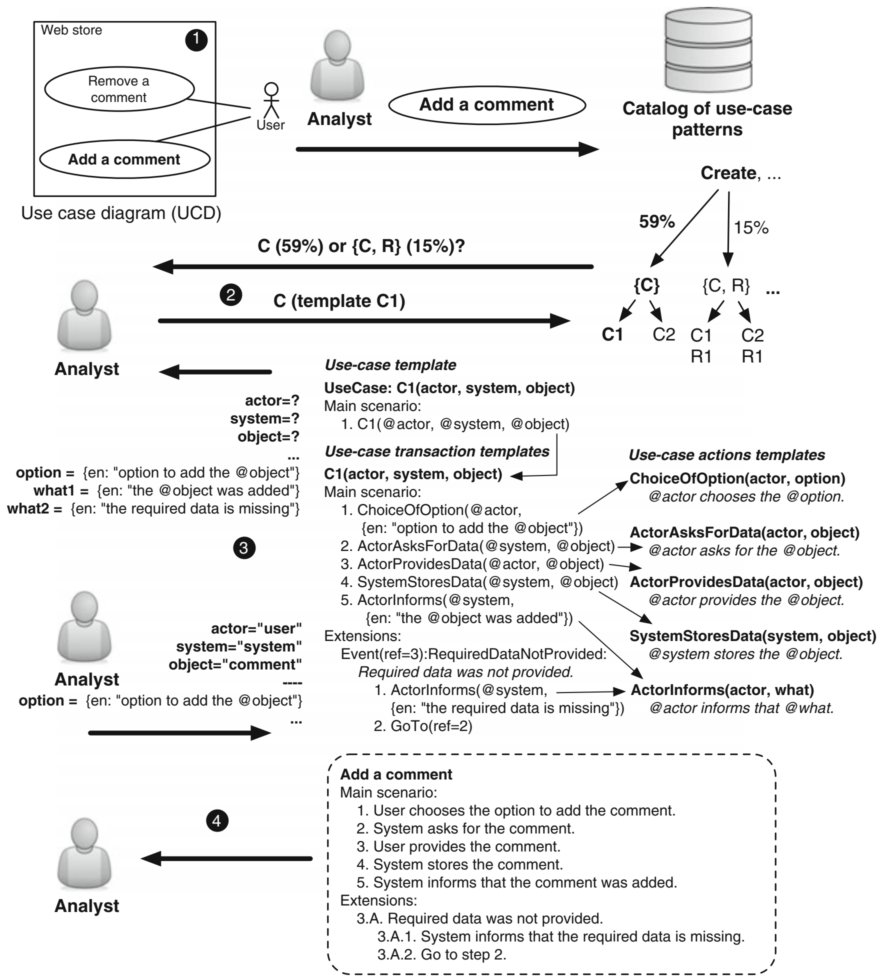
4 Sketching a Use Case Based on Its Goal

The process of sketching a use case based on use-case pattern is presented in Fig. 2 and consists of the following steps:

1. The goal of a use case is identified and documented as a use-case name.
2. Based on the goal, an analyst determines the dominant type of transaction in the use cases, chooses one of the structures of use-case scenarios available for the goal and one of the associated use-case templates.
3. A set of templates parameters that might be modified by the analyst is determined. This process begins with identifying parameters in the included action templates. Afterward, the parameters of transaction templates are determined. If any of these parameters is bound to the parameters of included actions, the action parameter is excluded from a set of modifiable parameters because its value will be automatically determined based on the transaction's parameter. In the following step, the actions' transforms are analyzed to determined the required linguistic form of a parameter. Finally, the analyst is provided with the default values of parameters and asked to modify them appropriately.
4. The values of parameters and transforms are merged to generate the text in a given natural language.

The above-described process of sketching use cases was implemented in a prototype tool called UC-Sketch[1] (see Fig. 3). The tool provides most of the features pre-

[1]https://ucsketch.cs.put.poznan.pl.



```
Use-case template
UseCase: C1(actor, system, object)
Main scenario:
  1. C1(@actor, @system, @object)
Use-case transaction templates
C1(actor, system, object)
Main scenario:
  1. ChoiceOfOption(@actor,
       {en: "option to add the @object"})
  2. ActorAsksForData(@system, @object)
  3. ActorProvidesData(@actor, @object)
  4. SystemStoresData(@system, @object)
  5. ActorInforms(@system,
       {en: "the @object was added"})
Extensions:
  Event(ref=3):RequiredDataNotProvided:
    Required data was not provided.
      1. ActorInforms(@system,
           {en: "the required data is missing"})
      2. GoTo(ref=2)
```

```
Use-case actions templates
ChoiceOfOption(actor, option)
  @actor chooses the @option.
ActorAsksForData(actor, object)
  @actor asks for the @object.
ActorProvidesData(actor, object)
  @actor provides the @object.
SystemStoresData(system, object)
  @system stores the @object.
ActorInforms(actor, what)
  @actor informs that @what.
```

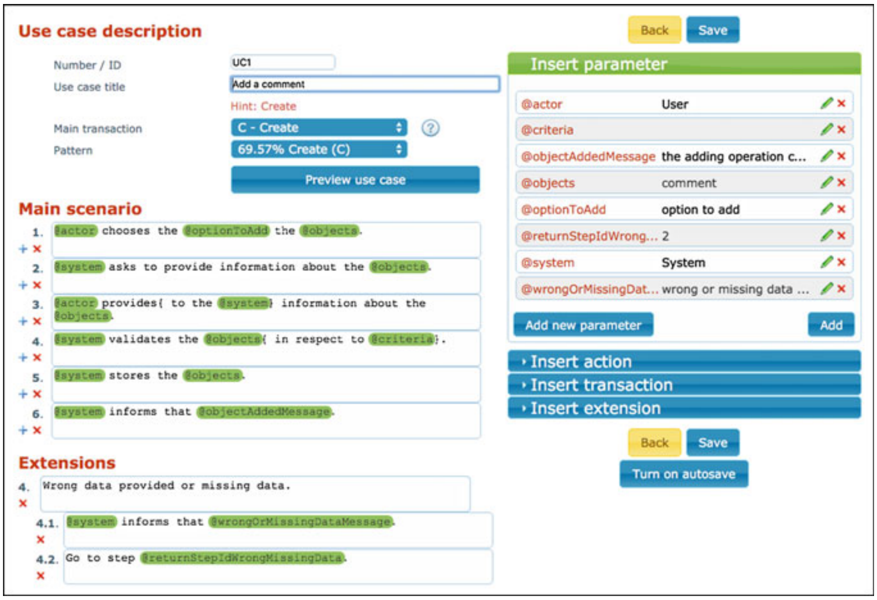
```
Add a comment
Main scenario:
  1. User chooses the option to add the comment.
  2. System asks for the comment.
  3. User provides the comment.
  4. System stores the comment.
  5. System informs that the comment was added.
Extensions:
  3.A. Required data was not provided.
    3.A.1. System informs that the required data is missing.
    3.A.2. Go to step 2.
```

Fig. 2 The overview of the process of sketching a use case based on use-case pattern

sented in the paper and some additional ones increasing its usability. For instance, it predicts the type of dominant transaction based on use-case names. It is also integrated with Wiktionary,[2] which is used to fetch the required linguistic forms of words.

The built-in catalog of use-case patterns provides a set of patterns identified during the study (available in Polish and English). The patterns are stored using a dedicated XML-schema and can be changed through a back-end of the system.

[2]https://www.wiktionary.org.



Use case description

Number / ID	UC1
Use case title	Add a comment
	Hint: Create
Main transaction	C - Create
Pattern	69.57% Create (C)

@actor	User
@criteria	
@objectAddedMessage	the adding operation c...
@objects	comment
@optionToAdd	option to add
@returnStepIdWrong...	2
@system	System
@wrongOrMissingDat...	wrong or missing data ...

Main scenario

1. @actor chooses the @optionToAdd the @objects.
2. @system asks to provide information about the @objects.
3. @actor provides{ to the @system} information about the @objects.
4. @system validates the @objects{ in respect to @criteria}.
5. @system stores the @objects.
6. @system informs that @objectAddedMessage.

Extensions

4. Wrong data provided or missing data.

4.1. @system informs that @wrongOrMissingDataMessage.

4.2. Go to step @returnStepIdWrongMissingData.

Fig. 3 UC-sketch use-case editor—sketching a use case view

5 Preliminary Evaluation

We decided to perform a technology acceptance forecasting study based on Technology Acceptance Model (TAM) [20] to investigate if the proposed approach and tool have a chance to be accepted by its potential users.

TAM is an information system theory that models how users come to accept and use technology. The purpose of this model is to predict the acceptability of information system and to identify the modifications which must be brought to the system to make it acceptable to the users. The model assumes that the acceptability of a system could be determined by measuring two main factors:

- **perceived usefulness (PU)** which is "the degree to which a person believes that using a particular system would enhance his or her job performance."
- **perceived ease of use (PEOU)** which is "the degree to which a person believes that using a particular system would be free from effort."

Devis [20] proposed to measure PU and PEOU using six Likert's items per each of them. All items create a summated rating scale. The answers for each item is treated as being expressed on a 7-point interval scale. The final score of a respondent is the sum of his/her answers to all items.

We conducted the TAM survey based on questionnaires provided in the appendix of Davis's work [20] within three groups of potential users of UC-Sketch:

- 3rd-year B.Sc. computer science students.

Table 2 Demographic information of the sample (values in brackets: 3rd-year students/4th-year students/practitioners)

Variables	Number (N)	Percent (%)
Research group		
Bachelor students	39	61.90
Master students	5	7.94
Professionals	19	30.16
Years of experience		
0–2	39 (32/3/4)	61.90
3–5	17 (6/1/10)	26.98
6–8	6 (1/1/4)	9.52
9–11	0 (0/0/0)	0.00
12 or more	1 (0/0/1)	1.59
Use case experience		
Never heard	0 (0/0/0)	0.00
Heard, but not used	2 (1/0/1)	3.17
Wrote a few UC	32 (30/2/0)	50.79
Used in one project	14 (4/3/7)	22.22
Used in many projects	15 (4/0/11)	23.81
Work experience (multiple answers possible or blank)		
Programmer	52 (28/5/19)	82.54
Architect	12 (4/1/7)	19.05
Analyst	12 (4/2/6)	19.05
Project Manager	8 (1/1/6)	12.70

- 4th-year students of M.Sc. degree in Software Engineering.
- Professionals working in software development industry (we did not include students who are working in IT companies).

Demographic information of the sample is presented in Table 2. Most of the participants have authored few use cases in their career. Unfortunately, only 16.67 % of respondents performed the role of an analyst (four of them were 3rd-year students, who had less than two years of experience in IT projects).

We began the analysis by verifying reliability and validity of the measurement instrument. To measure reliability we used Cronbach's alpha. It is agreed that the acceptable level of the alpha measure should be equal to 0.70 or above. We calculated alpha for each variable and group of respondents and received values between 0.89 and 0.98. Therefore, the measurement instrument seemed reliable. To verify validity we used Factor Analysis [21], which confirmed that two factors were measured by the instrument.

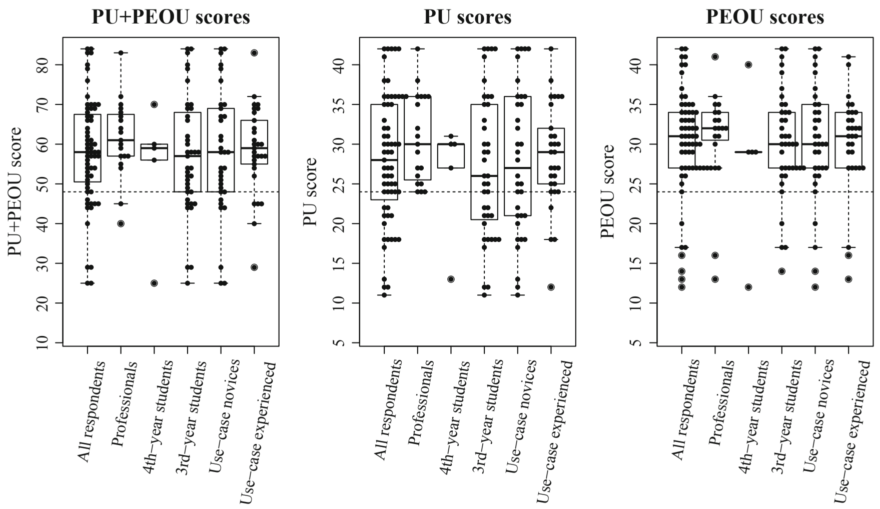


Fig. 4 The results of TAM analysis: PU, PEOU, and PU+PEOU scores (*dashed line*: a central point of the scale)

Finally, to interpret the results we summed up the scores for each participant. The scale ranged from 6 to 42 (6 items × 1:7) for a single variable and from 12 to 84 for two variables together.

The distributions of scores are presented in Fig. 4. The overall average score of participants was ca. 58 for all groups, which is 10 points above the midpoint of the scale, and could be interpreted as 69 % of the maximum value. From the two variables PU and PEOU, the higher results were observed for PEOU (according to the model it also affects PU). However, even PU was higher than the midpoint of the scale. It was higher for professionals and master students than for 3rd-year students. Similar results were observed when we divided participants based on their experience in applying use cases.

Based on the results, we might conclude that the respondents seemed positive towards using the proposed tool.

However, there are at least two threats to validity. First of all, the sample was not a homogeneous sample of analysts—the most appropriate respondents. The second problem is that the TAM method assesses the product rather than the concept. Therefore, it might be difficult to state whether participants liked/disliked the idea or particular features of the software tool.

6 Limitations and Generalizability

Although the results of the TAM analysis shows that the proposed approach to sketching use cases is perceived as useful, there are some limitations of the proposed approach.

First of all, the process of generating use-case scenarios based on use-case goals at early stages of requirements analysis is subjected to the risk of proposing a misleading scenario because of the uncertainty related to requirements. The other issue is the accuracy of the out-of-the-box generated scenarios. In some cases, the generated scenarios might be perceived as correct although they miss some important details. Therefore, a generated scenarios should be treated as a "sketch" and should be revisited and refined later on.

An important threat to the generalizability of the proposed approach is the approach used to author use cases. The use cases analyzed in the study were written in accordance with the state-of-the-art guidelines for writing use cases [4, 5]. Therefore, at the moment, we can generalize our observations only to such use cases.

7 Conclusions

In the paper, we investigated the problem of semi-automatic generation of use-case scenarios based on use-case goals with the use of use-case patterns.

We propose a new approach to organizing catalogs of use-case patterns and use this information to support analyst in documenting use cases by generating proposals of use-case scenarios.

The analysis presented in the paper allow us to provide the following answers to the research questions:

RQ1: *How to organize a catalog of use-case patterns that binds use-case goals and scenarios, but is independent of business domain?*

Based on the analysis of 217 use cases, we proposed a three-layered organization of use-case patterns catalog. The categorization is based on the concept of semantic type of use-case transaction. The considered types of transactions seem general and not bound to any particular business domain. The first layer of the catalog characterizes use-case goals based on dominant types of transactions in their scenarios. The second layer consists of structures of scenarios (types of transactions) that appear in use cases with a given type of goal. In the third layer, we store templates of use cases.

RQ2: *How to store information about use-case patterns and generate use case scenarios in a semi-automatic way?*

We proposed to store templates of use-cases at three levels: actions, transactions, and use cases. The templates on a higher level are created by including (referring to) the templates on the lower level. The two important mechanisms that allow semi-automatic generation of use-case scenarios are parameters which can be passed and bound between templates and language transforms that are used to generate text in the natural language.

We call the proposed approach sketching a use case because it allows generating initial documentation (a sketch) of use-case scenarios, which can be further altered by an analyst (e.g., by providing values of parameters, modifying the generated text).

To evaluate the potential acceptance of the proposed approach, we developed a prototype tool called UC-Sketch and performed an acceptance forecasting study on 63 potential users, with the use of Technology Acceptance Model (TAM). The results of the study suggest that the target users of the proposed approach are willing to use it.

References

1. Neill, C., Laplante, P.: Requirements engineering: the state of the practice. IEEE Softw. **20**(6), 40–45 (2003)
2. Tiwari, S., Gupta, A.: A systematic literature review of use case specifications research. Inf. Softw. Technol. **67**, 128–158 (2015)
3. Jacobson, I.: Object-oriented development in an industrial environment. ACM SIGPLAN Notices **22**(12), 183–191 (1987)
4. Adolph, S., Bramble, P., Cockburn, A., Pols, A.: *Patterns for Effective Use Cases*. Addison-Wesley (2002)
5. Cockburn, A.: *Writing Effective Use Cases*. Addison-Wesley, Boston (2001)
6. Nawrocki, J., Nedza, T., Ochodek, M., Olek, Ł.: Describing business processes with use cases. In: Abramowicz, W. (ed.) Proceedings of the Business Information Systems Conference. Lecture Notes in Informatics, vol. P–85, pp. 13–27. Koellen Druck+Verlag (2006)
7. Issa, A., Al-Ali, A.: Use case patterns driven requirements engineering. In: 2010 Second International Conference on Computer Research and Development, pp. 307–313. IEEE (2010)
8. Watahiki, K., Saeki, M.: Scenario patterns based on case grammar approach. In: Fifth IEEE International Symposium on Requirements Engineering, 2001. Proceedings, pp. 300–301. IEEE (2001)
9. Issa, A., AlAli, A.: Automated requirements engineering: use case patterns-driven approach. IET Softw. **5**(3), 287–303 (2011)
10. Övergaard, G., Palmkvist, K.: *Use Cases: Patterns and Blueprints*. Addison Wesley Professional (2004)
11. Saeki, M.: Reusing use case descriptions for requirements specification: towards use case patterns. In: Software Engineering Conference, 1999. (APSEC'99) Proceedings. Sixth Asia Pacific, pp. 309–316. IEEE (1999)
12. och Dag, N.J., Regnell, B., Carlshamre, P., Andersson, M., Karlsson, J.: Evaluating automated support for requirements similarity analysis in market-driven development. In: 7th International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ'01), pp. 190–201 (2001)
13. Kaindl, H., Smiałek, M., Nowakowski, W.: Case-based reuse with partial requirements specifications. In: 2010 18th IEEE International Requirements Engineering Conference (RE), pp. 399–400. IEEE (2010)
14. Rolland, C., Souveyet, C., Achour, C.: Guiding goal modeling using scenarios. IEEE Trans. Softw. Eng. **24**(12), 1055–1071 (1998)
15. Ridao, M., Doorn, J., do Prado Leite, J.: Domain independent regularities in scenarios. In: Fifth IEEE International Symposium on Requirements Engineering, 2001. Proceedings, pp. 120–127. IEEE (2001)

16. Rolland, C., Achour, C.: Guiding the construction of textual use case specifications. Data Knowl. Eng. **25**(1), 125–160 (1998)
17. Ochodek, M., Nawrocki, J., Kwarciak, K.: Simplifying effort estimation based on use case points. Inf. Softw. Technol. **53**(3), 200–213 (2011)
18. Diev, S.: Software estimation in the maintenance context. ACM SIGSOFT Softw. Eng. Notes **31**(2), 1–8 (2006)
19. Ochodek, M., Alchimowicz, B., Jurkiewicz, J., Nawrocki, J.: Improving the reliability of transaction identification in use cases. Inf. Softw. Technol. **53**(8), 885–897 (2011)
20. Davis, F.: Perceived usefulness, perceived ease of use, and user acceptance of information technology. MIS Q. pp. 319–340 (1989)
21. Spearman, C.: "General intelligence," objectively determined and measured. Am. J. Psychol. **15**(2), 201–293 (1904)

Domain Modeling Based on Requirements Specification and Ontology

Iwona Dubielewicz, Bogumiła Hnatkowska, Zbigniew Huzar and Lech Tuzinkiewicz

Abstract Domain model plays an important role in software development. Typically, it is a primary input to elaboration of a system model which in turn is translated into source code and related database schemas. Effective development of domain model is a part of requirement engineering during which domain experts are employed to identify domain entities and relationships among them. We claim that this task can be supported by the use of domain ontologies from which interesting knowledge can be extracted. The starting point to knowledge extraction is an existing requirements specification. In this paper, we investigate how the form of requirements specification influences the quality of extracted model. Some measures allowing to assess the quality are introduced. A case study has shown that in the most cases the simplified version of a requirements specification is enough to obtain a satisfactory domain model, however if the domain is very complex, the extended version of requirements specification could be necessary.

Keywords Ontology · Requirements specification · Domain modeling · Knowledge extraction

I. Dubielewicz · B. Hnatkowska (✉) · Z. Huzar · L. Tuzinkiewicz
Faculty of Computer Science and Management, Wrocław University of Science and Technology, Wyb. Wyspiańskiego, 27, 50-370 Wrocław, Poland
e-mail: Bogumila.Hnatkowska@pwr.edu.pl

I. Dubielewicz
e-mail: Iwona.Dubielewicz@pwr.edu.pl

Z. Huzar
e-mail: Zbigniew.Huzar@pwr.edu.pl

L. Tuzinkiewicz
e-mail: Lech.Tuzinkiewicz@pwr.edu.pl

L. Madeyski et al. (eds.), *Software Engineering: Challenges and Solutions*,
Advances in Intelligent Systems and Computing 504,
DOI 10.1007/978-3-319-43606-7_3

1 Introduction

Incremental development is one of the approaches to software systems development. The software system is developed as a series of increments (builds, releases). Each new increment adds some functionality to the previous one. Specification, development, and validation are activities interleaved within single increment preparation. Incremental development seems to be the most common approach for software systems, and it is applied both in plan-driven (especially iterative) and agile methodologies. Its popularity owes to their advantages over classic waterfall model: reduced cost of implementation of changing requirements; feedback from customers during development work, and early delivery of running software.

Software development process is initiated by a real problem that occurs in a given application domain. The software system is intended to bring a solution or at least to support the problem solution. The software developer must first understand the problem, and, to this end, must have adequate domain knowledge. Usually, the needed knowledge is delivered by domain experts. In the paper, it is assumed that this knowledge can be extracted from a domain ontology. Recall that [12]: *An ontology is a formal* (*machine-readable*), *explicit* (*consensual knowledge*) *specification* (*concepts, properties, relations, functions, constraints, axioms are explicitly defined*) *of a shared conceptualization* (*abstract model of some phenomenon in the world*). Further, we assume that it is available a consistent and complete domain ontology respective to the given problem.

The domain problem is usually informally outlined. The domain problem is described by requirements specification. The requirements specification is the base for system model design, and next for a code implementation. Symbolically, this process is depicted as follows (Fig. 1).

In the proposed approach, we demonstrate that the transition from the requirements specification to the domain model, which usually is performed by a business analyst in collaboration with domain experts, may be done semi-automatically with the use of a domain ontology. It means that the knowledge of the domain expert may be replaced by the knowledge contained in respective domain ontology provided that this ontology is consistent and complete. Of course, as in the traditional approach, the domain model should be validated by a domain expert. In the series of papers [3–5], we have presented the idea of this approach. Additionally, a set of programming tools, supporting the approach, have been developed [4, 7].

In this paper, we demonstrate how a form of requirements specifications may influence the resulting domain model. Our aim is to assess how the form and

$$\begin{bmatrix} requirements \\ specification \end{bmatrix} \xrightarrow{\substack{Domain \\ expert}} \begin{bmatrix} domain \\ model \end{bmatrix} \xrightarrow{\substack{Software \\ designer}} \begin{bmatrix} system \\ model \end{bmatrix} \xrightarrow{Programmer} \begin{bmatrix} source \\ code \end{bmatrix}$$

Fig. 1 Basic artifacts within development process

granularity of requirements specification influences the quality of the domain model obtained through knowledge extraction process from a given ontology. There are introduced some measures enabling assessment of this influence.

In Sect. 2 we present the forms of requirements specification assumed to be inputs to the knowledge extraction from existing ontologies. In Sect. 3 we shortly describe the extraction process as well as propose measures to access its results. Section 4 presents a case study, in which we try to answer the question if and how the form of requirements specification influences the quality of the resulting domain model. Section 4 gives the found conclusions and some comments on future works.

2 User Requirements Specifications

The notion of "requirement" has not unique meaning. According to IEEE [9], it is "*A condition or capability needed by a stakeholder to solve a problem or achieve objectives*". It can be found a variety of taxonomies for requirements, but the interesting one is this where requirements are defined for the product development and for the process which leads to this product development. As in the paper we concentrate on the product development process we consider taxonomy given in BABOK® [1]: business requirements, stakeholder (user) requirements, solution (or system) and transition requirements. Another requirements taxonomy is proposed in ISO/IEC 25030:2007 [10] but it is limited only to software system requirements.

In practice, the distinction between them remains problematic—while the business requirements are usually well discriminated from the others, distinguishing between user and system requirements is much more difficult.

Independently of software development approach, the requirements definition process is the of primary importance. In general, it consists of three sub-processes: requirement elicitation, analysis and specification (they all are generalized to the concept "requirement engineering") [2, 14]. Nowadays commonly used software development models are based on incremental and, more often adaptive approaches. Activities performed within one increment (release) form a kind of a mini-waterfall model. It means that the requirements for the release should be completely defined before the design of that release, and thus, the whole requirements elicitation process is performed iteratively.

Each new increment brings its releasable product (release). Within each increment, requirements are documented and analyzed. The specification of elicited requirements takes a form of plain text or sometimes more formal form—an use case model. They are expressed by a mixture of notions from the IT and business areas as in the iterative approach it is easy to cross the border between stakeholder and system requirements. The analysis aims at deciding whether the notions used in the requirements documentation are well-understood by all stakeholders and whether these notions are consistent to the given application domain. In this analysis, a domain model elaborated on the base of the requirements specification and a respective domain ontology may be used.

A specific approach to requirement elicitation, analysis, and specification has been proposed in adaptive (agile) methodologies. The requirements specification takes a form called *User stories* or eventually *Epic* (just a large user story that needs to be broken down into) and represent a high-level description of a capability that the system needs to provide [2]. They are expressed only with the business notions and served as a basis for effort/cost estimation. After once picked up for being worked on, the user story is described with a lot of details given directly by the future user (and thus some IT notions can appear in its description). It only happens when they start to be subject to implementation. At that time also the acceptance criteria for the user stories are defined for they could be tested as “done” at the end of the iteration.

Another approach can be seen in the Use Case v.2.0 concept proposed by Jacobson [11]. For matching the incremental (precisely-agile) approach, he has suggested defining the use case scenarios progressively. Instead of specifying all scenarios at the beginning one can select (and define and implement) only some of them according to the order specified by the user (“a priority driven” approach). The scenarios specified in such a way has been called *use case slices*, and they correspond to the user stories defined with many details.

Further, we assume that the requirements specification is expressed in the form of user stories, usually presented in the context of a system vision [13]. User stories are presented as statements in a standardized “role-feature-reason” form. Exemplary variants of the forms are:

As a <type of user>, I want <some goal> so that <some reason>.
As a <role>, I want <goal/desire> so that <benefit>.

Description of each user story may be augmented with test cases. There are many schemas of test case description. We have chosen the one marked “*given-when-then*” which has the form:

GIVEN: Set up the object to be tested
WHEN: act on the object
THEN: Make claims about the object, its collaborators/parameters or global state.

Summarizing, we assume that requirements specification *rs* is defined as:

$$rs = \{ < h_1, tc_1 >, \ldots, < h_n, tc_n > \} \tag{1}$$

where $h_1, \ldots, h_n$ are user stories (histories), and $tc_1, \ldots, tc_n$ are sets of test cases. Both user stories and test cases may be presented using above notations.

3 Towards Domain Model

Practical approach to requirements specification begins with the initial list of user stories $h_1, \ldots, h_n, \ldots$. The stories are ordered according to their decreasing priorities. On the base of the arbitral decision of an specifier, first n stories with the highest priority are selected to form first user requirements specification. This specification usually contains only user stories, but for the sake of unique notation, it will be written as in (1), where the sets of test cases may be empty.

The requirements specification, as a set of documents in natural language, contains some words or phrases that represent notions related to a given problem within an application domain. In the presented approach [8], we concentrated on nouns and noun phrases as the most likely elements relating to a structural aspect of the considered problem. The set of these elements is called a set of initial domain notions.

The specification *rs* contains a set of initial domain notions *DSN*, which some of them are domain and other are systems' or native notions. *DNS* forms a glossary of terms within a considered domain. The expert should define a function $EQ\colon DNS \rightarrow ONT$, which maps glossary terms into their equivalent ontology concepts *ONT*. We assume that such domain ontology relevant to the considered problem is given. The ontology should be consistent; additionally, its completeness is strongly required.

The *EQ* function is usually a partial function because only the terms which represent domain notions may have equivalents in the set of ontology concepts; the domain of *EQ* (*dom*(*EQ*)) is marked in Fig. 2 by the gray oval within *DSN* while the codomain of *EQ* (*ran*(*EQ*)) is marked by the gray oval within ONT.

A well-defined domain ontology should rather not contain specific notions related to information systems.

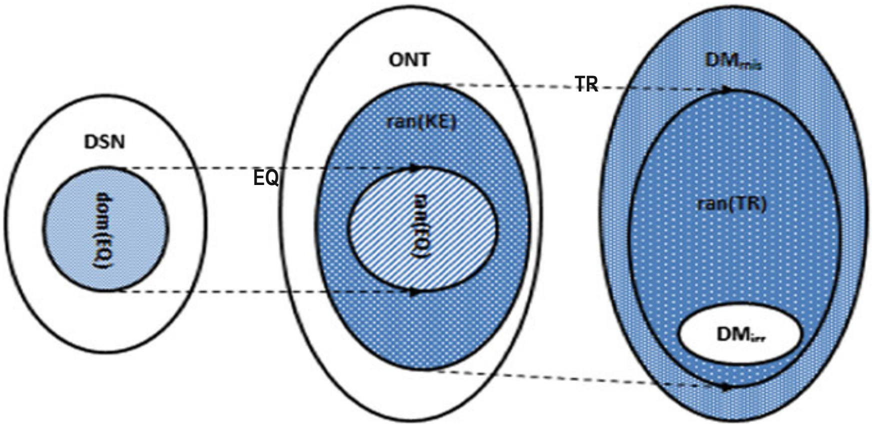


Fig. 2 Schema of transformations from a requirements specification to a domain model

The value:

$$coverage = \frac{card(dom(EQ))}{card(DNS)} \tag{2}$$

where *card*(*X*) means cardinality of the set *X*, is the measure how terms of the glossary are covered by ontology notions.

The set of ontology notions defined by the *EQ* function, i.e. the set representing its range *ran*(*EQ*), is the base for extracting additional knowledge from the ontology *ONT*. The additional notions are extracted by knowledge extraction algorithm [8], which abstractly is represented as a function $KE(ran(EQ), ont) \subseteq ONT$. The set of ontology notions extracted from the ontology is represented in Fig. 2 as a range of the function *KE*, i.e. *ran*(*KE*). Informally, *ran* (*KE*) may be called a sub-ontology of the ontology *ONT*. Of course, this set contains $ran(EQ)$ as its subset.

The value:

$$enrichment = \frac{card(ran(KE) \backslash ran(EQ))}{card(ran(KE))} \tag{3}$$

may be treated as a measure how the ontology enriches the initial mapping of glossary terms.

The set *ran*(*KE*), represented by the dark gray oval in Fig. 2, is a base for automatic UML class diagram generation [3]. The algorithm generating a class diagram is abstractly represented as a function $TR(ran(KE))$. The generated class diagram *ran*(*TR*) is represented in Fig. 2 by dark gray oval, a subset of *DM*. *DM* represents a final domain model, also represented in the form of UML class diagram, which is obtained by modification of *ran*(*TR*). The modification is the result of two activities performed by an analyst/domain experts. Within these activities, it is decided which elements of the *ran*(*TR*) on the class diagram are irrelevant, and which elements are missing with respect to the requirements specification.

An element $e \in ran(TR)$ is said to be irrelevant to the requirements specification if *e* is not used in the description of any user story or any test case. The set of irrelevant and missing elements are denoted by DM_{irr} and DM_{mis}, respectively. The following are measures of irrelevance:

$$irrelevance = \frac{card(DM_{irr})}{card(ran(TR))} \tag{4}$$

and missing:

$$missing = \frac{card(DM_{mis})}{card(ran(TR) \backslash DM_{irr}) + card(DM_{mis})} \tag{5}$$

In the incremental and iterative software life-cycle model, the requirements specification prepared in one iteration may be modified by extension or refinement in the next iteration. Requirements specification rs_1 may be developed in subsequent iterations. The specification rs_2 is an extension of rs_1 if $rs_2 = rs_1 \cup < h_{n+1}, tc_{n+1} > \ldots\}$. It means that new user stories are added to the previous specification rs_1. The specification rs_2 is a refinement of rs_1 if some test case tc_i assigned to the user story h_i (i = 1, ..., n) from rs_1 is replaced by $tc_i^{'}$ such that $tc_i \subset tc_i^{'}$. In practice, one modification step may contain both requirements extension and its refinement.

Summarizing, after a series of iterations, we have a sequence of requirements specifications: $rs_1, rs_2, \ldots, rs_n$ accompanying with a sequence of respective sub-ontologies: $ont_1, ont_2, \ldots, ont_n$, and a sequence of domain models: $dm_1, dm_2, \ldots, dm_n$.

The way how requirements specifications are constructed shows that $DNS_i \subseteq DNS_{i+1}$. Therefore also $ONT_i \subseteq ONT_{i+1}$, where ONT_i is the set of ontology notions for the sub-ontology ont_i, and $DM_i \subseteq DM_{i+1}$, where DM_i is the set of domain model notions for the domain model dm_i.

Now the question begs: how, in the context of the ontology, the form of rs influences the quality of dm. It is difficult to answer the question directly, but we may examine how the increment of the set of initial domain notions influences the set of ontology and domain model notions. For example, if for some i = 1, 2, ..., n the condition holds:

$$\frac{card(DNS_{i+1})}{card(DNS_i)} > \frac{card(ONT_{i+1})}{card(ONT_i)} > \frac{card(DM_{i+1})}{card(DM_i)}$$

it may be interpreted that each subsequent requirements specification has a weaker influence on the subsequent domain model.

It should be noted that from the domain expert perspective, it would be not convenient to prepare a domain model in incremental approach by creating a sequences of sub-ontologies. It is rather suggested that the sub-ontology should be created only ones on the base of the most developed requirements specification. Of course, the postulate of incremental system model and software release development remains still valid.

4 Case Study

The aim of the case study is to answer the question if and how the form of requirements specification influences the quality of the domain model obtained through knowledge extraction process from existing ontologies. We would like to know if a domain model of acceptable quality can be retrieved from existing ontologies at early stages of software development, when requirements specification doesn't contain many details, or if we should refine as many requirements as possible to obtain better results.

The study starts with a simplified version of requirements specification which consists of two user stories with accompanying test cases (see Sect. 4.1). The user stories refer to hotel reservation domain where a potential customer wants to browse a hotel offer and later, to check the availability of specific room type in a given period. Next, we follow the same extraction procedure twice, for two forms of requirements specification—the first contains only user stories (see Sect. 4.2), while the second also test-cases (see Sect. 4.3). The domain ontologies from which the knowledge is extracted are included into the following SUMO [15] files: Merge.kif, Mid-level-ontology.kif, Hotel.kif, Dining.kif, Catalog.kif. The first two files are upper ontologies while the last two are domain ontologies referenced by Hotel.kif. The models obtained in both cases are modified: missing elements (according to the test cases) are added, and irrelevant found out and marked. The proposed metrics are calculated what allows us to derive some conclusions.

4.1 Case Study—Requirements Specification

This subsection contains the definition of two user stories accompanied by test cases. Test cases are written in Gherkin language—"a business readable, domain specific language that lets you describe software's behavior without detailing how that behavior is implemented" [6].

User Story 1: As a *potential customer*, I want to see information about *hotel,*[1] *hotel rooms*, *rooms' amenities* and *prices* so that I can decide whether to become a *customer*.

Test cases for User Story 1:
Scenario 1: Basic information about hotel
Given that a hotel is defined with: <*name*>, <*postal address*>, and <*category*>
When a customer navigates to the main hotel page
Then he/she should be informed about hotel <name>, <postal address>, and <category>.

Examples (basic information about hotel):

Name	Postal address	Category
Hostel	Bema Street 5, Wroclaw, Poland	***

[1]Domain notions are written in italics.

Scenario 2: List of hotel rooms
Given that a hotel rents rooms of types defined with: <*room type*>, <*capacity*>, <*price per night*>, <*amenities*>
When a customer navigates from main hotel page to the list of room types page
Than he/she should be informed about the hotel room types.

Examples (types of hotel rooms):

room type	capacity	price per night	amenities
single	1	230 zł	TV, sejf
double	2	300 zł	balcony, TV, sejf

User Story 2: As a potential customer, I want to check availability of selected *room type* (*room availability*) in a given *reservation period* so that to be able to decide if to make *reservation* or not.

Test cases for User Story 2:
Scenario 1: No room of a given room type is available in selected period
Given that a hotel has reservations for rooms of <*room type*>
And that <*room type*> is reserved from <*dateFrom*> to <*dateTo*>
And that the hotel has <*nr instances*> of rooms of <*room type*>
When a customer asks for the <*room type*> availability from <*given dateFrom*> to <*given dateTo*>
Than he/she is informed that no room of specific <*room type*> is available from <*given dateFrom*> to <*given dateTo*>.

Examples:

Input:
room type: single given dateFrom: 1-1-2016 given dateTo: 2-1-2016

		reservation period	
room type	no instances	dateFrom	dateTo
single	2	1-1-2016	3-1-2016
		31-12-2015	2-1-2016

Scenario 2: There is a room of a given room type available in selected period
Given that a hotel has reservations for <*room type*>
And that <*room type*> is reserved from <*dateFrom*> to <*dateTo*>
And that the hotel has <*nr instances*> of rooms of <*room type*>
When a customer asks for <*room type*> availability from <*given dateFrom*> to <*given dateTo*>
Then he/she is informed that a room of <*room type*> is available from <*given dateFrom*> to <*given dateTo*>.

Examples:

Input:
room type: single given dateFrom: 1-1-2016 given dateTo: 2-1-2016

		reservation period	
room type	no instances	dateFrom	dateTo
single	2	1-1-2016	3-1-2016
		31-12-2015	1-1-2016

4.2 Domain Model Built Basing on the User Stories

This subsection describes the inputs/outputs of the functions involved in the extraction process. We start from the definition of *DNS* set:

DNS = {*hotel*, *hotel room, room amenity, price per night, room availability, reservation, reservation period, customer, potential customer*}.

The notions that, according to a domain expert, describe business notions at that moment, are mapped by him to SUMO ontology. List of mappings from *DSN* to *ONT* (*EQ* function) is as follows:

{*hotel* → *HotelBuilding*, *hotel room* → *HotelRoom, room amenity* → *roomAmenity, price per night* → *price, room availability* → *reserved-Room, reservation* → *HotelReservation, reservation period* → *(reservationStart, reservationEnd)*}.

It can be noticed that two of *DNS* notions are treated as the system notions and are not mapped to the ontology notions.

Further the ontology notions are used to extract additional knowledge from ontology (with *EQ* function) and to translate extracted notions to a UML class diagram (with *TR* function). The translation maps each ontology element into an appropriate UML element (the mapping is one to one). Figure 3 presents the results of automatic UML class diagram generation. Please note, that *HotelUnitType*, and *PhysicalType* are UML *PowerTypes*.

After generation, the domain model is investigated by a system analyst to find out missing and irrelevant elements. Missing elements are presented in Fig. 4 in bold (relations) or are written with Courier font (attributes, classes), while irrelevant elements are drawn by dashed line (classes) or by thinner lines (relations). The generated domain model doesn't contain many details that are necessary to realize all test cases, e.g. we lack the definition of properties like hotel name or category as well as the definition of some relationships, e.g. hotel building consists of hotel units. There is only one irrelevant element—we don't need to know who has defined a price for a hotel unit.

The values of basic measures for that scenario are listed in Table 1.

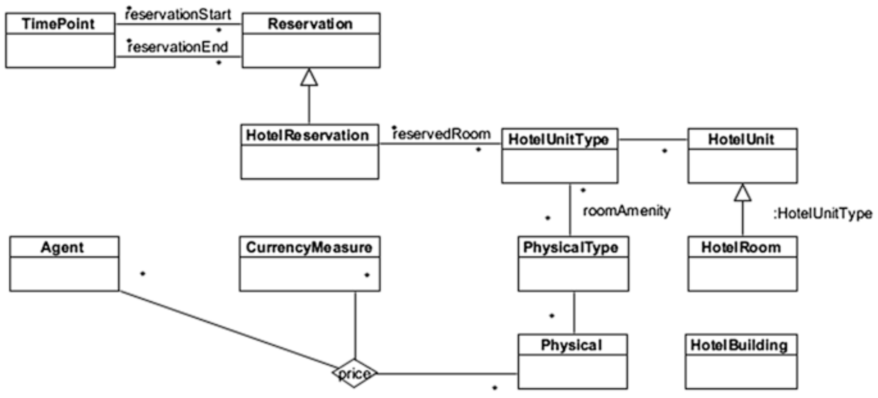


Fig. 3 Automatically obtained domain model (according to Scenario 1)

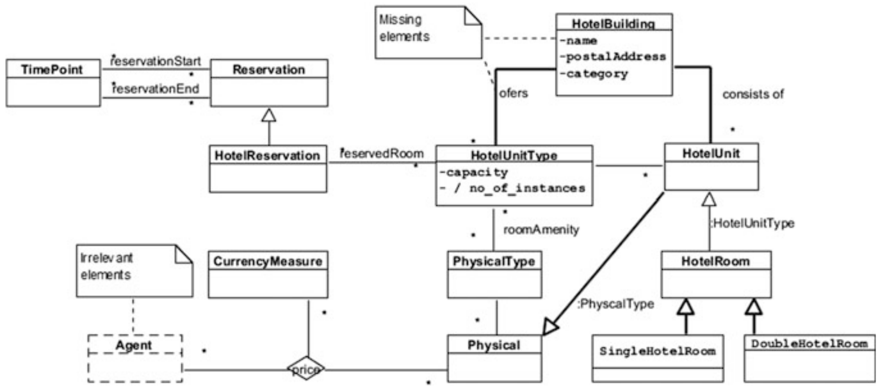


Fig. 4 Revised version of domain model (according to scenario 1)

Table 1 Basic measures counted for scenario 1 (columns refer to cardinality of defined sets)

DNS	dom (EQ)	ran (EQ)	ran (KE)	ran (TR)	DM_{irr}	DM_{mis}
9	7	8	20	20[a]	1	13[b]

[a]11 classes, 7 relations, 2 inheritance relationships
[b]2 classes, 5 attributes (1 unnecessary – \no of instances), 2 relations, 3 inheritance rel

4.3 Domain Model Built Basing on the User Stories and Test Cases

Similarly, to the previous section we start with the definition of *DNS* set. In comparison to the previous case, it contains additional notions, e.g. name, and address.

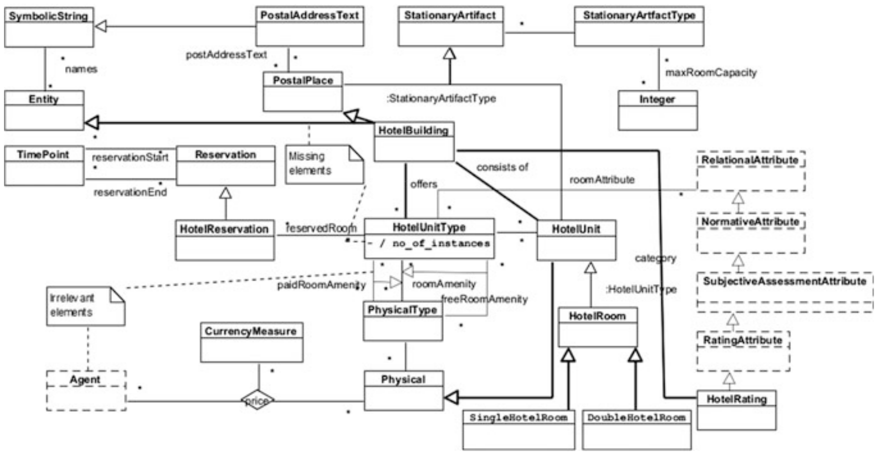


Fig. 5 Revised version of domain model (according to Scenario 2)

Table 2 Basic measures counted for scenario 2 (columns refer to cardinality of defined sets)

DNS	dom (EQ)	ran (EQ)	ran (KE)	ran (TR)	DM_{irr}	DM_{mis}
13	11	12	47	47[a]	10[b]	9[c]

[a]Classes, 14 relations, 10 inheritance (2 between relations)
[b]5 classes, 3 relations, 2 inheritance relations
[c]2 classes, 1 attribute (unnecessary: "\no of instances"), 1 relation, 5 inheritance relationships (most are defined in ontology)

DNS = {*hotel, hotel room, room amenity, price per night, room availability, reservation, reservation period, name, address, category, capacity*}.

The list of mappings from *DSN* to *ONT* (*EQ* function) is as follow:

{*hotel* → *HotelBuilding, hotel room* → *HotelRoom, room amenity* → *roomAmenity, price per night* → *price, room availability* → *reservedRoom, reservation* → *HotelReservation, reservation period* → *(reservationStart, reservationEnd), name* → *names, address* → *postAddressText, category* → *HotelRating, capacity* → *maxRoomCapacity*}.

Figure 5 presents a refined version of UML class diagram—both missing and irrelevant elements are marked here in the same way as in Fig. 4.

The values of basic measures for that scenario are listed in Table 2.

4.4 Comparison

In this section, we present the values of derived measures calculated for both scenarios of our case study—see Table 3.

Table 3 Derived measures calculated for both scenarios

Measure	Scenario 1: user stories	Scenario 2: user stories + test cases
Coverage	0.77	0.84
Enrichment	0.60	0.74
Irrelevance	0.05	0.21
Missing	0.41	0.19

When our *DNS* set contains more elements (scenario 2) more of them are considered to be domain notions (84 %) in comparison to the scenario 1 (77 %). It seems that this observation should be valid also in the context of incremental development, where, in an increment, a new user-stories are added or refined. Scenario 2 also allows extracting more elements from existing ontology (enrichment is equal to 74 % in comparison to 60 %). The extraction process is not ideal as it produces some irrelevant elements. The likelihood that irrelevant elements will appear is higher for scenario 2 (irrelevance is equal to 21 % in comparison to 5 %). On the other hand, the number of missing elements decreases when the input is richer (missing is equal to 19 % in comparison to 41 %).

The case study allows us to derive some preliminary conclusions. For the simple domains it should be enough to start with user stories only. A domain model resulting from the extraction procedure should be not complicated (only a few irrelevant elements), and relatively easy for extension by a domain expert. The missing elements can be added iteratively, on demand. For a complex domain, it is worth to invest with a more detailed requirements specification, which allows to obtain a domain model with less number of missing elements but possibly some out of scope. It is easier to cross out unnecessary parts and have an opportunity to familiarize deeper with the domain (from the perspective of a business analyst) than to ask domain experts for identification of all lacking elements.

5 Conclusions

The aim of the paper is to present an approach to the knowledge extraction from an ontology to create appropriate domain model and to evaluate the influence of the requirements specification on the quality of the resulting model. The main problems analyzed and discussed in this work are related to the evaluation of the possibility and potential benefits of using ontology in the process of creating domain models in the context of defined requirements (given in the form of the user stories). Defined measures: *enrichment*, *irrelevance*, and *missing*, allow to evaluate the quality of the extracted domain model.

Examples of knowledge extraction described in the case study show that the created domain model can contain redundant elements (*irrelevance* > 0). On the other side, one cannot assure that the resulting model fully meets user expectations (*missing* > 0). Everything depends on the quality of requirements specification as well as the completeness of the selected ontology. In most cases the simplified version of requirements specification is enough to obtain a satisfied domain model, however, if the domain is very complex, an extended version of requirements specification could be necessary.

Expected benefits of the proposed approach is to reduce the commitment and costs of participation of experts in the process of domain knowledge extraction. It is possible due to a partial automation of domain modeling process using ontology, and the permanent access to the knowledge domain by IT developers. Experts responsibilities may be limited to:

- mapping of *DNS* notions to ontology concepts,
- validation and possibly extension of created domain model.

The other benefit is a better quality of resulting domain model, which—by construction—is consistent with ontology from which it was created.

Future works are concentrated on experiments aiming at improving effectiveness and precision of the knowledge extraction algorithm from ontology which is used in the process of generating domain models. Moreover, the results of experiments with SUMO ontology challenged our assumption about the completeness of this ontology and therefore it is also planned utilization of OWL ontologies in further research.

References

1. A Guide to the Business Analysis Body of Knowledge® (BABOK® Guide) v2
2. Cobb, Ch.G.: Making Sense of Agile Project Management: Balancing Control and Agility. Wiley (2011)
3. Dubielewicz, I., Hnatkowska, B., Huzar, Z., Tuzinkiewicz, L.: Domain modeling in the context of ontology. Found. Comput. Decis. Sci. **40**(1), 3–15 (2015)
4. Dubielewicz, I., Hnatkowska, B., Huzar, Z., Tuzinkiewicz, L.: Development of domain model based on SUMO ontology. In: Zamojski, W., et al. (eds.) Proceedings of the 10th International Conference on Dependability and Complex Systems DepCoS-RELCOMEX, pp. 163–173. Springer (2015)
5. Dubielewicz I., Hnatkowska B., Huzar Z., Tuzinkiewicz L.: Problems of SUMO-like ontology usage in domain modelling. In: Nguyen, N.T., et al. (eds.) 6th Asian Conference, ACIIDS 2014, Lecture Notes in Computer Science, vol. 8397, pp 352–363, Springer (2014)
6. Gherkin: http://docs.behat.org/en/v2.5/guides/1.gherkin.html
7. Hnatkowska, B.: Towards automatic SUMO to UML translation. In: Kościuczenko, P., Śmiałek, M. (eds.) From Requirements to Software, Research and Practice, pp. 87–100. Polskie Towarzystwo Informatyczne (2015)
8. Hnatkowska, B., Dubielewicz, I., Huzar, Z., Tuzinkiewicz, L.: Conceptual modeling using knowledge of domain ontology. In: Nguyen, N.T., et al. (eds.) 8th Asian Conference, Intelligent Information and Database Systems, Proceedings, Part II, pp. 558–566. Springer (2016)
9. ISO/IEC/IEEE 29148-2011—Systems and software engineering—Life cycle processes—Requirements engineering. 2011
10. ISO/IEC 25030:2007 Software engineering—Software product Quality Requirements and Evaluation (SQuaRE)—Quality requirements

11. Jacobson, I.: Use Case 2.0. A Guide to Succeeding with Use Cases. https://www.ivarjacobson.com/sites/default/files/field_iji_file/article/use-case_2_0_jan11.pdf. Accessed 10 Apr 2016
12. Studer, R., Benjamins, V., Fensel, D.: Knowledge engineering: principles and methods. Data Knowl. Eng. **25**, 161–197 (1998)
13. Wikipedia: Vision document—wikipedia, the free encyclopedia, 2015. Accessed 4 Apr 2016
14. Wikipedia: Requirements engineering—wikipedia, the free encyclopedia, 2016. https://en.wikipedia.org/wiki/Requirements_engineering. Accessed 4 Apr 2016
15. Wikipedia: Suggested upper merged ontology—wikipedia, the free encyclopedia, 2016. Accessed 10 Apr 2016

Semantic Validation of UML Class Diagrams with the Use of Domain Ontologies Expressed in OWL 2

Małgorzata Sadowska and Zbigniew Huzar

Abstract The article proposes an algorithmic method for semantic validation of UML class diagrams. The method checks the compliance of the diagrams with the field described by the domain ontology expressed in OWL 2. More specifically, it allows for an automatic validation if all diagram elements and their relationships are contained or at least are not contradictory with the domain knowledge extracted from the selected domain ontology. A semantic correctness of UML class diagrams can be partly validated without involving domain experts in the process of validation of the diagrams.

Keywords UML · OWL 2 · Semantic validation

1 Introduction

A Unified Modeling Language (UML) [1] is a popular and commonly used modelling standard. It defines several types of diagrams among which UML class diagrams are in focus of this article. UML class diagrams are used to describe a static structure of a system [2]. The aim of this article is to present a method for an automatic validation of UML class diagrams if they are compliant with the field described by the domain ontology. Lindland et al. [3] in the framework for model quality distinguished between pragmatic quality (comprehensibility for the intended users), syntax quality (adhering to the rules in the language) and semantic quality (correct meaning and relations) with its two goals: validity and completeness. Lindland et al. [3] describe validity as an indicator if all statements made by the model are correct and relevant to the problem.

M. Sadowska (✉) · Z. Huzar
Faculty of Computer Science and Management, Wroclaw University of Science and Technology, Wrocław, Poland
e-mail: m.sadowska@pwr.edu.pl

Z. Huzar
e-mail: zbigniew.huzar@pwr.edu.pl

L. Madeyski et al. (eds.), *Software Engineering: Challenges and Solutions*,
Advances in Intelligent Systems and Computing 504,
DOI 10.1007/978-3-319-43606-7_4

In the proposed method, the UML is supported by the OWL 2 Web Ontology Language (OWL 2) [4], which is a popular knowledge representation language for defining ontologies. Despite the fact that UML and OWL 2 languages have different purposes, they also have many similar or equivalent elements (e.g. [5, 6]). For example, both languages have a concept of a Class and a Datatype, in UML one can specify multiplicity and in OWL 2 cardinality, UML offers class generalization and OWL 2 contains SubClassOf axiom, and so on. The similarities allow for translating UML class-based modelling into description logic, which gives UML class-based modelling a model-theoretic semantic [6].

OWL 2 ontologies consist of entities, expressions and axioms [4]. Entities (classes, properties and individuals) form primitive terms of an ontology, expressions (e.g. a class expression) represent complex notions and axioms are used to provide information about classes and properties (e.g. a subclass axiom). Direct model-theoretic semantics of OWL 2 is compatible with the description logic SROIQ [4]. Following [7], SROIQ is a fragment of first order logic with useful computational properties so that it can be exploited by OWL 2 tools. Ontologies that can be translated into a SROIQ knowledge base are called OWL 2 DL ontologies [7], and are the focus of this article.

UML [1] modelling language is semi-formal because it has a formally defined syntax using a subset of UML and informally defined semantics in natural language. The semantics in UML class diagrams have a reference to a selected reality and describes meaning of the used terms (classes and their relationships). OWL 2 [4] is a formal language with a model-theoretic semantics. However, the semantics of ontologies expressed in OWL 2 have a relation to the entities in the specific domain, similarly as it is in case of UML class diagrams. In this article, the concept of semantics refers to the elements from both descriptions (UML class diagram and OWL 2 ontology) with respect to the same domain of application.

The UML class diagram is an essential part of designing a compound business model [8]. Due to this fact, it is especially important if the diagram is compliant with the needed domain reality. A domain knowledge can be obtained from a number of sources. For example, it can be provided by domain experts, described in different documents, or contained in domain ontologies (as it is proposed in this article). In our approach UML class diagram needs to be confronted with the ontology expressed in OWL 2.

The approach proposed in this article does not assume that the designed class diagram is complete and is not able to state whether the diagram correctly addresses the user's needs. However, the approach will verify if all diagram elements and the relationships among the elements are contained (or not) in the domain knowledge. From this perspective, the method will automatically validate the model semantics if it is correct in accordance with the domain, and therefore if the designed diagram is meaningful.

The rest of the paper is organized as follows: Sect. 2 presents related works, Sect. 3 introduces necessary definitions and gives the outline of the proposed method of semantic validation of UML class diagrams, Sect. 4 concludes the paper and presents an outlook for future work.

2 Related Works

Transformation rules that convert UML class diagrams to OWL 2 representation are important in the presented approach to model validation against the domain they describe. A number of publications: [5, 9–15] present transformation rules of selected elements of UML diagrams to OWL 2. The transformation rules cover mapping of the most important elements of a UML class diagrams, i.e. classes with attributes, associations and generalizations. A revision and extension of the state of the art transformation rules for UML class diagrams is not in the scope of the paper and is planned as future work. In [12], the rules are used to check the internal consistency between several UML diagrams. Felfernig et al. [16] define transformation rules from the conceptual model to the logical model. More specifically, Felfernig et al. [16] define a logic based formal semantic for UML constructs, which allow to generate logical sentences and to process them by a problem solver.

Some approaches of manual validation of UML class diagrams are described in the literature. Unhelkar [17], presents three traditional quality techniques which are used to validate semantics of UML models. The techniques are: walkthroughs, inspections and reviews. All of them require judgement of domain experts. Following Unhelkar [17], a walkthrough is the least rigorous method and is more helpful to detect syntax errors rather than semantic ones. Inspections can identify both syntax and semantic errors. Also, Lindland et al. [3] and Bertolino et al. [18] indicate that validation if the model correctly captures the intended domain knowledge mostly entails manual inspection of the model. In [17], reviews are explained as a method with increased formality and focused mostly on detecting semantics and aesthetics problems in the model. Bertolino et al. [18] present a method and a tool for model validation, which generate a set of yes/no questions from the model. The approach still requires judgement of the domain expert in every case but the validation process is partially automated.

Letelier and Snchez [19] propose a method of validation of UML classes through animation and present a preliminary tool supporting the method. The tool generates a prototype from the conceptual model and executes scenarios obtained from stakeholders. When the prototype is started the behaviour of objects may be examined by observing the occurring actions and the reached states. As a result, one can compare the expected behaviour from the scenarios with the obtained result and correct the initial model, if needed.

3 Method of Validation

A necessary preliminary requirement before the method can be applied is that the UML class diagram and the domain ontology must follow one agreed domain vocabulary. Additionally, the domain ontology must be consistent, as it will serve as a knowledge base for the application area. Ontology consistency is one of the infer-

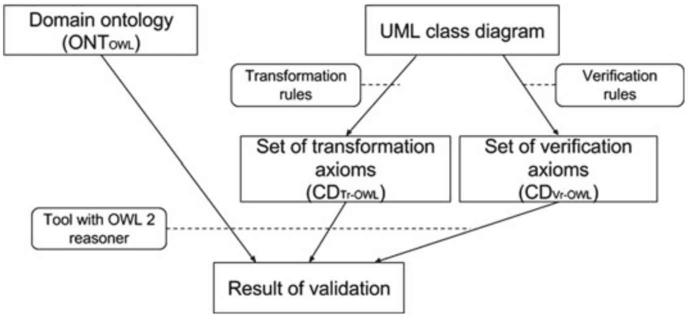


Fig. 1 Illustration of the method of semantic validation of UML class diagrams

ence problems that can be answered by OWL reasoners [7]. A consistency reasoning task in [20] is defined as: "given an ontology, decide whether it is contradictory (i.e., has any models)". In the following sections, OWL always means OWL 2, if not stated differently.

The proposed method of semantic validation of a UML class diagram requires a translation of the UML class diagram to OWL representation. The translation of the diagram is conducted with the use of the so called transformation and verification rules (the relevant definitions and examples are presented in the following subsections). Both the domain ontology and the class diagram are presented in the same notation in the form of OWL axioms.

The validation method is graphically illustrated in Fig. 1. The figure at the top shows inputs to the validation method and at the bottom presents an output. The rectangles symbolize artifacts and the rounded rectangles stand for transitions. The transitions related to transformation and verification rules, assign to each element from the UML class diagram corresponding set of OWL axioms.

The proposed method of validation compares two sets of OWL axioms (the domain ontology and the OWL representation of UML class diagram) for their compliance. OWL representation of UML class diagram consists of two parts: the transformational part and the verificational part. The initial domain ontology is iteratively modified by adding, one by one, axioms from the transformational part of OWL representation of UML class diagram. The modified domain ontology at some point may contain not only the domain knowledge but also (in the last iteration) the knowledge from the new source of information i.e. the validated UML class diagram. Each new axiom added to the domain ontology entails a risk of making the modified domain ontology inconsistent. Therefore, after adding every new axiom to the modified domain ontology, a reasoner is run in order to check its consistency.

It should be underlined that the modified domain ontology is only treated as a working artefact, which is used check the compliance of the model with the original ontology. In particular, the finding that the modified domain ontology is not consistent means that the UML class diagram is not compliant with the domain ontology.

Table 1 UML class generalization with transformation rules

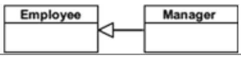
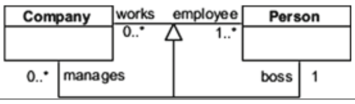
UML class diagram element	Class generalization
UML graphical symbol	
Transformation rules	SubClassOf(:Manager :Employee) Related works: [5, 10, 12, 13, 21]

Table 2 UML generalization between associations with transformation rules

UML class diagram element	Generalization between associations
UML graphical symbol	 Company works employee Person 0..* 1..* 0..* manages boss 1

Transformation rules	InverseObjectProperties(:works :employee)
	InverseObjectProperties(:manages :boss)
	SubObjectPropertyOf(:manages :works)
	SubObjectPropertyOf(:boss :employee)
	Related works: [5, 12, 13]

The important parts of the method are the transformation and verification rules. A result of applying rules to the UML class diagram are OWL axioms, here written with the use of the functional-style syntax [12]. The goal of the rules is to carry information from the UML class diagram to the domain ontology, in order to extend the knowledge contained in the original domain ontology. Due to the limited space, the article presents only two examples of the diagram elements with the defined transformation and verification rules (please refer to Tables 1, 2, 4 and 5). The tables include citations of the literature in which the selected transformation rules were initially introduced.

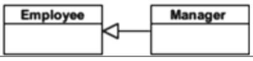
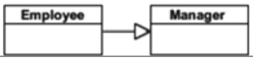
3.1 Transformation Rules

Definition 1 (*Transformation rule*) The transformation rule tr converts an element e of UML class diagram to a set $tr(e)$ of OWL axioms preserving semantics of the UML element.

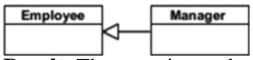
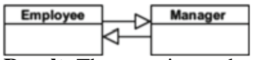
The set CD_{Tr-OWL} defined by Eq. 1 is called a transformational part of OWL representation of UML class diagram. CD_{Tr-OWL} constitutes a union of sets of results of applying transformation rules tr to all elements e of the UML class diagram CD.

$$CD_{Tr-OWL} = \bigcup_{e \in CD} tr(e) \qquad (1)$$

Table 3 Motivating example presenting the need for verification rules

Example extract from domain ontology	... SubClassOf(:Manager :Employee) ...	
Example ID	**Example 1**	**Example 2**
Example extract from UML class diagram		
Result of applying transformation rules from Table 1	SubClassOf(:Manager :Employee)	SubClassOf(:Employee :Manager)

Modified domain ontology (transformation axiom is added)	SubClassOf(:Manager :Employee)
	...
	(no new elements added)
		SubClassOf(:Manager :Employee)
		SubClassOf(:Employee :Manager)
		...
		(one new element added)

Result of consistency check of the modified domain ontology	**Result**: The modified domain ontology is consistent because no axioms were added	**Result**: The modified domain ontology is also consistent
Reengineering of the modified domain ontology to UML class diagram	 **Result**: The reengineered UML class diagram is correct	 **Result**: The reengineered UML class diagram is incorrect with respect to the semantics of the generalization relationship in UML

3.2 Motivating Example for Verification Rules

The following example aims to present the intention behind introducing verification rules. It shows that transformation rules themselves are not informative enough to validate UML class diagrams with the use of domain ontology. Table 3 contains two extracts from UML class diagrams and an extract from the domain ontology. The same domain ontology is used for both example diagrams. A last row in Table 3 presents a result of reengineering of the modified domain ontologies to UML class diagrams. The row is not a part of the method but is aimed to illustrate, what verification rules are and why they are needed in the proposed approach.

In the first example in Table 3, Manager class is generalized by *Employee* class. The transformation rule applied to this diagram results in the axiom: *SubClassOf(:Manager :Employee)*. The axiom, after being added to the domain ontology, does not change the ontology as the ontology previously contained this axiom. The consistency check conducted by OWL reasoner shows that the ontology is consistent.

In the second example in Table 3, *Employee* class is generalized by *Manager* class. The transformation rule applied to this diagram results in the axiom: *SubClassOf(:Employee :Manager)*. The axiom, after being added to the domain ontology, changes the ontology but the consistency check conducted by OWL reasoner would also indicate that the ontology is consistent. The ontology is indeed still consistent because the reasoner only marks that *Employee* and *Manager* entities are equivalent. UML follows a Unique Name Assumption [13], unlike the OWL [22] and such a result would change the original meaning contained in the UML class diagram. This means that the reverse transformation (reengineering) from the modified domain ontology to the UML class diagram may result in obtaining a contradiction with UML semantics, what was shown in the second example.

A conclusion from the motivating example is that relying only on the transformation rules, may result in an incorrect UML diagram after reengineering from the modified domain ontology to UML class diagram. The information obtained from the reasoner that the modified domain ontology is still consistent is not enough to state that the original UML class diagram is compliant with the domain ontology. If the domain ontology is consistent the validation rules are required to check if the axioms from transformation rules after being added to the ontology have not changed the original UML semantics, and hence the final interpretation of the obtained result.

The observation that the transformation rules are not enough to validate UML class diagrams, and the verification rules are needed, is a major contribution of this article. This observation constitutes an important complement to the transformation rules described in the literature. The literature presents a transformation of selected elements of UML class diagrams to OWL representation. For this purpose verification rules are not needed, but they are very important in verification if the UML class diagram (and its OWL representation) is compliant with the OWL domain ontology.

3.3 Verification Rules

As it was shown in Sect. 3.2, the verification rules are crucial for the validation method. The verification rules are used to check if specific axioms exist in the domain ontology. The existence of any axiom indicated by the verification rules in the ontology means that the reengineering transformation (from the ontology to the diagram) would remain in conflict with the semantics of UML class diagram. The example of such a conflict is presented in Table 3 in Example 2, where a reengineered transformation resulted in incorrect cross generalization between the UML classes.

Definition 2 (*Verification rules*) The verification rule *vr* converts an element *e* of UML class diagram to a set *vr*(*e*) of OWL axioms which presence in the ontology means that the reengineering transformation (from the ontology to the diagram) would remain in conflict with the semantics of UML class diagram.

The verification rules are designed with the aim of supporting the transformation axioms and they assure that the diagram obtained as a result of reengineering from the modified domain ontology still preserves the semantics of the original UML class diagram (see Tables 4 and 5 for examples).

Table 4 UML class generalization with verification rules

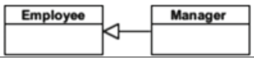
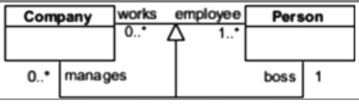
UML class diagram element	Class generalization
UML graphical symbol	
Verification rules	(1) SubClassOf(:Employee :Manager) (2) EquivalentClasses(:Manager :Employee)

Table 5 UML generalization between associations with verification rules

UML class diagram element	Generalization between associations
UML graphical symbol	 Company works employee Person 0..* 1..* 0..* manages boss 1
Verification rules	(1) SubObjectPropertyOf(:works :manages) (2) SubObjectPropertyOf(:employee :boss) (3) EquivalentObjectProperties(:manages :works) (4) EquivalentObjectProperties(:boss :employee)

The set CD_{Vr-OWL} defined by Eq. (2) is called a verificational part of OWL representation of UML class diagram. CD_{Vr-OWL} constitutes a union of sets of results of applying verification rules *vr* to all elements *e* of the UML class diagram *CD*. Each verification axiom must be associated with some transformation axioms.

$$CD_{Vr-OWL} = \bigcup_{e \in CD} vr(e) \tag{2}$$

3.4 Result of Validation

The consistency checks are used in the validation method in order to verify the UML class diagram against the domain ontology. Following [23], a consistency checker takes an ontology as input and returns a decision as either *Consistent*, *Inconsistent* or *Unknown*. In accordance with W3C recommendation [23], an *Unknown* result should not be returned by OWL 2 consistency checker. In practical realizations of OWL 2 reasoners the *Unknown* value is frequently omitted, for example, HermiT[1] and Pellet[2] reasoners return a *Boolean* value as a result of a method for checking consistency. Therefore, the *Unknown* value is also omitted in the proposed validation method and the results of the validation are stated on the basis of *Consistent* or *Inconsistent* result

[1]HermiT OWL Reasoner website: http://www.hermit-reasoner.com.

[2]Pellet website: https://github.com/Complexible/pellet.

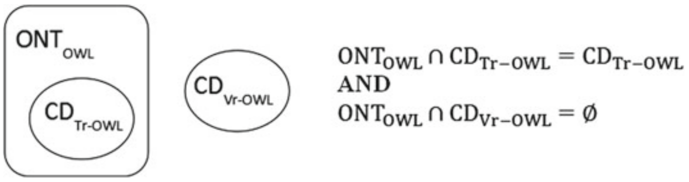


Fig. 2 A situation when the UML class diagram is compliant with the domain ontology

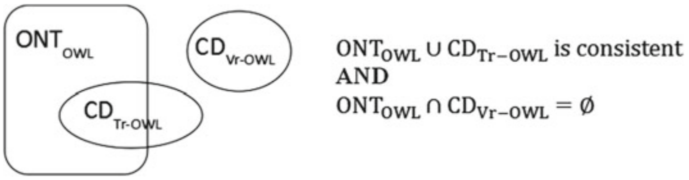


Fig. 3 A situation when the UML class diagram is not contradictory with the domain ontology

from the reasoner. Definitions 3–5 specify three possible results of the validation method.

Definition 3 (*Compliant diagram*) A UML class diagram is compliant with the domain ontology, only if all axioms from the transformational part of OWL representation of UML class diagram are contained in the axioms from the domain ontology AND the domain ontology does not contain any verification axioms.

Figure 2 presents a situation, when the UML class diagram is compliant with the domain ontology. This situation only appears when all axioms from CD_{Tr-OWL} are intersected with ONT_{OWL} and CD_{Vr-OWL} is not intersected with ONT_{OWL}.

Definition 4 (*Not contradictory diagram*) A UML class diagram which is not a compliant diagram, is also a not contradictory with the domain ontology, only if after adding all axioms from the transformational part of OWL representation of UML class diagram to the domain ontology, the domain ontology is consistent AND the domain ontology does not contain any verification axioms.

Figure 3 presents a situation, when the UML class diagram is not contradictory with the modified domain ontology. This situation only appears if CD_{Tr-OWL} intersected with ONT_{OWL} remains the modified domain ontology consistent and CD_{Vr-OWL} is not intersected with ONT_{OWL}. It can be noted that if the UML class diagram is not contradictory with the modified domain ontology, the modified domain ontology contains axioms that are a union of: $OWL_{ONT} \cup CD_{Tr-OWL}$.

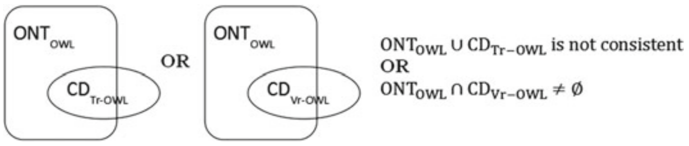


Fig. 4 Two situations when the UML class diagram is contradictory with the domain ontology

Definition 5 (*Contradictory diagram*) A UML class diagram which is not a compliant diagram, is a contradictory with the domain ontology, if at least one axiom from the transformational part of OWL representation of UML class diagram after being added to the domain ontology, causes the domain ontology to be inconsistent OR the domain ontology contains at least one verification axiom.

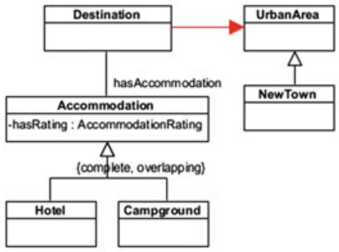
Figure 4 presents two situations, when the UML class diagram is contradictory with the domain ontology. This situation appears when either CD_{Tr-OWL} intersected with ONT_{OWL} remains the modified domain ontology inconsistent, or ONT_{OWL} is intersected with CD_{Vr-OWL}. UML class diagram is always contradictory with the domain ontology if the diagram and the ontology describe two different realities or the vocabulary between the ontology and the model was not initially agreed, what is a preliminary requirement to the method.

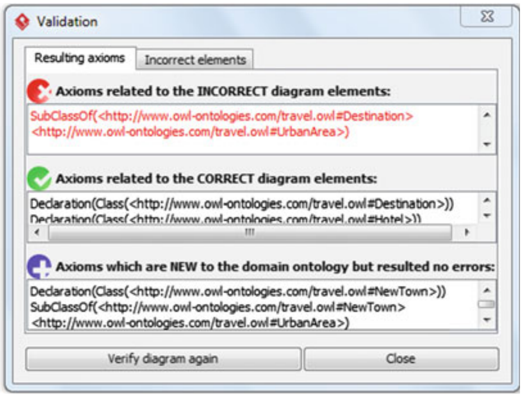
3.5 Prototype Tool

The proposed method for validation of UML class diagrams is implemented in a preliminary version of a tool (plug-in to Visual Paradigm[3]). The choice of the concrete UML class diagram (which needs to be validated) and the concrete domain ontology in OWL (which serves as a knowledge base) is made by a user of the tool. More specifically, the user decides which model and which domain ontology in OWL he or she would like to work with.

In the tool, the result of the validation is communicated visually to the user and the validation warnings are shown for diagram elements that have validation problems. Figure 5 presents an example result of an incorrect generalization detected by the tool in accordance with the selected domain ontology. The tool currently contains only selected transformation and verification rules which are related to classes with attributes, associations and generalizations. The current research is focused on further extension of the tool and the transformation and verification rules.

[3]Visual Paradigm for UML website: https://www.visual-paradigm.com.





Axioms related to the INCORRECT diagram elements:

```
SubClassOf(<http://www.owl-ontologies.com/travel.owl#Destination>
<http://www.owl-ontologies.com/travel.owl#UrbanArea>)
```

Axioms related to the CORRECT diagram elements:

```
Declaration(Class(<http://www.owl-ontologies.com/travel.owl#Destination>))
Declaration(Class(<http://www.owl-ontologies.com/travel.owl#Hotel>))
```

Axioms which are NEW to the domain ontology but resulted no errors:

```
Declaration(Class(<http://www.owl-ontologies.com/travel.owl#NewTown>))
SubClassOf(<http://www.owl-ontologies.com/travel.owl#NewTown>
<http://www.owl-ontologies.com/travel.owl#UrbanArea>)
```

Fig. 5 Incorrect generalization detected by the prototype tool

4 Conclusions

As presented in the article, domain ontologies and OWL reasoners can support the process of validation of the UML class diagrams. Currently, ontologies are more and more frequently used as a support for modelling in software development (e.g. [24, 25]), including business [26] and conceptual modelling [27]. There are a lot of databases and online libraries with OWL domain ontologies.[4]

Reusing of domain ontologies allows to draw upon specific domain knowledge without the necessity of constant involvement of domain experts.

A major contribution of this article is an observation that the transformation rules are not enough to validate UML class diagrams, and the additional rules (here called verification rules) are needed. It was discovered during implementation of the prototype tool.

The proposed method for validation of the semantic correctness of the models retrieves the domain knowledge captured in the domain ontologies expressed in OWL. The method, is based on changing the domain ontology by adding new axioms to it—one by one from the transformational part of OWL representation of UML class diagram—and on subsequent verification if the modified domain ontology is still consistent. If the modified domain ontology is consistent, all associated verification axioms are one by one checked for the existence in the ontology. The existence of any axiom indicated by the verification rules in the modified domain ontology means that a reengineering transformation (from the ontology to the diagram) would remain in conflict with the semantics of UML class diagram.

[4]Some examples of online OWL databases and libraries are as follows: The OBO Foundry database: http://www.obofoundry.org/, Information Systems Group Ontologies: http://www.cs.ox.ac.uk/isg/ontologies/, Protege Ontology Library: http://protegewiki.stanford.edu/wiki/Protege_Ontology_Library.

A full overview with a revision and extension of the state of the art transformation rules for UML class diagrams is a subject of the related article, which is currently in preparation. The current research is also focused on proposing verification rules and on extending the prototype tool in order to validate full UML class diagrams. The tool is aimed to automatically check the compliance of the UML class diagram with the field described by the domain ontology expressed in OWL. Thanks to this solution, the modellers will obtain a method for automatic validation of UML class diagrams against the domain they describe. Future work is also focused on extending the UML class diagrams by the knowledge from the OWL domain ontologies. The domain ontology may serve as a source of information that is not yet presented in the UML model but is worth considering to be included.

References

1. OMG, Unified Modeling Language,Version 2.5, Doc. No.: ptc/2013-09-05 (2015). http://www.omg.org/spec/UML/2.5
2. Niittgens, M., Fold, T., Zimmermann, V.: Business process modeling with EPC and UML: transformation or integration. In: Schader, M., Korthaus, A. [36], pp. 250–261 (1998)
3. Lindland, O.I., Sindre, G., Solvberg, A.: Understanding quality in conceptual modeling. Softw. IEEE **11**(2), 42–49 (1994)
4. OWL 2 Web Ontology Language. Direct Semantics (Second Edition). W3c Recommendation 11 December 2012 (2012). http://www.w3.org/tr/owl2-syntax/
5. Atkinson, C., Kiko, K.: A detailed comparison of UML and OWL. Technischer Bericht 4, Department of Mathematics and Computer Science. University of Mannheim (2008)
6. Parreiras, F.S., Staab, S., Winter, A.: On marrying ontological and metamodeling technical spaces. In: Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, pp. 439–448. ACM (2007)
7. OWL 2 Web Ontology Language Document Overview (Second Edition). W3c Recommendation 11 December 2012 (2012). https://www.w3.org/tr/owl2-overview/
8. Huzar, Z., Sadowska, M.: Towards creating complete business process models. In Chapter 5 In: From Requirements to Software: Research and Practice, pp. 77–86 (2015)
9. Bahaj, M., Bakkas, J.: Automatic conversion method of class diagrams to ontologies maintaining their semantic features. Int. J. Soft Comput. Eng. (IJSCE) **2** (2013)
10. Belghiat, A., Bourahla, M.: Transformation of uml models towards owl ontologies. In: 2012 6th International Conference on Sciences of Electronics, Technologies of Information and Telecommunications (SETIT), pp. 840–846. IEEE (2012)
11. Jesper, Z., Luttenberger, N.: Conceptual modelling in UML and OWL-2. Int. J. Adv. Soft. **7**(1 & 2) (2014)
12. Khan, A.H., Porres, I.: Consistency of uml class, object and statechart diagrams using ontology reasoners. J. Vis. Lang. Comput. **26**, 42–65 (2015)
13. Zedlitz, J., Jörke, J., Luttenberger, N.: From UML to OWL 2. In: Knowledge Technology, pp. 154–163. Springer, Berlin (2012)
14. Zedlitz, J., Luttenberger, N.: Transforming between uml conceptual models and owl 2 ontologies. In: Terra Cognita 2012 Workshop, vol. 6, p. 15 (2012)
15. Zedlitz, J., Luttenberger, N.: Data types in UML and OWL-2. In: Presented at the Semapro 2013: The Seventh International Conference on Advances in Semantic Processing (2013)
16. Felfernig, A., Friedrich, G.E., Jannach, D.: UML as domain specific language for the construction of knowledge-based configuration systems. Int. J. Soft. Eng. Knowl. Eng. **10**(04), 449–469 (2000)

17. Unhelkar, B.: Verification and validation for quality of UML 2.0 models, vol. 42. Wiley (2005)
18. Bertolino, A., De Angelis, G., Di Sandro, A., Sabetta, A.: Is my model right? Let me ask the expert. J. Syst. Softw. **84**(7), 1089–1099 (2011)
19. Letelier, P., Snchez, P.: Validation of UML classes through animation. In: Advanced Conceptual Modeling Techniques, pp. 300–311. Springer, Berlin (2002)
20. OWL 2 Web Ontology Language: Profiles. W3c Working Draft 11 April 2008 (2008). https://www.w3.org/tr/2008/wd-owl2-profiles-20080411/
21. Gherabi, N., Bahaj, M.: A new method for mapping UML class into OWL ontology. Spec. Issue Int. J. Comput. Appl. (0975 8887) Soft. Eng. Databases Expert Syst. SEDEXS 5–9 (2012)
22. OWL 2 Web Ontology Language Profiles (Second Edition). W3c Recommendation 11 December 2012 (2008). https://www.w3.org/tr/owl2-profiles/
23. OWL Web Ontology Language. Test Cases. W3c Recommendation 10 February 2004 (2004). https://www.w3.org/tr/owl-test/#consistencychecker
24. Ga, D., Djuric, D., Deved, V.: Model Driven Architecture and Ontology Development. Springer Science & Business Media (2006)
25. Parreiras, F.S., Staab, S., Ebert, J., Pan, J.Z., Miksa, K., Kühn, H., Zivkovic, S., Tinella, S., Assmann, U., Henriksson, J.: Semantics of software modeling. Semant. Comput. pp. 229–247 (2010)
26. Gailly, F., Poels, G.: Ontology-driven business modelling: improving the conceptual representation of the REA ontology. In: Conceptual Modeling-ER 2007, pp. 407–422. Springer, Berlin (2007)
27. Hnatkowska, B., Huzar, Z., Dubielewicz, I., Tuzinkiewicz, L.: Problems of SUMO-like ontology usage in domain modelling. In: Intelligent Information and Database Systems, pp. 352–363. Springer international publishing edn. (2014)

RSL-DL: Representing Domain Knowledge for the Purpose of Code Generation

Kamil Rybiński and Rafał Parol

Abstract In the paper a new extension to the existing RSL (Requirements Specification Language) language has been proposed, called RSL-DL (Requirements Specification Language-Domain Logic). Based on the declarative paradigm of programming its aim is to enable defining ontologies at the requirements level and as a consequence also describing the domain logic part of requirements specification. Being fully complement with RSL it supplements ReDSeeDS (Requirements-Driven Software Development System) technology with a possibility to generate by far missing code of the Model layer of standard MVP architectural pattern. The main idea of the solution has been described, together with the definition of the key language elements, in terms of syntax, notation and semantics as well. Additionally the role of the RSL-DL within the whole ReDSeeDS technology framework has been precisely defined and comparison with other languages has also been included.

1 Introduction

Functional requirements formulation, even nowadays in times of growing popularity of agile software development practices, is one of the most important tasks during the software development process, having a huge impact on the final quality of the application, as well as on the fact if the project ends with success or not. The possibility of transforming the requirements specification into fully or nearly working code could provide lots of benefits, including shortening the length of the project development cycle and reducing the effort of software architects, designers and programmers. This is exactly what the ReDSeeDS (Requirements-Driven Software Development System) technology [2] aims to do. With the use of dedicated language with well-defined syntax named RSL (Requirements Specification Language) [10] for specifying requirements, some set of transformation rules depending on the target technology (programming language, architectural framework) and transformation program

K. Rybiński (✉) · R. Parol
Warsaw University of Technology, Warsaw, Poland
e-mail: rybinskk@iem.pw.edu.pl

L. Madeyski et al. (eds.), *Software Engineering: Challenges and Solutions*,
Advances in Intelligent Systems and Computing 504,
DOI 10.1007/978-3-319-43606-7_5

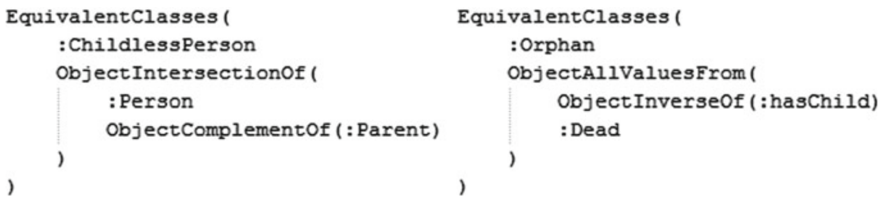
implementation written in MOLA language [11] it generates the majority of the code that is needed to run the properly working application. Most transformations try to apply the MVP architectural pattern and by far only the code of View and Presenter layers is being generated [17]. To fill with necessary code the classes and methods of the Model layer we need to find a way to describe the domain logic of the analyzed problem and possibly propose some extension to RSL language. Because such an extension should be understandable even for non-programmers (just like RSL language is) some declarative approaches, aiming at building ontologies and making the use of reasoning mechanisms should probably be considered. Therefore, in this paper we propose the RSL-DL (Requirements Specification Language-Domain Logic) extension of the RSL, a language following the declarative paradigm of programming using easy to understand graphical notation.

2 Overview of Knowledge Representation Languages

Before we propose a new language, we should check first if maybe some existing ones satisfy our needs. There are many options to choose from if we need a language appropriate for the task of modelling the knowledge representation or building ontologies. For the purpose of our analysis, we can distinguish three categories of such solutions. First one contains languages supporting semantic web idea such as RDF [12] or OWL [9, 14]. Second one is related to languages focused on knowledge representation and automatic reasoning such as OCML [15] or LOOM [3, 13]. Third one includes logic programming languages such as Prolog [4, 5]. Below we present a description of selected knowledge representation languages, specifically representative ones of each of mentioned categories.

2.1 OWL

The OWL (Web Ontology Language) language is discussed here as example of languages belonging to the first of previously listed categories. Its main idea is to make it possible to build a comprehensive taxonomy of classes, define relations between them and then try to classify different well-described individuals appearing in queries as instances of particular classes or not. Classes are described in relation to other classes (being a subclass of, belonging to set of disjoint classes), often in a very complex way with the use of logical class constructors (union of classes, intersection of classes, complement to a class), and by relations in which each class member participates. Such relations are called properties and can be used to connect objects with objects or objects with data values. Properties can also be defined in a very precise way: they can be for example classified as functional ones (the object can be linked by such a property with at most one another), symmetric ones or as inverses of other properties. The names appearing in OWL take the form of IRI (International



```
EquivalentClasses(
    :ChildlessPerson
    ObjectIntersectionOf(
        :Person
        ObjectComplementOf(:Parent)
    )
)
```

```
EquivalentClasses(
    :Orphan
    ObjectAllValuesFrom(
        ObjectInverseOf(:hasChild)
        :Dead
    )
)
```

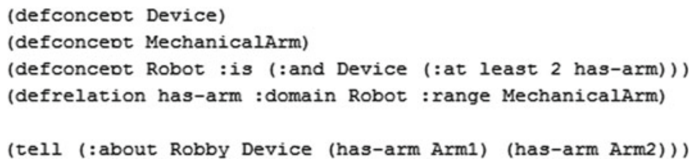
Fig. 1 Example of OWL code (based on example from [9])

Resource Identifiers) format [8]. Some sorts of abbreviations can also be used. We can see some example of relatively complex definitions of classes in Fig. 1.

We can find out from the example that a childless person is a one who is at the same time a person and not a parent. In turn, orphan is an individual who is characterized by the fact that he participates in relationships of being a child as an object (not as a subject) only with individuals who are already dead.

2.2 LOOM

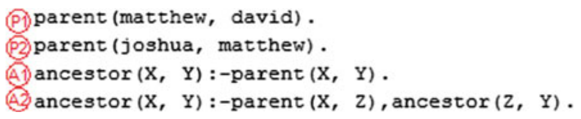
LOOM language can be considered as a good example from the second category. It can also be characterized as a mix of rule-based and frame-based solutions. Generally, in the case of rule-based approaches the reasoning mechanism infers on the basis of some conditions that have to be satisfied to draw particular conclusions. Frame-based solutions rely on providing some kind of detailed descriptions of the objects in terms of values of their attributes and relationships between them, which results in ontologies modelled in a very precise way. Similarly as in the case of OWL, each concept (that is how LOOM calls the class of objects) can be defined directly in terms of other more general concepts, as well as by relations in which it participates with other classes of objects. In a typical situation after implementing the program (using TBox part of the language—the one for describing concepts and relations), the user tries to describe some individual object and then asks a computer (all of this can be done with the ABox part of the language—the one for asserting facts and executing queries) if the individual (taking into account its individual definition) can be classified as a member of particular class. We can observe it in Fig. 2. The code



```
(defconcept Device)
(defconcept MechanicalArm)
(defconcept Robot :is (:and Device (:at least 2 has-arm)))
(defrelation has-arm :domain Robot :range MechanicalArm)

(tell (:about Robby Device (has-arm Arm1) (has-arm Arm2)))
```

Fig. 2 Example of LOOM code (based on example from [3])



```
P1 parent(matthew, david).
P2 parent(joshua, matthew).
A1 ancestor(X, Y):-parent(X, Y).
A2 ancestor(X, Y):-parent(X, Z),ancestor(Z, Y).
```

Fig. 3 Example of Prolog code (based on example from [4])

has so intuitive syntax that almost no additional explanations are needed. Of course the statement that *Robby* is a robot is going to be evaluated to be true.

2.3 Prolog

Prolog can be treated as an example language belonging to the third of the mentioned categories. As every other language presented here Prolog is a declarative one—when writing a code we tell the computer what we want to achieve, but not how it should be done. An example of Prolog code can be seen in Fig. 3.

The typical program written in Prolog consists of two types of clauses: facts and rules. The facts describe particular relations between different concrete objects. Placing the facts within the lines of the program aims to provide the computer with some explicitly given static knowledge about the nature of analyzed problem. For example the fact (P1) informs the computer that objects named *matthew* and *david* are in some relation with each other called *parent*. In turn, the rules represent the knowledge that makes it possible to the computer to discover new observations that are not explicitly given. For example the rule (A1) means that if variables *X* and *Y*, which can represent any objects, are in relation called *parent* it is a sufficient condition to make the statement that also the relation *ancestor(X, Y)* is true. Generally speaking each rule says that relation on the left takes place and is evaluated to be true only if all the conditions on the right are proven to be true. Rules can also be expressed in a recursive manner as we can see in the case of rule (A2).

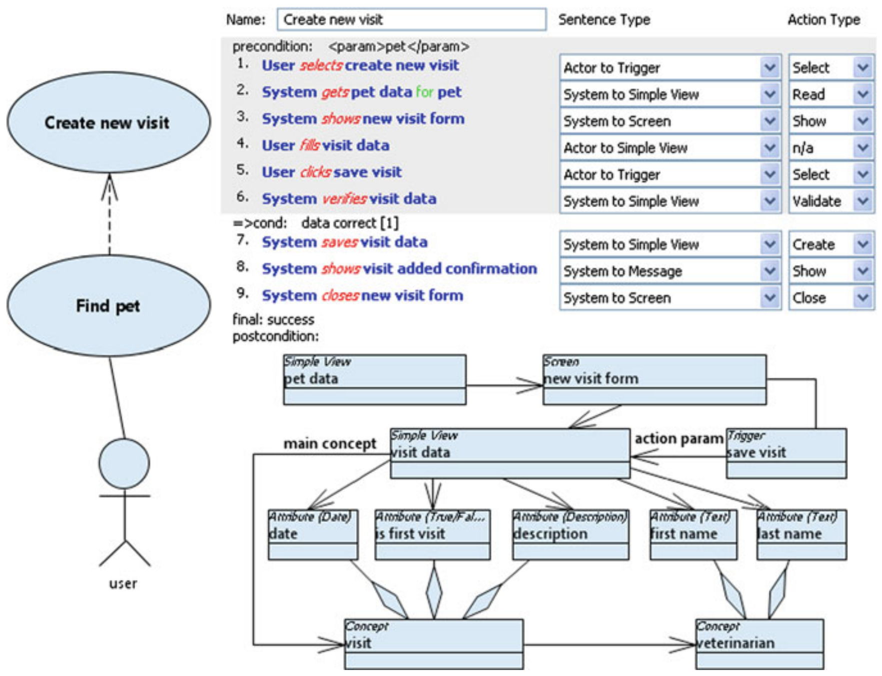
Having such a piece of code as in Fig. 3 we can try to ask the computer some questions, for example "*?- ancestor(joshua, david).*" (Is Joshua ancestor of David?). To prove the statement contained in the question, we have to make use of some corresponding fact or appropriate rule from the program. It means that we need to find explicit fact saying that *joshua* is ancestor of *david* or the general rule that can be proven to be true if respective variables *X* and *Y* are bound to those values (*joshua* and *david*). Making the use of rules involves performing reasoning tasks. Prolog is able to infer as well forwards (forward chaining—proving new statements on the basis of the already known ones until we prove the one we are interested in) as backwards (backward chaining—checking if we are able to prove all the statements necessary to prove the one we are interested in). Eventually, our question is of course evaluated to be true.

2.4 Disadvantages of Described Languages

Every single language has its own pros and cons. The detailed comparison of different languages (including the ones described above) can be found in [6, 7, 16]. Here we only focus on the properties important from the point of view of the analyzed issue. First it should be noticed that since one of the huge advantages of RSL language is making the use of the syntax that is easily understandable even for non-programmers (scenario sentences expressed in a language similar to the natural one, use case and domain notions models drawn as diagrams with the use of intuitive graphical notations), the syntax used to define the domain logic should follow this approach. Furthermore, the domain logic for specific problems sometimes turns out to be really complicated, so it seems to be a good idea to illustrate all the complexity with the use of diagrams. Unfortunately all previously described languages use some kind of textual notation. Moreover even if someone could say, that the syntax of LOOM and OWL can be treated as easily understandable, that is true only for programmers, especially the ones familiar with the object-oriented paradigm. We should also remember that our final goal is to generate some (fully or nearly) working code from our domain logic description. In most cases programs in other languages are rather interpreted within dedicated integrated environments and not transformed to any other form of code (executable or not). Also in most cases the other languages are dedicated to operate on knowledge bases defined inside the code, whereas typical business applications are strongly dependent on external data sources, such as databases or files. Maybe only languages from the first category are some kind of exception here, since for example every individual in OWL is identified by its name being an IRI, which means that it can be any object in the whole Internet. However even in that case there is no definition of internal data structure of such an object which is required to use the object and its data inside the domain logic code. Finally, sometimes there is a need to make use of constructs typical for procedural style of programming, like loops or functions. Even if some languages, like LOOM, contain them, they are most often not really intuitive to use.

3 ReDSeeDS Technology

To understand why we need the RSL–DL extension of the RSL language we must consider it within the context of ReDSeeDS technology, which aim is to generate (fully or nearly) working code from the level of requirements specification. Any transformation from requirements to code is not possible if requirements are written in unconstrained natural language. Therefore, we need some kind of special purpose constrained language with well–defined syntax. RSL language has been defined to satisfy these demands. The task of specifying the requirements starts with identification of use cases. We can see an example of decomposition of the small part of the system functionality to a set of single use cases in the Fig. 4. Every use case, like



Name: Create new visit

precondition: <param>pet</param>

	Sentence	Sentence Type	Action Type
1.	User selects create new visit	Actor to Trigger	Select
2.	System gets pet data for pet	System to Simple View	Read
3.	System shows new visit form	System to Screen	Show
4.	User fills visit data	Actor to Simple View	n/a
5.	User clicks save visit	Actor to Trigger	Select
6.	System verifies visit data	System to Simple View	Validate
	=>cond: data correct [1]		
7.	System saves visit data	System to Simple View	Create
8.	System shows visit added confirmation	System to Message	Show
9.	System closes new visit form	System to Screen	Close

final: success
postcondition:

Fig. 4 Example of RSL specification

"Find pet" or "Create new visit" in the considered example of pet clinic supporting system, has to be then described by a set of scenarios showing how the goal of a use case can be achieved (or not) from the point of view of communication of the user with the system. An example of such scenario for the "Create new visit" use case is shown in the same figure. Scenarios can end with success or with a failure depending on the data entered by the user, his decisions and some other specific conditions. Each scenario is a set of simple sentences written in SVO notation (Subject–Verb–Object). Possibly some special invocation sentences, join sentences and condition sentences can be included as well. The Subject of a SVO sentence (like "System" in the sentence no. 8) is always either the actor of a use case or the system itself. The Verb (like "saves" in the same sentence) may be chosen as a one from the set of some predefined verb types ("saves" is of a "Create" type), but it can be also any other verb that describes performed action well. Any word (or set of words) can also appear in the sentence as the Object (like "visit data"). However it needs to be classified to the one of some predefined groups ("visit data" is of a "Simple View" type). On the basis of the choice of Subject, Verb and Object different kinds of sentences can be distinguished (like "System to Simple View"). Every single kind causes different parts of the code within the range of particular architectural layers to be generated. Although the sentences written in RSL must follow some well–defined syntax rules, they are nearly as easy to understand as natural language sentences. Finally, the use

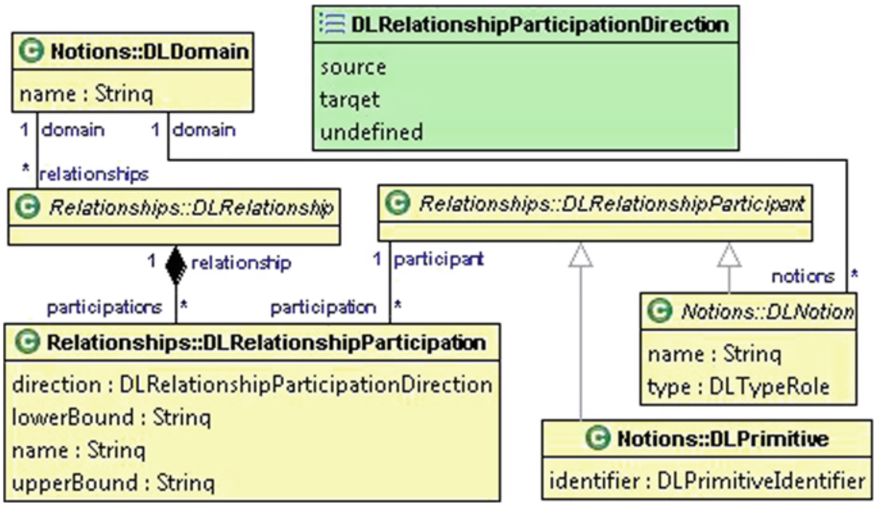


Fig. 5 Core fragment of RSL-DL meta-model

case model together with scenario sets has to be complemented with the domain notions model describing all the notions playing the role of Objects in the sentences and relationships between them. The example of such a domain model, precisely a part of it related to the "Create new visit" use case, has also been shown in Fig. 4.

All the necessary parts of requirements specification (use case model, scenarios sets, domain notions model) can be prepared with the use of dedicated tool (editor) being a part of ReDSeedS technology. When our specification is ready, the ReDSeeDS tool is able to invoke the transformation engine. Most of existing transformation programmes try to apply the MVP architectural pattern in the final code. It assumes that the set of all classes should be logically divided into three layers: Model layer, View layer and Presenter layer. By far the best existing transformation programmes are able to generate all the classes, methods and code needed to implement the tasks of the View and the Presenter layers. In the case of the Model layer all the classes with necessary methods are being generated, but by far no code in the bodies of these methods is being provided. For example the sentence no. 8 in the main scenario of "Create new visit" use case says that system should be able to "save visit data" if needed. According to that sentence and standard transformation rules for MVP architectural pattern [17] a class named MVisitData belonging to the Model layer should be generated, responsible for performing all the tasks regarding to the visit data abstract concept. Also the method "save" needs to be generated inside the MVisitData class. However, by far the requirements specification coded in RSL does not provide us with any necessary information enabling to do much more. Because the user delivers some information on how he understands the "visit data" notion with the use of domain notions model we should be able to generate the code related to all the database access operations, which may be even combined with generating the database schema. However we also need to express in RSL what the user means, for example by saying that the "visit data" needs to be "verified" (sentence no. 7). As we see, without any doubts we need some extension of RSL language being able to express information related to the domain logic.

4 Solution Overview

As the RSL-DL is intended to supplement ReDSeeDS code generation capabilities, its general structure is based on corresponding part of RSL language used by that system. However, in order to adapt better to the task of specifying domain knowledge, introduction of a few adjustments is necessary. General division into notions ("DLNotion"), reflecting occurring concepts, and relationships ("DLRelationship"), reflecting associations between them, has been preserved, although the way of representing the latter ones needs proper modification. As it can be seen in the meta-model in Fig. 5, separate meta-class for representing participation in relationship ("DLRelationshipParticipation") has been extracted, which enables more convenient way of modelling multipart relationships and more precise description of each element's role in them. Some adjustments in language concrete syntax have also been introduced by representing relationships as hexagon figures. Additionally each notion and relationship can be assigned to proper domain ("DLDomain") regarding to the area of interest that is being described. Similarly new meta-class("DLPrimitive") has been introduced for representing global concepts not related to any particular domain, like for example current date.

Moreover a few further clarifications have been made. As can be seen in Fig. 6 relationships have been further divided into groups to define their roles and meaning more precisely. First level division distinguishes references ("DLReference"), which reflect how notions are related to each other, from transitions ("DLTransition"), that define how to traverse between them or transform one into another. The first

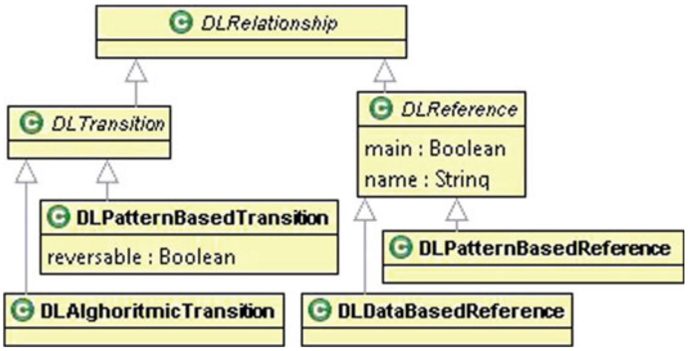


Fig. 6 Relationships in RSL-DL meta-model

group is further divided into pattern-based references (“DLPatternBasedReference”) and data-based references (“DLDataBasedReference”). Pattern-based references are used to model situations when relations between notions can be inferred from the set of rules. For example “siblings” can be defined as two separate individuals who have the same mother and father. Data-based references are to be applied when information about associations between notions is contained in some external data source, like relational database. The relationship between books and their authors could serve as an example of that type of reference. The second group of relations is also further divided into pattern-based transitions (“DLPatternBasedTransition”) and algorithmic transitions (“DLAlghoritmicTransition”). Pattern-based transitions are the ones that can be described by a set of static rules or formulas. For example, basing on the knowledge about the distance between two points and the time needed to move from one to another we can use pattern-based transition to determine the object’s velocity. Additionally such transitions can be also marked as reversible or irreversible ones. Algorithmic transitions are the ones which cannot be easily described in a previous way and require executing a given number of steps in a particular order to determine the counterpart in the relationship. A transition between unsorted list and sorted one with the use of bubble sort algorithm could be a good example.

As it can be seen in Fig. 7 notions are also divided into subclasses: entities (“DLEntity”) and properties (“DLProperty”). Properties are simple concepts associated with some specified values assigned to them, whereas entities represent independent, presumably more complex elements of the represented domain. “Person”, “student” or “physical body” can be seen as examples of entities while “name” and “speed” are examples of properties. In order to save space the enumeration presenting possible value types for properties is omitted on the diagram, as it is similar to the one from classical RSL. Each notion can be also described more precisely by assigning to it some additional conditions. There are three types of them: inheritance condi-

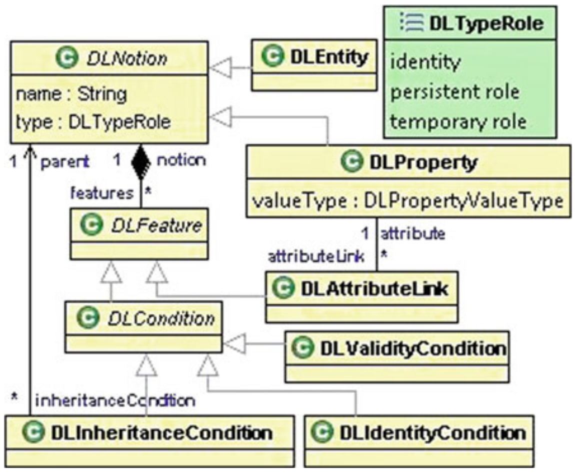


Fig. 7 Notions in RSL-DL meta-model

tions (“DLInheritanceCondition”), identity conditions (“DLIdentityCondition”) and validity conditions (“DLValidityCondition”). Inheritance conditions reflect specialisation dependencies between notions: for example every “student” is also a “person”. Identity conditions specify all other types of conditions which define notions identities, for example the object can be classified as a book if it has ISBN number having exactly thirteen digits. Validity conditions provide additional information which can be used to check notions validity, for example valid ISBN numbers should have proper check digits. Additionally properties can be assigned to entities as their attributes by attribute links (“DLAttributeLink”)—for example “name” property can be added as an attribute to the “person” entity. Objects can also be marked as belonging to the classes of different entities in many ways, depending on persistence of such an assignment and object’s life cycle. If a given notion is of an “identity” type, it means that particular object has to be a member of that class (notion) from the very beginning (creation of the object) to the end. In the case of two other types, the class (notion) membership can be assigned during the object’s life cycle. It means that object may become the member of the class (notion) temporarily (“temporary role” type of a notion) or persistently (“persistent role” type of a notion). “Person” could be treated as an example of “identity” type, “student” as an example of “persistent role” type that can be assigned to the object during its life cycle and “physical body” as an example of “temporary role” type used only for purpose of some calculations. It is worth noting that the permanence of object’s role (like in the case of being a “student”) does not imply its immutability, but only the permanence of its assignment, until it is directly changed.

Described language allows for storing domain rules, but proper selection of the ones needed to generate the code for given methods in the Model layer is equally important task. To achieve that goal inference engine is going to be used. With the use of forward or backward chaining proper tree structure representing the graph of needed rules has to be build based on particular references and transitions, as well as definitions of their possible inputs and outputs. In order to become independent from the form of given rules, symbolic computation can also be used to convert them in

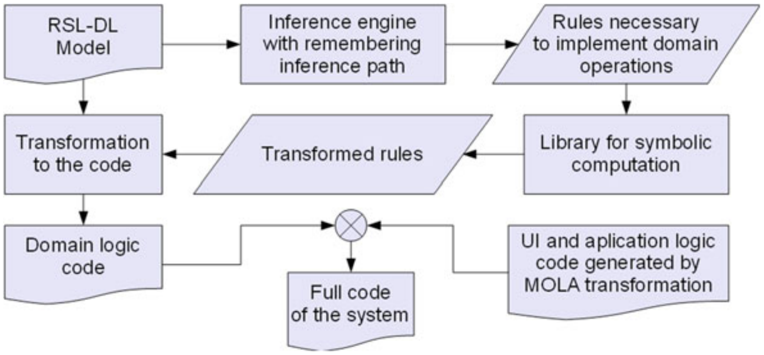

RSL-DL Model
Inference engine with remembering inference path
Rules necessary to implement domain operations
Transformation to the code
Transformed rules
Library for symbolic computation
Domain logic code
Full code of the system
UI and aplication logic code generated by MOLA transformation

Fig. 8 Transformation scheme

any convenient way. Then such a tree structure of rules should be transformed into code of proper methods. Additionally, on the basis of the taxonomy of notions and conditions the relevant structure of classes has to be also created. Together with current ReDSeeDS tool capabilities it should provide a possibility of a full code generation. Overview of the whole transformation process is shown in Fig. 8.

5 RSL-DL Model Example

A more comprehensive example is shown in Fig. 9. It presents domain model comprised of elements related to registration and evaluation processes in typical university. It consists of three major entities: "Student", "Subject" and "Partial grade", where "Student" is an example of "persistent role" which can be added to any "Person" and "Partial grade" is a specific kind of a grade, which not only has a value, but also a weight. Both value and weight are represented by valid properties—weight by a simple property of an integer type and value by a property of a float type constrained by identity condition determining available grade values. Based on these elements appropriate transformation can generate domain logic classes, where attributes will be based on properties and their value types and inheritance conditions together with role types will indicate proper definitions of class constructors. The example also contains three important relationships. First two of them, "Registration" and "Grade", represent associations between "Students" and "Subjects" and "Students", "Subjects" and "Partial grades" respectively, defined by means of external data sources—as a result they both belong to the data-based reference relationship subtype. Existence of these relationships allows for generation of proper database schemas and data access objects. Also proper methods for participating classes enabling them operations on such data set will be generated. The third relationship "Weighted average grade" is a pattern-based transition illustrating the way of computing the value of "Average grade" for given "Subject" and given "Student". Direction of arrows representing participations in that relationship indicates possible direction of computations and additional names accompanying the arrows allow for more concise specification of formulas. In our example only one direction of such computation is possible (given transition is irreversible one) and the expression is already given in a proper format (there is no need of using symbolic computation to transform any formulas in this case), so only the method to compute average grade for given student and given subject will be generated. Realization of all mentioned classes and methods can be observed in Fig. 10. At the same time, it also reveals the translational semantics of our solution, showing how RSL-DL concepts can be expressed in other languages (like UML or Java) with precise, well-known and well-defined meaning of particular constructs comprising them.

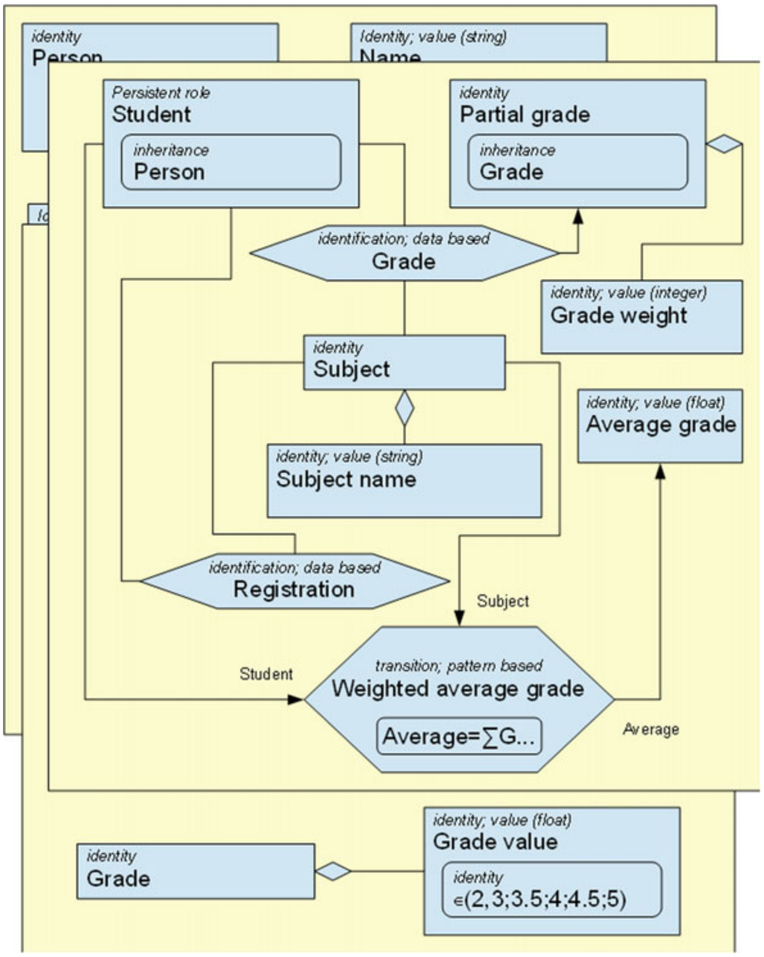
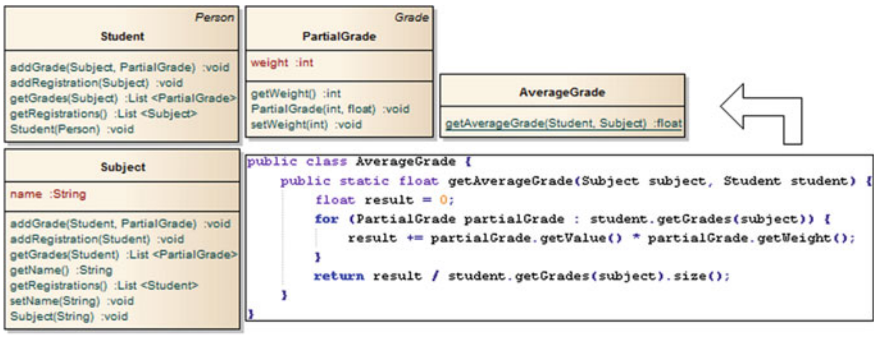


Fig. 9 RSL-DL example



Student
addGrade(Subject, PartialGrade) :void
addRegistration(Subject) :void
getGrades(Subject) :List <PartialGrade>
getRegistrations() :List <Subject>
Student(Person) :void

PartialGrade
weight :int
getWeight() :int
PartialGrade(int, float) :void
setWeight(int) :void

AverageGrade
getAverageGrade(Student, Subject) :float

Subject
name :String
addGrade(Student, PartialGrade) :void
addRegistration(Student) :void
getGrades(Student) :List <PartialGrade>
getName() :String
getRegistrations() :List <Student>
setName(String) :void
Subject(String) :void

```
public class AverageGrade {
    public static float getAverageGrade(Subject subject, Student student) {
        float result = 0;
        for (PartialGrade partialGrade : student.getGrades(subject)) {
            result += partialGrade.getValue() * partialGrade.getWeight();
        }
        return result / student.getGrades(subject).size();
    }
}
```

Fig. 10 Code example

6 Conclusions and Further Work

Attempts that have been already made seem to indicate that proposed classification of notions and relationships will make it possible to define proper models with all the necessary data processing rules for different domains. The possibility to make use of procedural style constructs (when needed) in an intuitive way, together with the ability of the language to refer to external data sources allows for greater flexibility in defining the domain logic. As we see in examples the use of graphical notation makes the domain logic description also easily understandable. For that purpose (drawing the RSL-DL diagrams) dedicated graphical editor has already been implemented, with the use of GMF Framework [1]. Our further work will focus on making the transformation rules definition and the transformation engine implementation, which are both currently in the advanced development stage, complete. In the future dedicated query language for the RSL-DL has to be formulated, together with procedures of mapping between RSL scenario sentences and such queries and their results.

References

1. Graphical Modeling Project. http://www.eclipse.org/modeling/gmp/
2. ReDSeeDS project home page. http://redseeds.eu/
3. Loom User's Guide. Technical report, ISX Corporation (1991)
4. Bramer, M.: Logic Programming with Prolog. Springer, London (2013)
5. Colmeraner, A., Kanoui, H., Pasero, R., Roussel, P.: Un systeme de communication homme-machine en francais. Luminy (1973)
6. Corcho, O., Fernández-López, M., Gómez-Pérez, A.: Methodologies, tools and languages for building ontologies. Where is their meeting point? Data Knowl. Eng. **46**(1), 41–64 (2003)
7. Corcho, O., Gómez-Pérez, A.: Evaluating knowledge representation and reasoning capabilites of ontology specification languages. In: Proceedings of the ECAI'00 Workshop on Applications of Ontologies and Problem Solving Methods (2000)
8. Dürst, M., Suignard, M.: Internationalized Resource Identifiers (IRIs). Technical report (2004)
9. Hitzler, P., Krötzsch, M., Parsia, B., Patel-Schneider, P.F., Rudolph, S.: OWL 2 Web Ontology Language Primer. Technical report (2012)
10. Kaindl, H., Śmiałek, M., Wagner, P., et al.: Requirements Specification Language Definition. Technical Report D2.4.2, ReDSeeDS Project (2009)
11. Kalnins, A., Barzdins, J., Celms, E.: Model Transformation Language MOLA. Lect. Notes Comput. Sci. **3599**, 14–28 (2005)
12. Lassila, O., Swick, R.R.: Resource Description Framework (RDF) Model and Syntax Specification (1999)
13. MacGregor, R., Bates, R.: The Loom Knowledge Representation Language. Technical report, DTIC Document (1987)
14. McGuinness, D.L., Van Harmelen, F.: OWL web ontology language overview. World Wide Web Consortium (W3C) Recommendation (2004)
15. Motta, E.: An overview of the OCML modelling language. In: the 8th Workshop on Methods and Languages (1998)
16. Slimani, T.: Ontology development: a comparing study on tools, languages and formalisms. Indian J. Sci. Technol. **8**(24) (2015)
17. Śmiałek, M., Nowakowski, W.: From Requirements to Java in a Snap. Springer International Publishing (2015)

Part II
Quality Assurance

Assessment of the Software Defect Prediction Cost Effectiveness in an Industrial Project

Jaroslaw Hryszko and Lech Madeyski

Abstract Software defect prediction is a promising, new approach to increase both, software quality and development pace. Unfortunately, the cost effectiveness of software defect prediction in industrial settings is not eagerly shared by the pioneering companies. In particular, the cost effectiveness of using the DePress open source software measurement framework, developed by Wroclaw University of Science and Technology, and Capgemini software development company, for defect prediction in commercial software development projects have not been previously investigated. Thus, in this paper, we explore whether defect prediction can positively impact an industrial software development project by generating profits. To meet this goal, we conducted a defect prediction and simulated potential quality assurance costs based on the best prediction result, as well as the proposed Quality Assurance (QA) strategy. Results of our investigation were optimistic: we estimated that quality assurance costs can be reduced by almost 30 % when proposed approach will be used, while estimated DePress tool usage Return on Investment (ROI) is fully 73 (7300 %), and Benefits Cost Ratio (BCR) is 74. Such promising results have caused the acceptance of continued usage of the DePress-based software defect prediction for actual industrial projects run by Volvo Group.

J. Hryszko (✉) · L. Madeyski (✉)
Faculty of Computer Science and Management, Wroclaw University of Science and Technology, Wrocław, Poland
e-mail: jaroslaw.hryszko@pwr.edu.pl

L. Madeyski
e-mail: lech.madeyski@pwr.edu.pl

J. Hryszko
Volvo Group, Gothenburg, Sweden

L. Madeyski et al. (eds.), *Software Engineering: Challenges and Solutions*,
Advances in Intelligent Systems and Computing 504,
DOI 10.1007/978-3-319-43606-7_6

1 Introduction

Until recently, software defect prediction process has been considered to be too complex, expensive and time-consuming, as well as there have been lack of solutions for wrapping required tools into one, universal, defect prediction framework which could be used for different software projects.

To fill this gap, Madeyski and Majchrzak [1] proposed a new, extensible (plugin-based) framework called DePress. DePress (*Defect Prediction for Software Systems*) builds upon the KNIME framework [2] and allows development of graphical workflows and uses an intuitive, user-friendly interface. Being intuitive and highly customizable, the DePress makes itself a perfect tool which can be conveniently utilized (thanks to its user-friendly interface and a wide range of plugins) in different commercial software development projects for software defect prediction. Detailed description of DePress framework and its capabilities can be found on DePress website [3] or in the article by Madeyski and Majchrzak [1].

Potential benefits of using the DePress framework for defect prediction in commercial software development projects have not been investigated [1]. To fill this gap, our study aimed to answer the following research questions:

RQ1: *What is the highest level of defect prediction, measured by F-measure, achievable by the DePress tool (using a default, non-tweaked configuration) in an industrial software project?*

The possible benefit varies, depending on the potential prediction effectiveness. This implies the need of first verifying what is the highest F-measure (harmonic mean of precision and recall [4]) value of the defect prediction process handled entirely by the DePress tool. DePress can be highly customizable thanks to its plugin-based architecture, as well as its open source nature. However, such adjustments can generate additional costs. Therefore, for the sake of simplicity, we decided to restrict the DePress usage only to its default set-up.

RQ2: *How cost effective is defect prediction using the DePress framework, in the default configuration, for defect prediction in an industrial software development project?*

The next step is to verify what will be the profit from the best prediction achievable using the default DePress' set-up. To achieve this, we used value of the *recall* measure corresponding to the highest F-measure value.

RQ3: *Will usage of the DePress framework pay off for an industrial project?*

To answer this question, we had to compare the costs of introducing and using the DePress based defect prediction to the potential benefits generated by its introduction. To achieve this, we used values such as return on investment (ROI) and benefit-cost ratio (BCR) [5].

1.1 Project Context and Target Software

Volvo Group, one of the leading automotive companies, was invited to take part in this research. The primary motivation for Volvo Group's interest was to verify, if their company can use DePress and its software defect prediction to increase quality and cost-effectiveness of quality assurance (QA) in their software development projects.

During our previous research, we recognized elements occurring in software projects that hindered or prevented completion of defect prediction [6]. A project selected finally as a research subject—an initiative which develops and maintains an application called Texas—was chosen due to absence of aforementioned elements.

Within the considered project, we can observe three stages of the software life-cycle (project phases): development, testing, and post-release phase.

1.2 Related Work

The first publication related to an industrial application of defect prediction was published in 1997 by Khoshgoftaar et al. [7]. It was a case study of quality modeling for a very large telecommunications system. Two other publications of Khoshgoftaar and Seliya from 2004 [8] and 2005 [9] continued with the previous concept and focused on commercial data analysis, but were not applied to a real-world environment. A similar approach can be found in publications by Ostrand and Weyuker [10], Ostrand et al. [11], Tosun et al. [12], Turhan et al. [13, 14]. Examples of industrial applications of information gathered by using defect prediction can be found in publications by Wong et al. [15], Succi et al. [16] and Kläs et al. [17]. Complete cases describing the introduction of defect prediction in industrial environments were presented by Ostrand et al. [18], Li et al. [19] and Tosun et al. [20]. Unfortunately, none of the aforementioned works contain information on cost effectiveness of applied prediction techniques and tools. To the best of our knowledge, the only research focused on the cost effectiveness of software defect prediction in an industrial project, is conducted by Monden et al. [21]. However, they investigated cost effectiveness only from the acceptance testing effort perspective and do not use any quantitative measure of potential cost of quality assurance-focused work and cost of investment during the entire software life-cycle period. Thus, in our research we also followed approaches used when cost effectiveness of other than defect prediction quality assurance technique was investigated, such as Test-Driven Development return on investment research conducted by Müller and Padberg [5].

2 Assessment Method

To investigate the cost effectiveness of defect prediction applied to an industrial software development project using the DePress framework, we developed the following plan to follow:

1. Development of a QA effort allocation strategy, based on defect prediction provided by DePress;
2. Analysis of actual, real-life costs of quality assurance for the selected release of the Texas project (4.0.0);
3. Building software prediction models for the chosen release;
4. Selection of the highest prediction *F-measure* and the corresponding *recall* measure;
5. Usage of an effort allocation strategy, based on the prediction effectiveness characterized by *recall*, to simulate a prediction-based quality assurance scenario;
6. Results analysis.

2.1 Quality Assurance Effort Allocation Strategy

In the case investigated (the release 4.0.0 of the Texas software), developers agreed that all the modules that caused 2 and more registered defects are considered as "high risk" modules. These modules accounted for 22.4 % of all modules and were responsible for 80.36 % of all registered errors and the aim was to eliminate the maximum number of software defects using the available resources within a limited time period. A similar distribution of defects in the software modules was observed by different authors [22–24] and can be interpreted as the Pareto principle existence in software quality. Additionally, in 1976 Boehm argued that defect fixing costs are the more expensive the later defects are removed [25]. That observation, which is widely called Boehm's Law [22], results in another important consequence of smart quality assurance efforts allocation: the earlier the QA actions will take place, the better it is from the perspective of the software development project's budget.

Considering the above facts, we proposed a strategy which would use the prediction model to indicate as much as possible of the mentioned "high risk" software modules (22.4 % in our case) responsible for most of the defects (80.36 % in our case), therefore helping to integrate as much as possible the QA efforts into the coding stage of the software development, while defect fixing cost is still relatively low. Such an approach should ideally decrease the total cost of bug fixing in the project and generate savings for the total project's budget [26].

If we denote M_{total} as the total number of testable software modules and H_{total} as the total number of discoverable defects, we can say that, in the project we analyzed, approximately $0.8H_{total}$ comes from approximately $0.22M_{total}$.

The impact of the prediction effectiveness on the overall effort allocation strategy can be reflected by using the *recall* measure (*Rec*)—the proportion of code units predicted as defective that were actually defective [27].

We can expect that:

$$0 < Rec < 1 \tag{1}$$

where *Rec* is the measured *recall* value corresponding to highest possible *F-measure* of defect prediction performed using the DePress framework [1]. Then, expected number of predicted modules M_i, responsible for 80 % of discoverable defects, should be:

$$M_i = 0.22 \times Rec \times M_{total} \tag{2}$$

Accordingly, we should expect that if the machine learning mechanism will be able to point out the "high risk" 22 % of software modules with the measured *recall* (*Rec*), the number of defects which can be avoided by allocation of the best quality assurance efforts on the first (development) project's phase, shall be:

$$H'_1 = 0.8 \times Rec \times H_{total} \tag{3}$$

Number of defects expected to be detected in the second and the third phase of the project:

$$H'_{2+3} = H_{total} - H'_1 \tag{4}$$

2.1.1 Return on Investment

To investigate if usage of the DePress will pay off, we will use Return on Investment (ROI) [5]:

$$ROI = \frac{Benefit - Investment}{Investment} \tag{5}$$

If the investment will not pay off, ROI is negative, otherwise positive. In our evaluation of defect prediction cost-effectiveness we will focus on potential benefits that method will generate:

$$Benefit = C_{total} - C'_{total} \tag{6}$$

where C'_{total} is the simulated total quality assurance cost in the project with defect prediction applied, and C_{total} is the actual QA cost in the project, without defect prediction.

Investment is defined as the total cost of defect prediction introduction. Moreover, *NetReturn* is calculated as *Benefit* reduced by *Investment*:

$$NetReturn = C_{total} - (C'_{total} + Investment) = Benefit - Investment \tag{7}$$

2.1.2 Benefit Cost Ratio

To analyze potential benefits from the usage of defect prediction, we will use the Benefit Cost Ratio (BCR) [5]:

$$BCR = \frac{Benefit}{Investment} \tag{8}$$

Values larger than 1 for the BCR mean a monetary gain from the DePress based defect prediction usage, while values smaller than 1 mean denote a loss.

2.2 *Actual Project's Quality Assurance Costs*

The Volvo Group policy did not allow us to publish the real costs of work invested in the project. For the purpose of research, we agreed that the man-hour cost of work by a software developer C_d will be marked as:

$$C_d = x \tag{9}$$

In that case, the average man-hour cost of work by a software tester C_t shall be, calculated according to current labor market data rates [28]:

$$C_t = 0.85x \tag{10}$$

That means, that when a tester and a developer are working together on bug fixing during the later stages of the project (not in the coding phase), the average cost per man-hour should be:

$$C_{d+t} = \frac{C_d + C_t}{2} = 0.925x \tag{11}$$

Other costs, such as infrastructure and hardware, will remain constant for the real-life and alternative (prediction-based) scenario, so they will be omitted.

Time spent on project work was traced by every team member using the JIRA tool. As a result of analysis of that data, we could obtain an average of the total time spent on fixing a single defect for each phase of the project (Table 1). The amount of time spent on quality assurance, together with the number of hours spent and number of defects fixed, divided by phases, are shown in Table 2.

According to the data in Table 2, the total number of defects discovered in release 4.0.0 are:

$$H_{total} = \sum H_{phase} = 190 + 383 + 264 = 837 \tag{12}$$

Table 1 Average defect fixing costs

Phase	1. Development	2. Testing	3. Post-release
Team members involved	Developer	Developer tester	Developer tester
Average fixing time per one defect [hours], T	1	3	3
Assumed cost of man-hour C_{hour}	C_d	C_{d+t}	C_{d+t}
Cost per one defect $C_{defect} = T \times C_{hour}$	x	2.775x	2.775x

Table 2 Actual resources consumed on defect fixing

Phase	1. Development	2. Testing	3. Post-release
Number of defects discovered H_{phase}	190	383	264
QA cost per one defect C_{defect}	x	2.775x	2.775x
QA cost per phase $C_{phase} = H_{phase} \times C_{defect}$	190x	1063x	733x

Accordingly, the total quality assurance cost is:

$$C_{total} = \sum C_{phase} = 190x + 1063x + 733x = 1985x \tag{13}$$

The ratio between defects fixed in testing and those fixed during the post-release stages is:

$$\frac{H_2}{H_3} = \frac{383}{264} \approx \frac{3}{2} \tag{14}$$

2.3 Model Construction and Prediction

Since prediction results are categorical (*faulty* or *not-faulty*), we decided to use *F-measure* and *recall* to evaluate classifiers often used in software defect prediction [29–32], which are available in the basic package of KNIME: Naive Bayes, Probabilistic Neural Network and Decision Tree.

For each classifier, four different experimental setup preparations were possible, thanks to the module-based architecture of the DePress tool.

Table 3 Prediction results: F-measure values for all experimental set-ups

Classifier	Without feature selection		With feature selection	
	Class imbalance	Class balance	Class imbalance	Class balance
Probabilistic neural network	0.167	0.72	0.24	0.74
Decision tree	0.279	0.667	0.357	0.682
Naive Bayes	0.237	0.621	0.412	0.766

2.4 The Highest F-Measure Value and the Corresponding Recall

Using the approach described in the previous section, defect prediction was performed and its *F-measure* collected (Table 3) for all four experimental set-ups, classifiers and samples. The best prediction results (the highest *F-measure* values) were obtained for the balanced class sample, slightly better with the feature selection step. Hence, we are able to answer **RQ1**: The highest *F-measure* (based on the Naive Bayes algorithm) was 0.766. The corresponding *recall* was:

$$Rec = 0.783 \tag{15}$$

2.5 Prediction-Based Costs Simulation

For the purpose of cost simulation in this scenario, where defect prediction is introduced to the project using the DePress framework, we assumed that:

- The total number of discoverable defects in release 4.0.0 (Eq. 12) is a constant value;
- The defects distribution among code is preserved;
- Average fixing cost per one defect (Table 1) is also true for the considered scenario;
- Information on location of "high risk" software modules, with *recall Rec*, will be available in the first phase of the project;
- Ratio (Eq. 14) is preserved.

Considering the *recall* value for best prediction achieved (characterized by the highest *F-measure* value) for release 4.0.0 as a result of the prediction models development (Eq. 15) and the total number of discovered defects in that release (Eq. 12), based on the proposed strategy (Eq. 3) we should expect, that the number of software issues which can be solved by allocation of the best quality assurance practices in the first, development phase of the project is:

$$H'_1 = 0.8 \times 0.783 \times 837 = 524 \tag{16}$$

Table 4 Simulated QA costs, with defect prediction used

Phase	1. Development	2. Testing	3. Post-release
Number of defects fixed H'_{phase}	524	188	125
QA cost per one defect C_{defect}	x	2.775x	2.775x
QA cost per phase$C'_{phase} = H'_{phase} \times C_{defect}$	524x	522x	347x

Regarding the number of defects which are expected to be found in later phases of the project (Eq. 4):

$$H'_{2+3} = 837 - 524 = 313 \tag{17}$$

As we assumed that ratio in Eq. (14) is preserved, the number of defects which are expected to be found in the project's second and third phase (connected) are:

$$H'_2 = 313 \times 0.6 = 188 \tag{18}$$

$$H'_3 = 313 \times 0.4 = 125 \tag{19}$$

Considering the above values, we simulated quality assurance costs assuming that the machine learning mechanism will be able to point out the "high risk" 22 % of software modules with the measured *recall* (Eq. 15), and the best quality assurance efforts will be allocated to the development phase to avoid the calculated number of defects (Eq. 16). Results of that simulation are presented in Table 4.

Total quality assurance cost in this scenario will be:

$$C'_{total} = \sum C'_{phase} = 524x + 522x + 347x = 1393x \tag{20}$$

2.5.1 Cost of Investment

Costs of defect prediction introduction were calculated as the sum of such elementary costs:

Tool acquisition and installation costs of the case investigated shall be considered as zero costs. In Volvo's organization, the DePress can be ordered and installed on user's computers without any additional costs for the project.

Training time costs—after measuring time spent on training a single person, we can state that: a developer needs to spend a maximum of 4 h on training, 1–2 h of general introduction plus another 1–2 h of training in DePress tool usage.

Data collection cost is mostly the man-hour cost of exporting the proper data from data sources and code metrics generation for two selected releases. After measuring time spent on that activity, we found that it took no more than 2 man-hours.

Table 5 Defect prediction investment costs

Activity	Time required [hours]	Cost [man-hours]
The DePress tool acquiring and installation costs	0	0
Training time costs	4	4x
Data collection cost	3	3x
Defect prediction preparation cost	1	x
TOTAL (Investment)	8	8x

Defect prediction preparation cost is the man-hour cost of defect prediction preparation (workflow creation) using the DePress tool. Results of time measurement say that creation of a proper workflow should not take more than one man-hour.

Summary of investment costs is presented in Table 5.

2.6 Results Analysis

Here, with respect to research questions **RQ2** and **RQ3** we summarize the results of our simulation (research question **RQ1** was answered in Sect. 2.4).

RQ2: *How cost effective is defect prediction using the DePress framework, in the default configuration, for defect prediction in an industrial software development project?*

As shown in Table 5, the expected total investment cost of the DePress tool-based defect prediction application in software development project is:

$$Investment = 8x \tag{21}$$

Benefit Cost Ratio (8) calculated using (6) and (21) values:

$$BCR = \frac{592x}{8x} = 74 \tag{22}$$

Such BCR value shows that we should expect a high monetary gain from the DePress tool usage for supporting quality assurance with defect prediction. Moreover, *NetReturn* (Eq. 7) from the simulated defect prediction application is:

$$NetReturn = 592x - 8x = 584x \tag{23}$$

Answering research question **RQ2**, when the defect prediction application strategy proposed in Sect. 2.1 is applied and *recall* of the prediction model will be 0.783, such an approach can result in reduction of final QA costs by almost 30 %:

$$1 - \frac{C'_{total}}{C_{total}} = 1 - \frac{1393x}{1985x} = 0.298 \tag{24}$$

Such result can be achieved only after fixing 62.6 % of the detectable bugs (Eq. 16) by effective use of quality assurance practices in the first, developmental phase of the project, on predicted "high risk" software modules.

RQ3: *Will usage of the DePress framework pay off for an industrial project?*

Simulation shows, that we should expect *Benefit* (Eq. 6) from the DePress usage in the project:

$$Benefit = 1985x - 1393x = 592x \tag{25}$$

Accordingly, expected Return on Investment (5):

$$ROI = \frac{592x - 8x}{8x} = 73 \tag{26}$$

As ROI is positive, we can state that investment will pay off.

3 Threats to Validity

In this paper, defects are not distinguished according to their severity (minor, major, etc.) and use an average, fixed time for each single defect. Omitting the severity measure in defect prediction studies is a frequent practice [33], however it can be important when simulated QA cost calculation will be compared to real-life values. In our simulation we assumed equal severity for each defect, which is reflected in an equal, average cost (see Table 1). However, when we apply the proposed effort allocation strategy into a real-life environment, we can deal with the situation when defects left undetected until the later phases of *testing* and *after-deployment* will be characterized by higher severities than defects resolved while within the coding phase. Such a situation would negatively impact overall quality assurance costs, when the DePress tool would be used for defect prediction purposes, in comparison to simulated values. This threat opens an interesting opportunity for further research.

4 Discussion

Cost effectiveness within the simulated scenario is strictly related to the quality of prediction when measured with recall (*Rec*). Based on the simulation presented, it is possible to calculate the *NetReturn* of using a proposed quality assurance effort allocation strategy for a series of defect prediction recall values. In such a way, we can observe how *NetReturn* depends on *Rec* in terms of the proposed QA effort allocation strategy.

In industrial software development projects, quality assurance (defect fixing) consumes a significant amount of time and resources. By using the defect prediction technique, project members can obtain information on possible defect-prone elements of the software, before defects will occur, to optimally plan their quality assurance process. What is proposed in this paper, is a simple effort allocation strategy which is based on the DePress framework-driven defect prediction, defects distribution and the Boehm's Law, and which eliminates most of the quality assurance work during late (after-development) project's phases. Such an approach significantly increases quality assurance costs in development phase, however overall, QA costs will decrease in comparison to actual, real-life costs observed in the investigated project, as significantly less discoverable defects are left to be fixed in the later phases, where, according to Boehm's Law, bug-fixing costs are considerably higher. At the same time, we need to mention the low investment costs, when the open source DePress framework is used for defect prediction purposes. Low investment costs and high recall of even simple defect prediction performed by default in DePress, can result with high *NetReturn* of DePress-aided quality assurance planned on a basis of the proposed effort allocation strategy. More sophisticated prediction models, especially ones using software process metrics [34, 35], may help to achieve even more impressive results. It is also worth mentioning that cross-project software defect prediction [36, 37] is sometimes used to reduce costs.

References

1. Madeyski, L., Majchrzak, M.: Software measurement and defect prediction with depress extensible framework. Found. Comput. Decis. Sci. **39**(4), 249–270 (2014). http://dx.doi.org/10.2478/fcds-2014-0014, doi:10.2478/fcds-2014-0014
2. KNIME.COM AG: KNIME Framework Documentation (2016). https://tech.knime.org/documentation/, Accessed 06 May 2016
3. Madeyski, L., Majchrzak, M.: ImpressiveCode DePress (Defect Prediction for software systems) Extensible Framework (2016). https://github.com/ImpressiveCode/ic-depress
4. Rijsbergen, C.J.V.: Information Retrieval. Butterworth-Heinemann Newton (1979)
5. Müller, M.M., Padberg, F.: About the return on investment of test-driven development. In: International Workshop on Economics-Driven Software Engineering Research EDSER-5, pp. 26–31 (2003)
6. Hryszko, J., Madeyski, L.: Bottlenecks in Software defect prediction implementation in industrial projects. Found. Comput. Decis. Sci. **40**(1), 17–33 (2015). http://dx.doi.org/10.1515/fcds-2015-0002, doi:10.1515/fcds-2015-0002
7. Khoshgoftaar, T.M., Allen, E.B., Hudepohl, J.P., Aud, S.J.: Application of neural networks to software quality modelling of a very large telecommunications system. IEEE Trans. Neural Netw. **8**(4), 902–909 (1997)
8. Khoshgoftaar, T.M., Seliya, N.: Comparative assessment of software quality classification techniques: an empirical case study. Empir. Softw. Eng. **9**(3), 229–257 (2004)
9. Khoshgoftaar, T.M., Seliya, N.: Assessment of a new three-group software quality classification technique: an empirical case study. Empir. Softw. Eng. **10**(2), 183–218 (2005)
10. Ostrand, T.J., Weyuker, E.J.: The distribution of faults in a large industrial software system. SIGSOFT Softw. Eng. Notes **27**, 55–64 (2002)

11. Ostrand, T.J., Weyuker, E.J., Bell, R.M.: Programmer-based fault prediction. In: Proceedings of the Sixth International Conference on Predictive Models in Software Engineering, pp. 1–10 (2010)
12. Tosun, A., Bener, A., Turhan, B., Menzies, T.: Practical considerations in deploying statistical methods for defect prediction: a case study within the Turkish telecommunications industry. Inf. Softw. Technol. **52**(11), 1242–1257 (2010)
13. Turhan, B., Kocak, G., Bener, A.: Data mining source code for locating software bugs: a case study in telecommunication industry. Expert Syst. Appl. **36**(6), 9986–9990 (2009)
14. Turhan, B., Menzies, T., Bener, A., Stefano, J.D.: On the relative value of cross-company and within-company data for defect prediction. Empir. Softw. Eng. **14**(5), 540–578 (2009)
15. Wong, W.E., Horgan, J., Syring, M., Zage, W., Zage, D.: Applying design metrics to predict fault-proneness: a case study on a large-scale software system. Softw. Pract. Exp. **30**(14), 1587–1608 (2000)
16. Succi, G., Pedrycz, W., Stefanovic, M., Miller, J.: Practical assessment of the models for identification of defect-prone classes in object-oriented commercial systems using design metrics. J. Syst. Softw. **65**(1), 1–12 (2003)
17. Kläs, M., Nakao, H., Elberzhager, F., Münch, J.: Predicting defect content and quality assurance effectiveness by combining expert judgment and defect data-a case study. In: Proceedings of the 19th International Symposium on Software Reliability Engineering, pp. 17–26 (2008)
18. Ostrand, T.J., Weyuker, E.J., Bell, R.M.: Predicting the location and number of faults in large software systems. IEEE Trans. Softw. Eng. **31**(4), 340–355 (2005)
19. Li, P.L., Herbsleb, J., Shaw, M., Robinson, B.: Experiences and results from initiating field defect prediction and product test prioritization efforts at ABB Inc. In: Proceedings of the 28th International Conference on Software Engineering, pp. 413–422 (2006)
20. Tosun, A., Turhan, B., Bener, A.: Practical considerations in deploying AI for defect prediction: a case study within the Turkish telecommunication industry. In: Proceedings of the Fifth International Conference on Predictor Models in Software Engineering, p. 11 (2009)
21. Monden, A., Shinoda, S., Shirai, K., Yoshida, J., Barker, M., Matsumoto, K.: Assessing the cost effectiveness of fault prediction in acceptance testing. IEEE Trans. Softw. Eng. **39**(10), 1345–1357 (2013)
22. Endres, A., Rombach, D.: A Handbook of Software and Systems Engineering. Addison-Wesley (2003)
23. Pressman, R.: Software Engineering: A Practitioner's Approach. McGraw-Hill (2010)
24. Rizwan, M., Iqbal, M.: Application of 80/20 rule in software engineering waterfall model. In: Proceedings of the International Conference on Information and Communication Technologies '09 (2009)
25. Boehm, B.W.: Software engineering. IEEE Trans. Comput. **25**(12), 1226–1241 (1976)
26. Slaughter, S.A., Harter, D.E., Krishnan, M.S.: Evaluating the cost of software quality. Commun. ACM **41**(8), 67–73 (1998)
27. Witten, I.H., Frank, E., Hall, M.A.: Data Mining: Practical Machine Learning Tools and Techniques. Morgan Kaufmann (2005)
28. Source of Information on Salaries in Poland (2015), http://wynagrodzenia.pl/, Accessed 28 Feb 2015
29. Hall, T., Beecham, S., Bowes, D., Gray, D., Counsell, S.: A systematic literature review on fault prediction performance in software engineering. IEEE Trans. Softw. Eng. **38**(6), 1276–1304 (2012)
30. Khoshgoftaar, T.M., Pandya, A.S., Lanning, D.L.: Application of neural networks for predicting faults. Ann. Softw. Eng. **1**(1), 141–154 (1995)
31. Moser, R., Pedrycz, W., Succi, G.: A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In: ACM/IEEE 30th International Conference on Software Engineering, 2008. ICSE '08, pp. 181–190 (2008)
32. Selby, R.W., Porter, A.: Learning from examples: generation and evaluation of decision trees for software resource analysis. IEEE Trans. Softw. Eng. **14**(12), 1743–1756 (1988)

33. Menzies, T., Jalali, O., Hihn, J., Baker, D., Lum, K.: Stable rankings for different effort models. Autom. Softw. Eng. **17**(4), 409–437 (2010)
34. Jureczko, M., Madeyski, L.: A review of process metrics in defect prediction studies. Metody Informatyki Stosowanej **30**(5), 133–145 (2011). http://madeyski.e-informatyka.pl/download/Madeyski11.pdf
35. Madeyski, L., Jureczko, M.: Which process metrics can significantly improve defect prediction models? An empirical study. Softw. Qual. J. **23**(3), 393–422 (2015). http://dx.doi.org/10.1007/s11219-014-9241-7, doi:10.1007/s11219-014-9241-7
36. Jureczko, M., Madeyski, L.: Towards identifying software project clusters with regard to defect prediction. In: Proceedings of the 6th International Conference on Predictive Models in Software Engineering. pp. 9:1–9:10. PROMISE '10, ACM, New York, USA (2010). http://dx.doi.org/10.1145/1868328.1868342, doi:10.1145/1868328.1868342
37. Jureczko, M., Madeyski, L.: Cross–project defect prediction with respect to code ownership model: an empirical study. e-Informatica Softw. Eng. J. **9**(1), 21–35 (2015). http://dx.doi.org/10.5277/e-Inf150102, doi:10.5277/e-Inf150102

Dynamic Stylometry for Defect Prediction

Adam Roman and Rafał Babiarz

Abstract All metrics used by QAs to assess code quality describe the characteristics of a ready-made code. In this article we propose a novel, different approach which tries to catch the dynamics of the coding process. For this purpose we utilize some basic ideas from the science of stylometry. We show how to quantify the process dynamics using several simple metrics. We also present the results of an experiment performed on a group of CS students to validate the prediction power of the proposed model.

1 Introduction

Accurate prediction of software quality, number of defects and their location in the source code has been a holy grail for software quality engineers since the very beginning of QA discipline. The ability to assess these values gives us many great opportunities for improving both software development and software quality processes. Among others, it allows us:

- to distinguish between good and error-prone modules,
- to prioritize the modules for testing and other QA activities,
- to estimate the effort for testing,
- to assess project progress and plan defect detection activities for the project manager [5],

A. Roman (✉)
Institute of Computer Science, Jagiellonian University, Kraków, Poland
e-mail: roman@ii.uj.edu.pl

A. Roman
Rivet Group, Warsaw, Poland

R. Babiarz
UBS, Kraków, Poland

L. Madeyski et al. (eds.), *Software Engineering: Challenges and Solutions*,
Advances in Intelligent Systems and Computing 504,
DOI 10.1007/978-3-319-43606-7_7

- to estimate the final product quality level, together with the number of defects remaining in the code [5],
- to improve capability and assess process performance for process management [5].

The ability to control and describe the actual and predicted final software quality is a crucial tool for defining the cost and economics of software quality and their relationship to business value [10]. In this article we understand the software quality in the simplest possible way: as the defect density, that is, the number of defects per LOC.

There exist many defect models for predicting software quality. The vast majority are parametric models of the form $y = f(x_1, \dots, x_n)$. Here y stands for the predicted defect density and x_i values correspond usually to metrics describing some static properties of the code (cyclomatic complexity, Lorenz & CK metrics for OO programs etc.) or to design metrics that predict complexity of the yet non-existent code (function points, object points and so on). We measure all x_i values, put them into the formula and compute the value of y, which is the model prediction of defect density for a given input. There is also a statistical approach that utilizes the distribution of defect detection in time. The most classical among such models are Rayleigh [17, 18] (used for the whole life cycle), exponential (used only for the late, testing phases) or S-shaped [19]. The problem with parametric models is that their accuracy heavily depends on the environment, programming language used, project type, team experience and other factors not included explicitly in the model formula. A model can be quite accurate for some type of projects and—at the same time—it can return completely incorrect values for projects of different type. If statistical model is to be used, we have to feed it with some initial data, so it cannot be used in the initial project phases. More information on parametric and statistical models can be found in [11]. A survey on other approaches can be found in [1]. Table 1 gathers some examples of well-known metrics used in defect models, together with their pros and cons.

Another approach for predicting software quality utilizes mutation analysis. Mutation involves program modification by introducing small changes to the code. A change is injected into the code by a so-called mutation operator (for example, arithmetic mutation operator may replace expression `x=x+y` with `x=x-y`, `x=x*y` etc.). Each mutant simulates a programmer's mistake. If, for a given SUT P, its mutant M, and test t the output of P run on t differs from the output of M run on t, we say that the mutant was killed by t, because we detected an error [4]. Having a set T of tests and SUT P we generate n mutants of P and run all tests on all mutants. Assume that our test suite T detected k defects in P and killed m out of n mutants. We may estimate now that T is able to find $m/n \cdot 100\,\%$ of all defects. If it detects k real defects, we estimate that their total number was $k \cdot n/m$, so after removing these k defects there should remain $k \cdot (n - m)/m$ defects in the code. This approach works fine if we are able to provide a good set of mutation operators (meaning that mutations reflect the real mistakes done by developers) and we have a good test suite in advance. Unfortunately, although very effective, mutation analysis is very time- and resource consuming. Also, some mutants may be equivalent to the original program,

Table 1 Examples of metrics used in different defect models

Metric	Remarks
LOC, KLOC [3]	Measures volume of the source code in terms of number of lines. Before using this metric one needs to define precisely what is understood by the “line of code” (executable lines? physical instructions? lines in the program text? comments? etc., see [9]). Once the definition is established, collecting this metric is easy, cheap and simple
Cyclomatic complexity [15]	Also known as McCabe complexity. Measures the complexity of the code structure in terms of the number of independent paths in a control flow graph for a given program. Another way to measure it is by counting the number of decision points in the code plus one. Relies only on a code structure and does not take into account other aspects of the code, like data flow, syntax structures etc.
Halstead’s [7]	A set of metrics that take under account the number of lexemes (tokens) used in the code. Lexem can be operator or operand. There is a controversy about so-called Halstead’s Software Science that uses these metrics, because the predicted values in Halstead models (like program size) are directly related to the metrics values, so the correlation between these two is apparent—it holds by definition
Function points [2]	Similar to LOC/KLOC, but instead of the physical size of the program they measure its ‘amount of functionality’. FP metric can be computed having only software architecture plan, without a single line of code written, so it is useful in prediction. From the other hand, the methodology for computing FP is quite complex and the whole process is time consuming
Code churn [12, 16]	They measure the degree to which the program—understood as a set of modules or files—has changed during the development. In order to compute them one needs to have at least several versions of each file, so they cannot be used at the beginning of the development process
CK, Lorenz [4, 13]	Classical CK (Chidamber-Kemerer) and Lorenz sets of metrics are used to describe the structure and dependencies between artifacts created within the object oriented paradigm. The examples of these metrics are: weighted number of methods in class, depth of inheritance tree, coupling between objects, lack of cohesion, LOC per method etc.

which may influence the analysis results. It is known that, in general, the mutant equivalence problem is undecidable. See [8] for the survey on mutation testing.

Finally, there is an approach that utilizes not product, but process metrics. A few examples of such metrics are [14]:

- number of revisions,
- number of distinct committers,

- number of modified lines,
- number of defects in previous version.

Some examples of models built on this approach can be found in [12, 16]. They are able to estimate the defect density by analyzing the code churn which measures the changes made to a component over a period of time. This approach belongs to the family of parametric models, but the model variables describe metrics like churned LOC, deleted LOC, files churned etc. instead of the values related to structural properties of the source code itself. The model can be used only if we are able to collect large enough repository containing many versions of each analyzed file. Therefore, it cannot be used in the initial phases of the project, neither in the small projects with only a few planned releases.

All models described above utilize static properties of the code. Even the churn model, which takes into account the differences between consecutive versions of modules, operates on static fragments of already written pieces of code. But no program is created in a flash. It is a process extended in time and this process consists entirely of particular developer activities. Therefore, we asked ourselves the following question:

Does the *way* the program is created affect the final product quality?

Look at how programmer works: she does not write code in a uniform, steady way. She may start writhing the code from the middle of the module, then jump to the beginning, copy and paste some portion of the code, delete some lines, replace one instruction with another, slow down, speed up, go away from the computer for some time, and so on. It is a very dynamic process which can be described by some "low-level" metrics (see Fig. 1). It seems reasonable to assume that the coding style

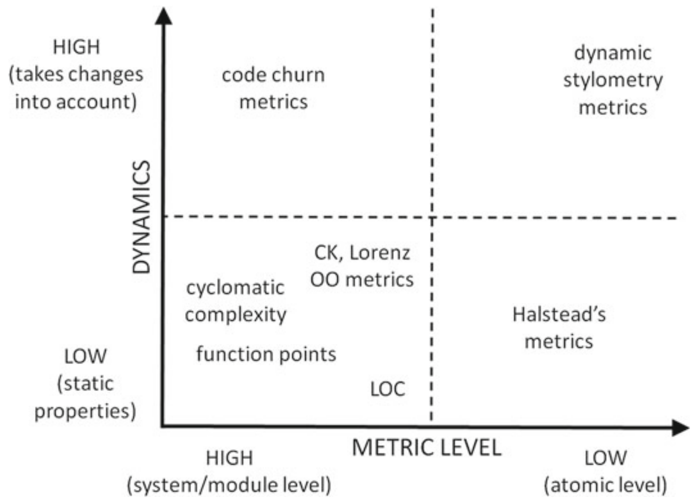


Fig. 1 Metrics taxonomy regarding two dimensions: level of dynamics and level of detail. The proposed new metrics are dynamic, but unlike code churn metrics they focus on a low-level developer activities, not static properties of the whole, already written modules

of a professional, experienced developer who rarely makes mistakes differs from the coding style of a first-year graduate CS student writing buggy programs. So, does all these programmer activities influence on the probability of making a mistake? To check this hypothesis we designed a set of so-called dynamic stylometry metrics (see Sect. 2) that quantitatively measure the dynamics of code creation. Then we conducted an experiment on a group of CS students (see Sect. 3). The experiment was necessary, as—to the best of our knowledge—there are no publicly available data describing the dynamics of a coding process. The collected data was used to build a mathematical model (described in Sect. 4) that predicts defect density. In Sect. 5 we present the conclusions and future work.

2 Dynamic Stylometry Metrics

We describe the coding process with nine simple metrics based on the six types of actions (see Table 2).

The action in our model represents a single unit of programmer's work and introduces one change, but can result in changing more than one character in the source code. The actions allow us to completely recreate the coding process. Each action corresponds to the metric describing the total time spent on a given activity. We denote these six metrics in the same way as the actions (T, BT, R, BR, RP, FL). Let $A_1, A_2, \ldots$ be the consecutive programmer's actions. Because the real time of inserting, replacing or deleting text is close to 0, we define the time of A_n as the time

Table 2 The set of six dynamic stylometry actions

Action	Action name	Description
T	Type	Corresponds to entering a single character in the editor. May result by pressing a key on a keyboard or by pasting one letter
BT	Bulk type	Corresponds to a situation, when a single action results in appearance of more than one character in the editor. Usually results from copy and paste operation or from using statement completion popular in most programming IDEs
R	Remove	Complementary to typing. Corresponds to removing a single character by pressing Delete or Backspace key when no large portion of text is highlighted
BR	Bulk remove	Corresponds to removing more than one character from the editor. Caused by highlighting a large portion of text and pressing Delete or Backspace key
RP	Replace	Corresponds to simultaneous (bulk) remove and (bulk) type. Caused by highlighting a large portion of text and invoking 'paste' operation
FL	Focus lost	Corresponds to a situation, when user switches from IDE to another application

between A_{n-1} and A_n. Time of A_1 is equal to time between IDE startup and timestamp of action execution. We ignore time of every action directly preceding focus lost FL, because the real time of FL is itself non-zero, so we are not able to split it between FL and the action that follows it.

Apart from metrics T, BT, R, BR, RP, FL we define three other ones:

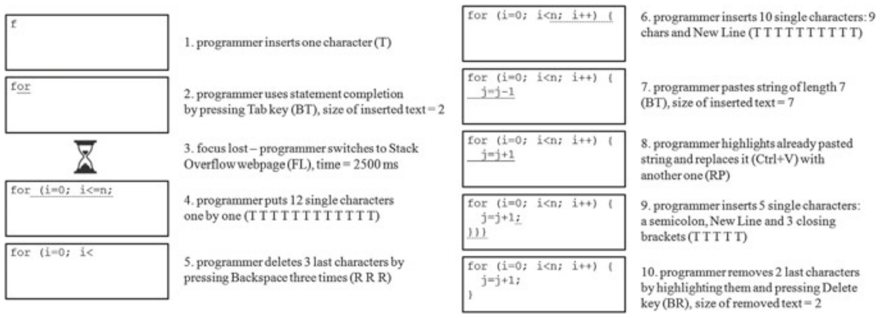
- *BTAL*—average length of bulk-typed text,
- *BRAL*—average length of bulk-removed text,
- *CU*—code used (computed as the ratio of the final code size and the total size of the code that was written).

Low values of *CU* may indicate that the programmer was not well prepared to the programming task or that she didn't understand properly the architectural design of the created module. Figure 2 presents the exemplary session of code creation. Underlined solid line denotes action *BT* (bulk type) and dashed line denotes action *T* (type). The process shown in Fig. 2 can be described by the following set of actions:

$$T, (BT), FL, T^{12}, R^3, T^{10}, BT, RP, T^5, BR,$$

where X^n denotes n repetitions of action X. According to the rules given above, we do not count time for actions in parentheses. Assuming that the time between each two actions is 1000 msfd (except for FL, for which it is 2500 ms) we have the following metrics values:

$T = 28000\,\text{ms}$, $BT = 1000\,\text{ms}$, $R = 3000\,\text{ms}$, $BR = 1000\,\text{ms}$, $RP = 1000\,\text{ms}$, $FL = 2500\,\text{ms}$, $BTAL = (2 + 7)/2 = 4.5$, $BRAL = 2/1 = 2$, $CU = 32/44 \approx 0.73$.



```
f
```
1. programmer inserts one character (T)

```
for
```
2. programmer uses statement completion by pressing Tab key (BT), size of inserted text = 2

3. focus lost – programmer switches to Stack Overflow webpage (FL), time = 2500 ms

```
for (i=0; i<=n;
```
4. programmer puts 12 single characters one by one (T T T T T T T T T T T T)

```
for (i=0; i<
```
5. programmer deletes 3 last characters by pressing Backspace three times (R R R)

```
for (i=0; i<n; i++) {
```
6. programmer inserts 10 single characters: 9 chars and New Line (T T T T T T T T T T)

```
for (i=0; i<n; i++) {
  j=j-1
```
7. programmer pastes string of length 7 (BT), size of inserted text = 7

```
for (i=0; i<n; i++) {
  j=j+1
```
8. programmer highlights already pasted string and replaces it (Ctrl+V) with another one (RP)

```
for (i=0; i<n; i++) {
  j=j+1;
}}}
```
9. programmer inserts 5 single characters: a semicolon, New Line and 3 closing brackets (T T T T T)

```
for (i=0; i<n; i++) {
  j=j+1;
}
```
10. programmer removes 2 last characters by highlighting them and pressing Delete key (BR), size of removed text = 2

Fig. 2 The dynamics of code creation and its relation to dynamic stylometry actions

3 The Experiment

In order to verify our hypothesis that the way the program is created influences its final quality, we had to build the model and validate it. Unfortunately—to the best of our knowledge—there is no publicly available data that could allow us to compute our metrics. Therefore, we had to build such dataset from scratch. It required to perform an experiment, which we did on a group of Computer Science students. Most of them were 2nd year undergraduate. Each of them had to implement several Java methods for a program called 'Electronic Timetable'. The application was a timetable manager for a municipal transportation company. It also had to provide a content for electronic screens displaying the nearest bus arrival times according to a predefined set of rules.

Students were implementing the system in Eclipse environment with a special plug-in that was responsible for gathering the data needed to compute all our nine metrics. We received 86 solutions out of which 20 failed on compilation or did not fulfill any of the task requirements. Further 15 solutions were removed due to the very poor code quality resulting in failing on almost all tests. Hence, the final dataset for the analysis consisted of 51 valid solutions. Apart from the solutions themselves, all participants had to perform a self assessment about their programming abilities. They had to grade it on a 5-level Likert scale (1 = almost no experience in programming, 5 = an expert in programming).

In order to measure the quality, all received programs were tested using JUnit framework with a set of 93 unit tests. Tests were carefully designed in order to cover all functionalities and possible boundary values, decisions and conditions that might occur in the submitted solutions. Figure 3 shows a noticeable relation between the self assessed programming skills and the fraction of failed tests. This means that students were able to accurately predict their real abilities as programmers. The fractions of students with grade resp. 1, 2, 3, 4, 5 are resp. 6 %, 18 %, 47 %, 27 % and 2 %.

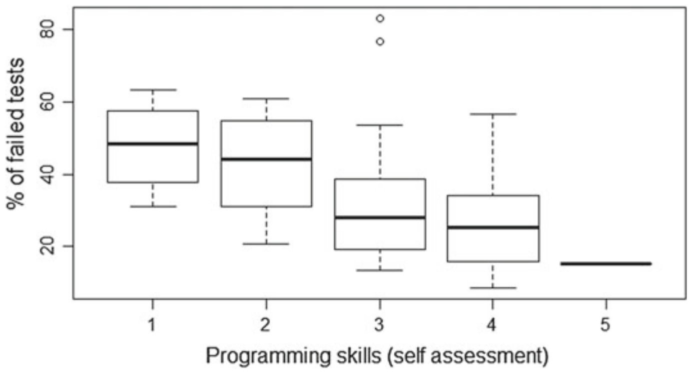


Fig. 3 Students self assessment versus percentage of failed tests in their solutions

It guarantees that our model described in the next section was trained on programs written by programmers with a wide spectrum of experience.

4 Model and Experiment Results

For all statistical computations we used the R Statistical Package tool. Predictive models should be as simple as possible—that is why we decided to build a simple, linear model of the form $y = \sum_i \alpha_i x_i$. The dependent variable y represents the defect density (number of defects per LOC). As we were not capable to count defects in all submitted solutions, we represented defect density as a ratio of failed tests and LOC. We took all 9 metrics and all $9 \cdot 8 = 72$ possible ratios between them as the candidates for independent variables x_i. Next, we used R procedure `glmulti` to choose the best possible subset of them, that is, a subset that gave us the best model in terms of the Akaike Information Criterion. This criterion takes into account not only the model error, but also the number of model parameters (the smaller, the better). Each ratio represents one main effect, that is, a factor that impacts on the final defect density. We allowed the model to include not only main effects, but also pairwise interactions between them. The final model for our data has the following form:

$$\begin{aligned} DD = & -0.76 \cdot CU + 0.035 \cdot BTAL + .0182 \cdot F/R + 0.008 \cdot CU \cdot BRAL \\ & -0.0003 \cdot BTAL \cdot BRAL + 5.09 \cdot CU \cdot BR/T - 0.18 \cdot BTAL \cdot BR/T \\ & +0.004 \cdot CU \cdot BT/F + .177 \cdot CU \cdot RP/BR - 0.004 \cdot BTAL \cdot RP/BR \\ & -0.054 \cdot (F/R) \cdot (RP/BR), \end{aligned}$$

where DD is the predicted defect density and CU, BAL, $BRAL$, BR, BT, T, RP, F and R are nine dynamic stylometry metrics. Notice the high negative impact of CU on the dependent value. This result seems to confirm our hypothesis from Sect. 2 that low values of CU may reflect unpreparedness of the developer and hence, the low quality of her work.

The correlations between the independent variables are mostly between -0.3 and 0.3. This means that the predictors seem to be independent and that there is no collinearity between variables. All 11 estimations of model variables were statistically significant. Residual standard error was 0.085 on 40 degrees of freedom. Adjusted multiple R-squared value was 0.91, which means that our model explains ca. 90 % of the total variance in the dataset. Figure 4 shows the relation between the real defect density and the predicted one. Correlation between these two values is high (Pearson r coefficient equals 0.68).

5 Conclusions

In this paper we presented a new set of software metrics which may be used as predictors in software defect prediction models. It is a well-known fact that such models receive much critique. The models are weak “because of their inability to cope with the, as yet, unknown relationship between defects and failures” [6]. However, it was not our intention to build a complete and mature defect prediction model. Rather we wanted to present a new type of metric and show that it *may* have a potential to be a good defect predictor. The simple experiment performed on a group of students seems to support this claim. The regression coefficient of the linear regression for data points in Fig. 4 is statistically significant. This means that the higher the defect prediction is, the higher is the real one. In comparison to classical metrics, stylometry metrics reflect a more holistic view on the software development process and are independent of many (often problematic) technical issues. They concentrate on the design process performed by developer, which makes them more universal than “static” metrics.

5.1 Threats to Validity

Presented model seems to work quite well, although we are aware of several caveats. First, the defect density is not computed directly, but using the number of failed tests metric. We decided to choose this approach, because in many submitted solutions there were large parts of functionality missing. However, this might cause the overestimation of the real defect values. For example, one “elementary” bug might result in a failure of (almost) all tests for a given class.

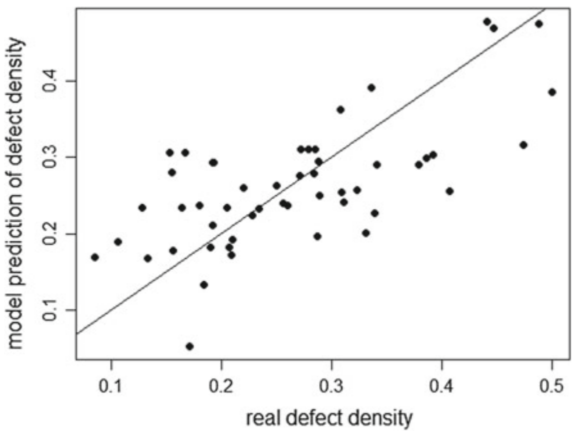


Fig. 4 Relation between the real defect density and its prediction from our model. Each point represents one submitted solution. The values concentrate around y = x line, which shows good model accuracy

Second, the average size of all submitted programs was very small. Students had to implement 10 functions in 2 h. This certainly cannot be compared to the real programs developed in the IT industry. Also, students were not professional developers. Some of the submitted programs failed to compile. Hence, the sample size was relatively small. We were able to analyze only 51 out of 86 submitted solutions.

Third, the response variable in our model is not normal (p-value for the Shapiro-Wilk test was around 0.002): the distribution is slightly right-skewed. Of course we could use some statistical transformations to normalize the variable (e.g. Box-Cox transformation) or use a general linear model with Poisson-type response. However, our intent was just to check in general whether the stylometry metrics are able to predict defect density. Therefore, we have used a simple linear model slightly abusing its assumptions. Another approach would be to use some more sophisticated, nonlinear model, like neural network, nonlinear regression or regression trees.

Finally, there is an important psychological argument that may make the whole measurement process difficult. In order to measure the stylometry metrics, the developers' IDE needs to be instrumented: we need to use a tool, similar to the keylogger applications, that collects data from this IDE. This may be an issue, especially for the companies with strict security policies. Second, no one likes to be controlled: if a developer is aware that all her actions are recorded, this may impact on her productivity. It is therefore crucial for developer to understand what type of information is recorded, why it is recorded and that no private information will be logged.

5.2 Discussion

Despite these shortcomings, there are also some good news. First, the model does not necessarily need to predict the exact defect density. Even if burdened with prediction error, it can still allow us to prioritize the analyzed modules regarding the defect density. Having such prioritization we can group modules into several classes: the heavily error-prone ones that require intense QA activities (like exploratory testing, inspections, code reviews etc.); "fair enough quality" modules that don't necessarily need to go through a careful QA process; "clean" modules that probably have no (or low number of) defects; and so on. Using the Pareto rule we can select and analyze small sample of modules that contain majority of all defects, therefore optimizing somehow the trade-off between our effort and possible profit in terms of quality level.

Second, notice that we did not use any metric regarded as a good defect predictor (like FP, CK metrics, cyclomatic complexity and so on) in our model, and yet we were still able to obtain a good prediction rate. This means that the dynamic stylometry metrics may also serve as good defect predictors. It may be worthwhile to build a model that contains both classical, well-known metrics and dynamic stylometry metrics. Such combined models may be much more accurate. Our metrics can also be used in classification models, where each class/module/unit is classified as error-prone or as a clean one.

Third, the way of creating programs should be independent of the established project methodology, product life-cycle, programming language used and other environmental factors. At least, the environment impact on the dynamic stylometry model should be much smaller than in case of models using classical, well-known metrics like LOC, cyclomatic complexity, function points and so on. This makes the stylometry metrics more universal. We think that a stylometry model trained on the data from one particular project done in a particular company should predict well the defect density in other projects done in other companies, as stylometry factors should be environment-independent. Unfortunately, verifying this hypothesis requires a lot of experimental work, which we were unable to perform.

5.3 Future Work

The successive work on dynamic stylometry metrics should focus on a wide research performed in a real IT development environment. The experiment like the one described in this paper should be repeated on a large group of professional programmers. The model should also contain metrics recognized in a literature as good defect predictors. It should be checked if adding a stylometric metric to the known models improve the model accuracy. We believe that such approach will allow us to predict the defect density yet more accurately.

Acknowledgments Both collected data and the detailed description of the exercise used in our experiment are available for other researchers from www.ii.uj.edu.pl/~roman/DynamicStylometry.

References

1. Abaei, G., Selamat, A.: A survey on software fault detection based on different prediction appoaches. Vietnam J. Comput. Sci. **1**, 79–95 (2014)
2. Albrecht, A.J.: Measuring application development productivity. In: Proceedings of the Joint SHARE, GUIDE, and IBM Application Development Symposium, California, pp. 83–92 (1979)
3. Boehm, B.: Software Engineering Economics. Prentice-Hall, Englewood Cliffs (1981)
4. Chidamber, S., Kemerer, C.: A metrics suite for object-oriented design. IEEE Trans. Softw. Eng. **20**, 476–493 (1994)
5. Clark, B., Zubrow, D.: How good is the Software: a review of defect prediction techniques. In: Software Engineering Symposium, Carreige Mellon University (2001)
6. Fenton, N.E., Neil, M.: A critique of software defect prediction models. IEEE Trans. Softw. Eng. **25**(5), 675–689 (1999)
7. Halstead, M.H.: Elements of Software Science. Elsevier (1977)
8. Jia, Y.: An analysis and survey of the development of mutation testing. IEEE Trans. Softw. Eng. **37**(5), 649–678 (2011)
9. Jones, C.: Programming Productivity. McGraw-Hill, New York (1986)
10. Jones, C., Bonsignour, O.: The Economics of Software Quality. Addison-Wesley, Upper Saddle River, NJ (2012)

11. Kan, S.: Metrics and Models in Software Quality Engineering, 2nd edn. Pearson Education, Boston (2003)
12. Kpodjeto, S., et al.: Design evolution metrics for defect prediction in object oriented systems. Empir. Softw. Eng. **16**(1), 141–175 (2011)
13. Lorenz, M.: Object-Oriented Software Development: A Practical Guide. Prentice Hall (1993)
14. Madeyski, L., Jureczko, M.: Which process metrics can significantly improve defect prediction models? An empircal study. Softw. Qual. J. **23**, 393–422 (2015)
15. McCabe, T.J.: A complexity measure. IEEE Trans. Softw. Eng. **2**(4), 308–320 (1976)
16. Nagappan, N., Zeller, A., Zimmermann, T., Herzig, K., Murphy, B.: Change bursts as defect predictors. In: Software Reliability Engineering, pp. 309–318 (2010)
17. Putnam, L.H.: A general empirical solution to the macro software sizing and estimating problem. IEEE Trans. Softw. Eng. **SE-4**, 345–361 (1978)
18. Putnam, L.H., Myers, W.: Measures for Excellence: Reliable Software on Time Within Budget. Yourdon Press, Englewood Cliffs (1992)
19. Yamada, S., Ohba, M., Osaki, S.: S-shaped software reliability growth models and their applications. IEEE Trans. Reliab. **33**(4), 289–292 (1984)

Mutant Generation for WSDL

Ilona Bluemke and Wojciech Grudziński

Abstract Web Service technology is becoming increasingly popular because it simplifies the use and integration of existing applications and creation of new services. Because available services should be of high quality effective testing of Web Services is essential. The idea of mutation testing of Web Services is discussed in this paper and some mutation operators are described. A tool named Web Services Mutant Generator (WSMG), supporting the mutation testing of Web Services, is also presented. This tool accepts files with the description of Web Services written in WSDL (Web Services Description Language) and generates altered (mutated) versions of services using four mutation operators: OTCE, OTCA, STCE and STCA. The architecture and the implementation of the WSMG tool is shortly described and an example of its application is included.

Keywords Mutation testing · Mutation operators · Web services · WSDL

1 Introduction

The W3C [1] (World Wide Web Consortium) defines Web Service as "a software system designed to support interoperable machine-to-machine interaction over a network". Web Services are independent Internet applications based on SOAP [2] and XML [3] technology. Several protocols and standards, such as the Web Service Description Language (WSDL) [4], the Universal Description Discovery and Integration (UDDI) [5] protocol or the Web Service Inspection Language (WSIL) [6], and the Simple Object Access Protocol (SOAP) [1] are used to describe what the Web Service delivers and how it interacts with other services. As WSDL,

I. Bluemke (✉) · W. Grudziński
Institute of Computer Science, Warsaw University of Technology,
Nowowiejska 15/19, 00-665 Warsaw, Poland
e-mail: I.Bluemke@ii.pw.edu.pl

W. Grudziński
e-mail: W.Grudzinski@stud.elka.pw.edu.pl

L. Madeyski et al. (eds.), *Software Engineering: Challenges and Solutions*,
Advances in Intelligent Systems and Computing 504,
DOI 10.1007/978-3-319-43606-7_8

UDDI, WSIL, and SOAP are all based on XML, the Web Service does not depend on a hardware platform, a language and a service vendor. Therefore, developers of a Web Service can't assume which clients will use the Web Service, also the developers at the client side, are not aware of the platform and the language used at the server side. A Web Service and client software using it often is implemented in different programming languages and on different platforms.

WSDL (Web Service Description Language [4]), a standard established by the W3C [1], is usually used to describe Web Services. Interfaces with operations and data types used or returned by Web Services are specified in WSDL. Some system use SOAP [2] messages to interact with Web Services, in a manner defined by the WSDL description.

Because of general availability of Web Services, they normally need to be of high quality, so they need to be thoroughly tested. The cost of testing is usually a significant part of the total cost of the software development and sometimes this cost constitutes more than 50 % of the overall amount [7]. The testing of services is especially difficult due to service architecture, organization and collaboration of services. Mutation testing is one of approaches to the testing of software, that was proposed more than 40 years ago. In mutation testing the faults are deliberately introduced into the program and a set of faulty programs called mutants are built. Each mutant program is produced by a particular change in the original program. This change is defined by a mutant operator e.g. substitution of the multiplication by the division operator. Mutants are executed against the set of input data to check, if inserted faults can be detected, and this process enables to assess the quality of a given set of tests. Jia and Harman prepared a very good survey of mutation techniques [8].

The goal of our research was to examine mutation testing of Web Service described by its WSDL document.

An application, named WSMG—Web Services Mutant Generator, supporting the mutation testing of Web Services has been designed and is being implemented at the Institute of Computer Science, Warsaw University of Technology. Currently, it is responsible only for the first part of the mutation testing process: generates mutated versions of WSDL description. The tool is still under development and components, which are used in further steps of the testing process, are being implemented.

The organization of this paper is as follows. Key features of Web Services testing are identified in Sect. 2. WSDL is also briefly described there. Section 3 discusses mutation testing of Web Services as well as some mutation operators and testing approaches. Section 4 focuses on the architecture of the developed mutant generator (WSMG) and presents some examples of using the developed tool. Finally, Sect. 5 contains some conclusions and future research directions.

2 Testing Web Services

Precise separation of roles between the users and the provider, the owner, and the developers of a service, as well as the dynamic nature of Web Service cause new challenges to the testing of service system. Canfora and Di Penta [9] discussed special characteristics of services, that increase the complexity of the testing process.

2.1 Levels of Testing

The testing of Web Services, similarly to testing any other software, can be performed at different levels:

- testing of atomic services (unit testing),
- testing the composition of services,
- testing the integration and the interoperability of services,
- regression testing,
- testing of non-functional properties.

Testing of services has been receiving a lot of attention in recent years. Papers published in this area are dealing with different levels of testing, unit testing of services and orchestrations, integration testing, testing of non-functional properties and regression testing. Some research is also dedicated to the improvement of testability of services and service-oriented systems. Canfora and Di Penta [9] survey research in testing of services and identified several problems, which remain open and need more work. These problems are concerned with validating decentralized systems, improving testability, integrating testing and run-time verification, etc. Others surveys on testing of Web Services are given in [10–13]. Ladan [10] classifies approaches to testing Web Services as:

1. WSDL-based test case generation,
2. mutation-based test case generation (e.g. [14, 15]),
3. test modeling (e.g. [16]),
4. XML-based approaches (e.g. [17]).

A framework, that can be used to test the robustness of a Web Service, was proposed by Hanna and Munro [18]. The analysis of the Web Service Description Language (WSDL) document of service gives information helping for the identification of faults. These faults could affect the robustness, so test cases detecting them can be designed. The WSDL document is also used by Bai and Dong [19] to generate tests in a testing framework that contains test case generation, test controller, test agents and test evaluator. An example of a tool (named WSDLTest) for automatic testing of Web Services based on WSDL documents is described in [20]. This tool can support testing Web Services described in WSDL 1.1 or WSDL 2.0

document. In [21] another tool, but of the same name, is described. In this tool, WSDL schemas are used to generate service requests. The tester writes pre-condition assertions to which the generated requests are adjusted. The tool dispatches the requests and captures the responses and after testing verifies the response against the post-condition assertions prepared by the tester.

It often happens that a single service is not able to fully satisfy the user but a composition of some existing services may be satisfying. The process of integrating services is called Web Service Composition (WSC). The discovery, selection and composition of Web Services attracted many researchers but the testing of Web Service composition is still immature [22]. An interesting overview and evaluation of current approaches to WSC testing was published by Rusli et al. [13]. Surveys on Web Service composition include Bucchiarone et al. [23] and Zakaria et al. [24].

2.2 Types of Testing

Different categorization of Web Services, proposed by Chen et al. [25] identifies the following types of testing:

- WSDL document testing,
- SOAP message testing,
- testing published features and bindings of an SOA system,
- testing synchronous Remote Procedure Call (RPC) and the asynchronous notification of Web Services,
- testing the intermediary capability of the SOAP messages, monitoring the SLA (service-level agreement) and the Quality of Service (QoS) levels,
- performance testing of Web Services.

Testing facilities in Chaos Monkey [26] can also be used for checking already operational Web Services. Chaos Monkey service identifies groups of operating programs and randomly terminates one program from a group. The service operates during business hours so the staff can immediately locate and correct the error. This approach can't be used at the early stages of the service development as above listed approaches.

Several tools, supporting the testing of Web Services, are available on the market and the most noticeable among them are: HP QuickTest Professional [27], Parasoft SOAtest [28], SOAPSonar [29], SoapUI [30].

The focus of this paper is the WSDL-based and mutation-based approach to the testing of Web Services.

2.3 WSDL

WSDL [4], Web Services Description Language based on XML, defines functions provided by a Web Service and its environment. WSDL contains all details about the service, for example, the required parameters and the returned results. XML Schema contains the description, in WSDL, of messages sent and received from the service. SOAP messages are used for the communication with the Web Service and they are described by WSDL as operations. WSDL specifies the format of interfaces, it can also describe the interactivity of a service. WSDL description contains the possible but not necessarily required interactions. W3C [1] currently recommends WSDL in version 2.0 but the older version—WSDL 1.1 is still common and some software supports only this version. The structures of these versions are shown in Fig. 1.

A WSDL document consists of the following sections (based on [32]):

- **Definition**: a container, inside of which the remaining sections are located. It defines the name of the service, declares multiple name spaces used throughout the remainder of the document, and contains all service elements described below.
- **Data types**: the data types used in the messages are in the form of XML schemas. When simple types are used the document does not need to have a types section.
- **Message**: an abstract definition of the data, in the form of a message presented either as an entire document or as arguments to be mapped to a method invocation.

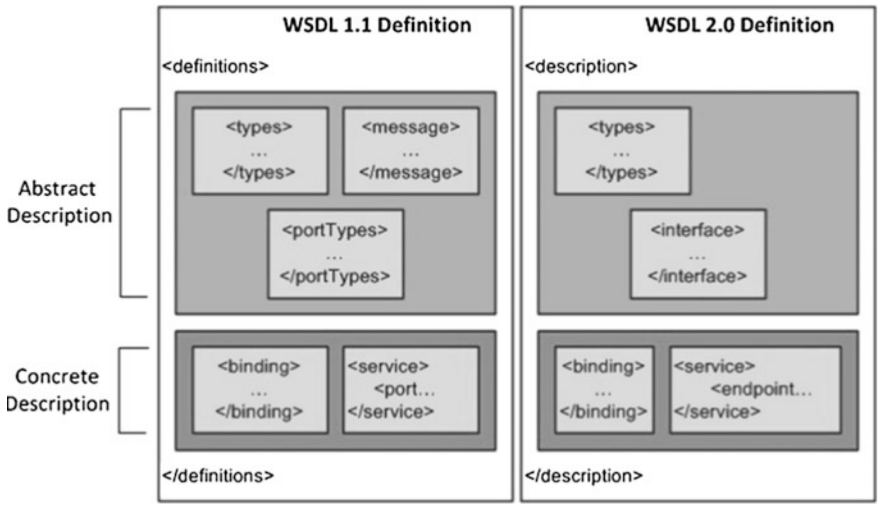


Fig. 1 Structure of WSDL documents (based on [31])

- **Operation**: abstract definition of the operation for a message, such as naming a method, message queue, or business process, that will accept and process the message.
- **Port type**: abstract set of operations mapped to one or more end-points, defining the collection of operations for a binding; the collection of operations, as it is abstract, can be mapped to multiple transports through various bindings.
- **Binding**: protocol and data formats for the operations and messages defined for a particular port type.
- **Port**: a combination of a binding and a network address, providing the target address of the service communication.
- **Service**: a collection of related end-points encompassing the service definitions in the file; the services map the binding to the port.

Additionally, the WSDL specification also defines the following sections (based on [32]):

- **Documentation**: contains comments, documentation and can be included inside any other WSDL element.
- **Import**: used to import other WSDL documents or XML Schemas.

3 Mutation Testing of Web Services

The general idea of mutation testing of Web Services is presented in Fig. 2. The tester provides a WSDL document with a Web Service to be tested. The mutation tool applies the mutant operators (marked as “A”) selected by the tester. For each mutant, SOAP messages are also generated (marked as “B”) and respective clients are created. The tool sends these SOAP messages to the Web Service under testing, receives responses (marked as “C”) and presents them to the tester so that they can be analyzed.

Nine **mutation operators** were suggested by Siblini and Mansour in [14]:

- Switch Types Complex Type Element (STCE),
- Switch Types Complex Type Attribute (STCA),
- Occurrence Types Complex Type Element (OTCE),
- Occurrence Types Complex Type Attribute (OTCA),
- Special Types Element Nil (STEN),
- Switch Types Simple Type Element (STSE),
- Switch Types Simple Type Attribute (STSA),
- Switch Messages Part (SMP),
- Switch Port Type Message (SPM).

The first four were implemented (STCE, STCA, OTCE, OTCA) in our WSMG tool. The chosen group of operators is responsible for manipulating the complex

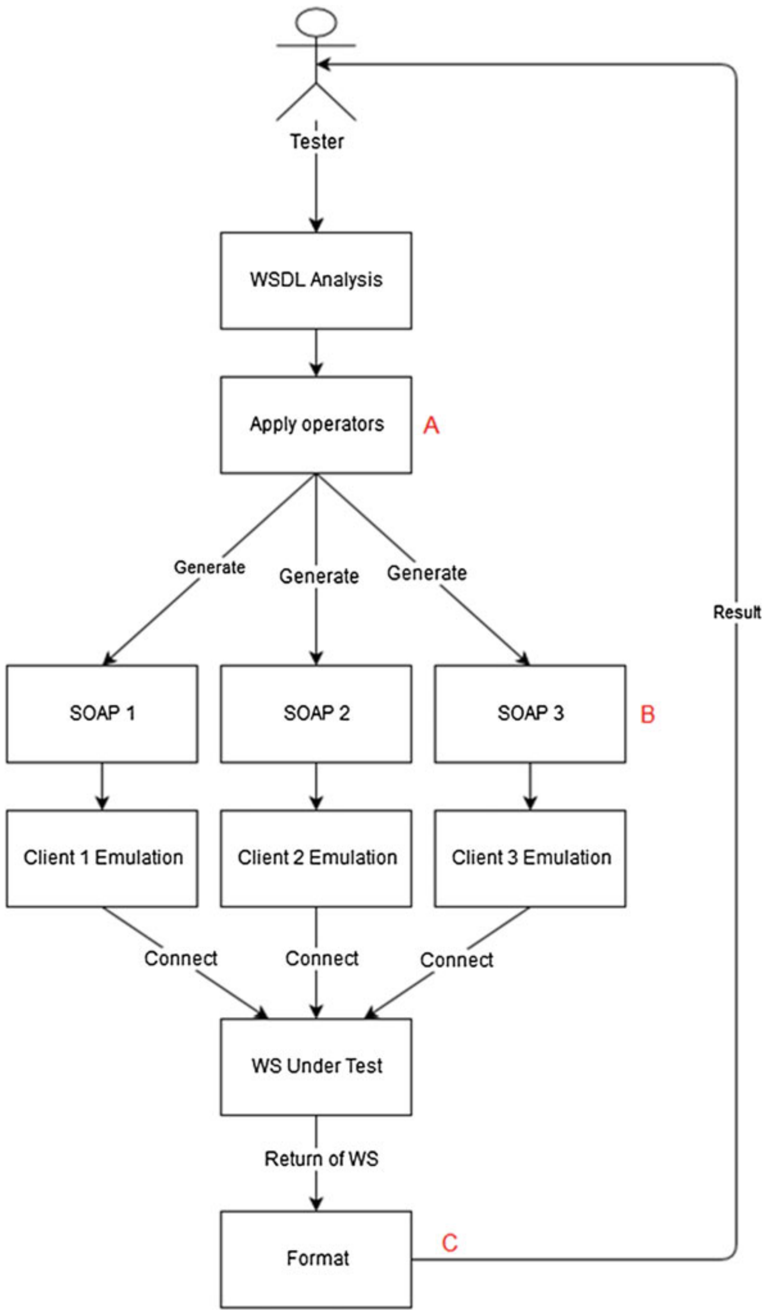


Fig. 2 Mutation testing of WS based on [15]

types part of a WSDL document and they represent possible faults related to the Web Service implementation.

The Switch Types Complex Type Element (STCE) operator is defined as the operator that will switch elements of the same type (e.g. strings, integers) within a single complex type element that could be defined in the *types* section of the WSDL document.

The Switch Types Complex Type Attribute (STCA) operator is defined as the operator that will switch attributes of the same type (strings, integers) within a single Complex Type element that could be defined in the *types* section of the WSDL document. The operation of the Switch Types Complex Type Attribute operator is similar to STCE. The only difference is in the switched parts—in case of STCA, attributes of the same type are switched instead of elements.

The Occurrence Types Complex Type Element (OTCE) operator adds or deletes an occurrence of an element in the complex type element in the *types* part of a WSDL document. This operator can be applied only if there are several elements in the complex type.

The Occurrence Types Complex Type Attribute (OTCA) operator adds or deletes an occurrence of an attribute in the complex type element that could be defined in the *types* part of a WSDL document. The operator can be applied only if there is more than one element in the complex type. Similar relation as in STCE and STCA operators occurs in OTCE and OTCA. The difference between these two lies in the part being added or deleted—in case of OTCE it is an element, whereas in OTCA—an attribute.

Other mutations operators were proposed by Solino and Vergilio in [15]:

- *changeServiceSigning* (similar to STCE)—changes the order of parameters of a service,
- *changeTypeonServiceSigning*—similar to *changeServiceSigning*, the difference is that this operator changes the type of parameters,
- *changeTypeofMessageElement*—modifies the type of an element present in a message by changing it into another type of an element also present in the same WSDL document.
- *changeInputOutput*—modifies the type of an input message by changing it by the type of an output message of the same transaction.

Other approach in mutation testing of Web Services and completely different mutation operators are presented in [25]. The authors propose to mutate SOAP message parameters and create combined mutants, using more than one mutation operator at a time. They also built a tool enabling the proposed type of testing.

4 WSMG

In Fig. 3 the general operation of WSMG tool is presented. A WSDL file or its URL needs to be supplied, mutation operators must be chosen and the output location needs to be set. WSMG generates mutated versions of WSDL files and their corresponding difference files. The difference files (with .diff extension) indicate what alterations have been made in each mutant in comparison with the original WSDL file. In the output location, pointed by the user, new folders named as the mutation operators are created and mutated WSDL and diff files are stored.

The WSMG Tool consists of the Mutant Generator, a WSDL Parser and a Graphical User Interface (GUI) (Fig. 4) and is implemented in Java. The Mutant Generator (MG) requires a WSDL Parser to read the WSDL document and to create mutants. There are several WSDL parsers available e.g. Apache Woden [33], EasyWSDL [34], Membrane SOA Model [35]. The Membrane SOA Model was chosen for the implementation of WSMG. It contains a Java API for WSDL and XML Schema and tools to compare and analyze WSDL and Schema documents. It allows to create, parse and manipulate a WSDL document and to compare two WSDL files. It is the only WSDL parser able to create a SOAP request. The last feature is very useful in the development of further steps of mutation testing (Fig. 2). Membrane uses WSDL 1.1, so the WSMG tool accepts only WSDL 1.1. The main screen of our tool is shown in Fig. 5.

Mutant Generator is responsible for:

- reading the WSDL from the file specified in GUI,
- transforming the WSDL into an object model using the WSDL Parser,
- generating mutants using mutation operators selected in the GUI checkboxes,
- saving mutants into files in directories named after mutation operators,

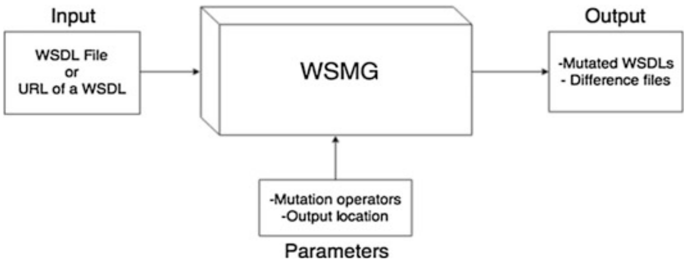
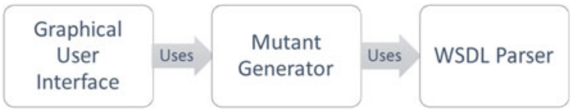
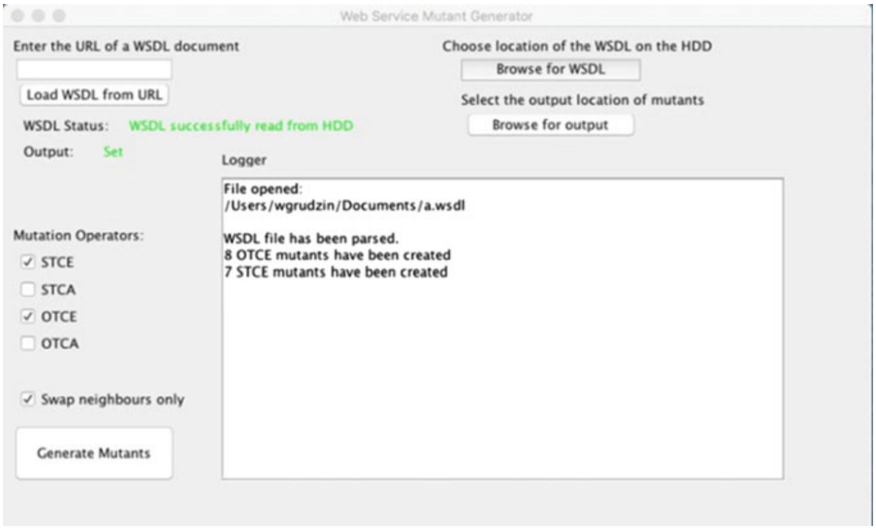


Fig. 3 Operation of WSMG

Fig. 4 Main parts of WSMG




Web Service Mutant Generator
Enter the URL of a WSDL document
Load WSDL from URL
Choose location of the WSDL on the HDD
Browse for WSDL
Select the output location of mutants
Browse for output
Logger
Mutation Operators:
Generate Mutants

WSDL Status: WSDL successfully read from HDD
Output: Set

- [✓] STCE
- [ ] STCA
- [✓] OTCE
- [ ] OTCA
- [✓] Swap neighbours only

```
File opened:
/Users/wgrudzin/Documents/a.wsdl

WSDL file has been parsed.
8 OTCE mutants have been created
7 STCE mutants have been created
```

Fig. 5 Main screen of WSMG

- creating files showing the differences between each mutant and the original WSDL,
- passing output information to the GUI logger.

As Membrane SOA is able to compare two WSDL files so a file containing differences between a particular mutant and the original WSDL structure is also generated.

4.1 Reduction of Mutants

Originally, in the STCE and STCA operators, all possible permutations of elements of the same types are used for the generation of new WSDL files. Such method, for n elements of the same type generates n! mutants of WSDL documents. Executing all these mutants will be time consuming so we propose to reduce the number of mutants and eliminate some permutations. We propose to **swap only the neighboring** elements. An element at position i is swapped with an element at position $(i + 1)$ or $(i - 1)$. Such approach, for a sequence of length 3, generates only three mutants instead of six ones in the Siblini and Mansour [14] approach. If all permutations are generated then different faults in WSDL documents that do not represent little mistakes made by programmers will be also introduced. The solution proposed by us uses hints given in [36] by DeMillo and others for the test data selection: "Programmers create programs that are close to be correct" and coupling effect "complex errors are coupled to simple ones". The **swapping algorithm** is given in Fig. 6.

In the tested WSDL document (presented in Fig. 7) this algorithm reduced the number of mutants from 23 to 4 and as it was designed on the basis of hints widely used for many years we expect that the efficiency of mutation testing will not be decreased.

```
Input: n - the upper boundary of the sequence, length of the generated
sequences
Output: A set of permutations of the sequence (1,2,…,n) referred to as
$F_n$, size of $F_n$ denoted as $|F_n|$

For n=1 there is only one element, so only $F_1$={1} is generated; $|F_1|$=1
For n>1 the algorithm has two steps:
    1. Take all sequences from $F_{n-1}$ and append n at the end
        ($|F_n|=|F_{n-1}|$)
    2. Take all sequences from $F_{n-1}$ whose last element is n-1 and ap-
       pend pair (n,n-1) at the end
        ($|F_n|=|F_{n-1}|+|F_{n-2}|$)
```

Fig. 6 Swapping algorithm

```
<wsdl:types>
 <xsd:schema attributeFormDefault="unqualified"
 elementFormDefault="qualified" targetNamespace
  ="http://thomas-bayer.com/blz/">
  <xsd:element name="getBank" type="tns:getBankType"></xsd:element>
  <xsd:element name="getBankResponse"
   type="tns:getBankResponseType">
  </xsd:element>
  <xsd:complexType name="getBankType">
   <xsd:sequence>
    <xsd:element name="blz" type="xsd:string"></xsd:element>
   </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="getBankResponseType">
   <xsd:sequence>
    <xsd:element name="details"
     type="tns:detailsType"></xsd:element>
   </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="detailsType">
   <xsd:sequence>
    <xsd:element minOccurs="0" name="bezeichnung" type="xsd:string">
    </xsd:element>
    <xsd:element minOccurs="0" name="bic"
      type="xsd:string"></xsd:element>
    <xsd:element minOccurs="0" name="ort"
      type="xsd:string"></xsd:element>
    <xsd:element minOccurs="0" name="plz"
      type="xsd:string"></xsd:element>
   </xsd:sequence>
  </xsd:complexType>
 </xsd:schema>
</wsdl:types>
```

Fig. 7 Types section of BLZ Web Service [37]

4.2 Example

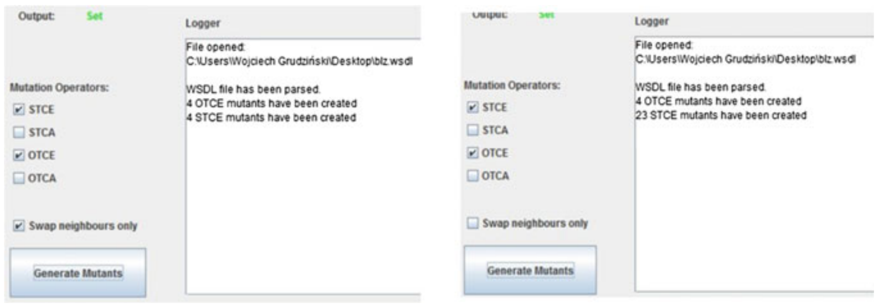
In Fig. 7 the types section of WSDL document describing BLZService [37] is given. This service was used in tests. There are three complex types, but only one (`detailsType`, lines 15–22) contains a sequence of multiple elements. The `detailsType` complex type has 4 elements, all of the same type `string`. While generating mutants using all possible permutations there should be $4! - 1 = 23$ mutants.

If the “Swap only with neighbours” mode is selected the number of mutants should be equal to the 4th element (diminished by one) of the Fibonacci sequence with initial elements equal to 1 and 2. The 3rd element is equal to $1 + 2 = 3$ and the 4th equals $2 + 3 = 5$. Hence, there are $24 - 1 = 23$ STCE mutants in case of generating all permutations and $5 - 1 = 4$ in case of “Swapping neighbours only” mode. The number of OTCE mutants is the same regardless of the mode of operation and should be equal to the number of elements in the complex type (four in this case).

The logger outputs are presented in Fig. 8 (using neighbours only mode on the left) and without neighbours only mode (on the right).

In Fig. 9 part of the STCE mutant is shown, lines correspond to lines 15–22 from the WSDL description of the BLZService given in Fig. 6.

Our tool WSMG generates also files with .diff extensions in which the differences between a particular mutant and the original WSDL file are listed. The .diff



Mutation Operators:
[✓] STCE
[ ] STCA
[✓] OTCE
[ ] OTCA
[✓] Swap neighbours only

```
File opened:
C:\Users\Wojciech Grudziński\Desktop\blz.wsdl

WSDL file has been parsed.
4 OTCE mutants have been created
4 STCE mutants have been created
```

Mutation Operators:
[✓] STCE
[ ] STCA
[✓] OTCE
[ ] OTCA
[ ] Swap neighbours only

```
File opened:
C:\Users\Wojciech Grudziński\Desktop\blz.wsdl

WSDL file has been parsed.
4 OTCE mutants have been created
23 STCE mutants have been created
```

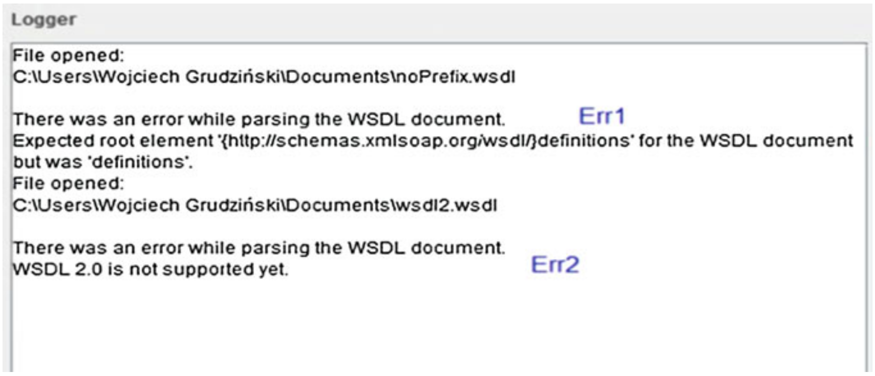
Fig. 8 Screens of WSMG—swapping and no swapping mode

```
<xsd:complexType name='detailsType'>
  <xsd:sequence minOccurs='1' maxOccurs='1'>
    <xsd:element name='ort' type='xsd:string' minOccurs='0' />
    <xsd:element name='bezeichnung' type='xsd:string' minOccurs='0' />
    <xsd:element name='plz' type='xsd:string' minOccurs='0' />
    <xsd:element name='bic' type='xsd:string' minOccurs='0' />
 </xsd:sequence>
</xsd:complexType>
```

Fig. 9 Part of STCE mutant

```
Definitions:
 Types:
  Schema http://thomas-bayer.com/blz/ has changed:
   ComplexType detailsType:
    Sequence:
     Position of element bezeichnung changed from 1 to 2.
     Position of element bic changed from 2 to 4.
     Position of element ort changed from 3 to 1.
     Position of element plz changed from 4 to 3.
```

Fig. 10 Example of. diff file associated with mutant from Fig. 8



```
File opened:
C:\Users\Wojciech Grudziński\Documents\noPrefix.wsdl

There was an error while parsing the WSDL document.
Expected root element '{http://schemas.xmlsoap.org/wsdl/}definitions' for the WSDL document
but was 'definitions'.
File opened:
C:\Users\Wojciech Grudziński\Documents\wsdl2.wsdl

There was an error while parsing the WSDL document.
WSDL 2.0 is not supported yet.
```

Fig. 11 Examples of errors

files are saved in the same directory as mutant files, they are named according to the rule: mutant_X.wsdl.diff, where X is replaced by a number of mutant. Every operator saves its mutants in the directory named the same as the operator. An exemplary .diff file is shown in Fig. 10.

In Fig. 11 two examples of errors detected while parsing WSDL files are given. In the first file the prefix was missing and the second file was containing WSDL 2.0 document which is not supported yet by the parser.

5 Conclusions

In this work the mutation testing of Web Services is described and WSMG—tool generating four mutation operators proposed by Siblini and Mansour [14] is presented. The tool uses a WSDL file describing a Web Service and generates mutants files for STCE, STCA, OTCE, OTCA (described in Sect. 3) mutation operators. We also proposed the “swapping” algorithm reducing significantly the number of generated mutants (Sect. 4.1).

Currently this tool is not able yet to fully perform the task of mutation testing of Web Services as it implements only the first step i.e. the generation of mutants. In the near future the SOAP messages generator and a SOAP client will be added to enable the whole process of testing. Also other mutations operators e.g. Special Types Element Nil (STEN), Switch Types Simple Type Element (STSE), Switch Types Simple Type Attribute (STSA), Switch Messages Part (SMP) and Switch Port Type Message (SPM) proposed in [14] or in [15] can be easily added. Currently WSMG operates on WSDL documents in version 1.1, in the near future WSDL documents in version 2.0 should also be accepted.

Tools for mutations testing of Web Services are unique. Only Solini and Vergilo implemented a tool WSeTT [15] generating mutants for theirs operators. Siblini and Mansour proposed mutation operators [14] but we were not able to find a tool implementing theirs operators, so we think it is worth to implement theirs mutation operators and do some experiments evaluating the usefulness of mutation testing of Web Services.

References

1. W3C Official Website. http://www.w3.org/
2. SOAP: Simple Object Access Protocol. http://www.w3.org/TR/soap/
3. XML—Extensible Markup Language. http://www.w3.org/XML/
4. WSDL. http://www.w3.org/TR/wsdl
5. UDDI. http://uddi.xml.org/
6. WSIL. http://www.ibm.com/developerworks/library/ws-wsilover/
7. Elberzhager, F., Rosbach, A., Eschbach, R., Münch, J.: Reducing test effort: a systematic mapping study on existing approaches. Inf. Softw. Technol. **54**(10), 1092–1106 (2012)
8. Jia, Y., Harman, M.: An analysis and survey of the development of mutation testing. IEEE Trans. on Soft. Eng. **37**(5), 649–678 (2011)
9. Canfora, G., Di Penta, M.: Service-oriented architectures testing: a survey. In: De Lucia, A., Ferrucci, F. (eds.) Software Engineering, pp. 78–105. Springer (2009)
10. Ladan, M.I.: Web services testing approaches: a survey and a classification. In: Zavoral, F., et al. (eds.) NDT 2010, Part II, CCIS 88, pp. 70–79. Springer, Berlin, Heidelberg (2010)
11. Bozkurt, M., Harman, M., Hassoun, Y.: Testing web services: a survey. Technical Report, King's College London (2010)
12. Metzger, A., Benbernou, S., Carro M., Driss, M., Kecskemeti, G., Kazhamiakin, R., Krytikos, K., Mocci, A., Di Nitto, A.E., Wetzstein, B., Silvestri, F.: Analytical quality assurance. In: Papazoglou, M., Pohl, K., Parkin, M., Metzger, A. (eds.) Service Research Challenges and Solutions for the Future Internet, pp. 209–270. Springer (2010)
13. Rusli, H.M., Puteg, M., Ibrahim, S., Tabatabaei, S.G.H.: A comparative evaluation of state-of-the-art web service composition testing approaches. In: International Workshop on Automation of Software Test, pp. 29–35 (2011)
14. Siblini, R., Mansour, N.: Testing web services. In: 3rd ACS/IEEE International Conference on Computer Systems and Applications (2005)
15. da Silva Solino, A.L., Vergilio, S.R.: Mutation based testing of web services. In: 10th Latin American Test Workshop, pp. 1–6 (2009)
16. Feudjio, A.G.V., Schieferdecker, I.: Availability Testing for Web Services. ISSN: 0085-7130 Telektronikk (2009)

17. Tsai, W.T., Paul, R., Song, W., Cao, Z.: An XML-based framework for web services testing. In: Proceedings of 7th IEEE International Symposium on High Assurance Systems Engineering, pp. 173–174 (2002)
18. Hanna, S., Munro, M.: An approach for WSDL-based automated robustness testing of web services. In: Information Systems Development Challenges in Practice, Theory, and Education, vol. 2, pp. 1093–1104, Springer (2008)
19. Bai, X., Dong, W.: WSDL-based automatic test case generation for web services testing. In: IEEE International Workshop on Service-Oriented System Engineering (SOSE'05), pp. 207–212 (2005)
20. Bluemke, I., Kurek, M., Purwin, M.: Tool for automatic testing of web services. In: Proceedings of the 2014 Federated Conference on Computer Science and Information Systems, Ganzha M., Maciaszek, L., Paprzycki, M. (eds.) Annals of Computer Science and Information Systems, vol. 3 (2014)
21. Sneed, H.M., Huang, S.: WSDLTest—a tool for testing web services. In: Eighth IEEE International Symposium on Web Site Evolution (WSE'06), pp. 14–21 (2006)
22. Canfora, G., Di Penta, M.: Testing services and service-centric systems: challenges and opportunities. IT Prof. **8**(2), 10–17 (2006)
23. Bucchiarone, A., Melgratti, H., Severoni, F.: Testing service composition. In: Proceedings of the 8th Argentine Symposium on Software Engineering, Argentina, ASSE 2007 (2007)
24. Zakaria, Z., Atan, R., Ghani, A.A., Sani, N.F.M.: Unit testing approaches for BPEL: a systematic review. In: Proceedings of the 2009 Asia-Pacific Software Engineering Conference, Penang, Malaysia, 1–3 Dec, pp. 316–322 (2009)
25. Chen, J., Li, Q., Mao, C., Towey, D., Zhan, Y., Wang, H.: A Web services vulnerability testing approach based on combinatorial mutation and SOAP message mutation. SOCA **8**, 1–13 (2014)
26. https://insights.sei.cmu.edu/devops/2015/04/devops-case-study-netflix-and-the-chaos-monkey.html
27. HP QuickTest Professional. http://www8.hp.com/us/en/software-solutions/unified-functional-testing-automation/
28. Parasoft. http://www.parasoft.com/soatest
29. SOAPSonar. http://www.crosschecknet.com/products/soapsonar.php
30. SoapUI. http://www.soapui.org/
31. Java Solution Architect: WSDL 2.0 VS WSDL 1.1. http://javasolutionarchitect.blogspot.com/2013/11/wsdl-20-vs-wsdl-11.html
32. WSDL elements. http://www.tutorialspoint.com/wsdl/wsdl_elements.htm
33. Woden. http://ws.apache.org/woden/
34. EasyWSDL Toolbox. http://easywsdl.ow2.org/
35. Membrane SOA Model. http://membrane-soa.org/soa-model/index.htm
36. DeMillo, R.A., Lipton, R.J., Sayward, F.G.: Hints on test data selection: help for the practicing programmer. Computer **11**(4), 34–41 (1978)
37. BLZService WSDL Document. http://www.thomas-bayer.com/axis2/services/BLZService?wsdl

Improving Measurement Certainty by Using Calibration to Find Systematic Measurement Error—A Case of Lines-of-Code Measure

Miroslaw Staron, Darko Durisic and Rakesh Rana

Abstract Base measures such as the number of lines-of-code are often used to make predictions about such phenomena as project effort, product quality or maintenance effort. However, quite often we rely on the measurement instruments where the exact algorithm for calculating the value of the measure is not known. The objective of our research is to explore how we can increase the certainty of base measures in software engineering. We conduct a benchmarking study where we use four measurement instruments for lines-of-code measurement with unknown certainty to measure five code bases. Our results show that we can adjust the measurement values by as much as 20 % knowing the systematic error of the tool. We conclude that calibrating the measurement instruments can significantly contribute to increased accuracy in measurement processes in software engineering. This will impact the accuracy of predictions (e.g. of effort in software projects) and therefore increase the cost-efficiency of software engineering processes.

1 Introduction

With the introduction of the measurement information model in the international ISO/IEC 15939 standard for measurement processes the discipline of software engineering evolved from discussing metrics in general to categorizing them into three categories—base measures, derived measures and indicators. The use of base measures is fundamental for the construction of derived measures and indicators. The base measures are also the types of measures which are collected directly and are a

M. Staron (✉) · R. Rana
Computer Science and Engineering, University of Gothenburg, Gothenburg, Sweden
e-mail: miroslaw.staron@gu.se

R. Rana
e-mail: rakesh.rana@gu.se

D. Durisic
Volvo Car Group, Gothenburg, Sweden
e-mail: darko.durisic@volvocars.com

L. Madeyski et al. (eds.), *Software Engineering: Challenges and Solutions*,
Advances in Intelligent Systems and Computing 504,
DOI 10.1007/978-3-319-43606-7_9

result of a measurement method. In many cases this measurement method is an automated algorithm (e.g. a script) which we can refer to as the *measurement instrument* which quantifies an attribute of interest into a number.

Since in software engineering we do not have reference measurement etalons as we do in other disciplines (e.g. kilogram or meter for physics), we often rely on arbitrary definitions of the base quantities. One of such quantities is the size of programs measured as the number of lines of code. Even though the number of lines of code of a given program is a deterministic and fully quantifiable number the result of applying different measurement instruments to obtain the number might differ. The difference can be caused by a number of factors, such as: (i) difference in implementation of the same measurement method, (ii) difference in the definition/design of the measurement method, or (iii) faults in the measurement instruments. Since we do not know the true value without the measurement procedure we need to find what the accuracy (certainty) of the measured value is. When using measurement instruments it is often not possible to explore the measurement methods in details by analyzing the implementation of the measurement method (in practice the measurement instruments are provided as compiled code), which means that the measurement engineer needs to either accept the fact that the uncertainty is unknown or estimate the uncertainty. The latter leads to more benefits in terms of improved quality of the measurement results and therefore it is of interest for our work, which addresses the following research question:

How to reduce the uncertainty of measurement results obtained from measurement instruments with an unknown measurement method?

This research question is addressed by constructing a benchmarking study where we use four different measurement instruments which quantify the number of physical lines of code of a program. We use these measurement instruments on five different open-source code bases in order to explore the deviations between the results obtained from these tools. We also use a simple program with a known number of physical lines of code in order to estimate the systematic error of the tools and then we use that number (converted to a percentage) to reduce the error of the measurement obtained in the first step.

In the study we use the lines-of-code measure because of its practicality and availability of measurement instruments (both open source and proprietary) with unknown measurement method [2]. Another practical advantage is the fact that the lines-of-code measure is usually calculated using automated software tools which provide deterministic results and redefine the notion of the error for this measure. In other engineering disciplines the measurement tools are prone to systematic errors (related to calibration of the tools) and random errors (related to the measurement process). When using automated tools for measuring such quantities as LOC the measurement process always results in the same value when measuring the same entity—therefore seemingly without the random error. Calibrating the tools is sometimes difficult, which results in not being able to distinguish the systematic errors for measurement from the random ones. The measure has been both widely used in software engineering for calculating the size of programs [8], software complexity [7] or to predict software size [4]. The wide use of the measure has resulted in multiple variants of the definition—e.g. non-commented lines of code, total lines of code, or source statements. Although the different variants of the measure should not be mixed, it is often the case that these definitions are not recognizable in measurement tools (also known as measurement instruments) and thus result in measurement errors.

Our results show that using this simple approach we can reduce the uncertainty to 1.85 from 20 % in some cases. We perceive this type of uncertainty reduction to be applicable to many other base measures (i.e. other than the lines-of-code measure used in this study) and have a potential to increase the validity of the prediction models using these base measures (e.g. COCOMO or defect density, [11]).

The paper is structured as follows. Section 2 outlines the most relevant work in the field. Section 3 presents the concept of the measurement error used in this paper. Section 4 presents the measure of lines of code. Section 5 presents the evaluation of the different definitions of errors on a set of open source programs. Section 6 provides recommendations on how to use the LOC measure in practice and Sect. 7 presents the conclusions.

2 Related Work

We review work in three areas—standardization in the area of measurement in software engineering, measurement theory (in general and its applications in software engineering) and the overview of critical articles of measurement in software engineering.

Lincke et al. [10] studied the deviations of results from measurement tools for sizing object-oriented designs. Their results showed significant deviations between tools. Their work shows how important the notion of systematic and random measurement errors are and can to a large extent be explained by the work presented in our paper.

The recent advances in the field of metrology have also started to arrive in the field of software engineering. Abran [1] used a combination of the ISO VIM standard to discuss the validity of the most common metrics used in software engineering e.g. software complexity or the function points measures. Although the developments are recent, they are based on the previously defined needs for more precise terminology and validation of software metrics, e.g. [1]. One of the major current trends identified by Abran is the search for etalons—elementary units—in software engineering.

The basic concepts of measurement theory for software engineering have been redefined by Briand et al. [5]—for example the concepts of relational systems, mappings and scales. Similar to the definition of relational systems one can define properties of software measures, which are a foundation for defining a general measurement instrument model. Such properties are defined by Briand et al. [6] based on the general properties of measures defined by Weyuker [19], Zuse [21] and Tian and Zelkowitz [15]. Although the property-based definition of measures addresses the problem of correct definition of software metrics, it does not address the issues of uncertainty of the measurement process in practice (i.e. the instantiation of the mapping between the empirical world and the relational systems).

Measurement theory has been used as a basis for the main international standard in measurement on common vocabulary in metrology—VIM [16]. The standard defines such concepts as measurement uncertainty, measurand and quantification. These definitions capture the meaning of the concepts from the measurement theory in engineering.

VIM standardizes the most important concepts which influence measurement processes, for example:

- Measuring instrument: device used for making measurements, alone or in conjunction with supplementary device(s)
- Instrumental measurement uncertainty: component of measurement uncertainty arising from the measuring instrument or measuring system in use, and obtained by its calibration
- Measuring system: set of one or more measuring instruments and often other devices, including any reagent and supply, assembled and adapted to give measured quantity values within specified intervals for quantities of specified kinds

VIM does standardize the vocabulary, but the international standard ISO/IEC 15939:2007 [12] (Systems and Software Engineering - Measurement Processes) focuses on the processes of measuring—data collection, processing and analysis. The ISO/IEC 15939 standard has an impact on the definition of metrics and measurement guidelines for the ISO/IEC 25000 series of standards. However, none of these two standards addresses the actual problem of measurement errors.

3 The Concept of Measurement Error

Measurement error is defined in measurement theory as the deviation between the real value of the measurand and the value obtained from the measurement process. It is derived from the concept of *measurement uncertainty* which is the dispersion of the values attributed to the measurand. The main definition is provided in ISO/IEC 17045 (General requirements for the competence of testing and calibration laboratories, [17]) and later on used in software engineering in ISO/IEC 25000 series of standards [18].

The measured quantity can be referred to as the *estimator* as the true value of the measurand always remains unknown because of the finite accuracy of the measurement instruments. In general the estimators (X) combine both the expected value of the measurement (M) plus the error (E)—see formula 1.

$$\hat{X} = \hat{M} + \hat{E} \tag{1}$$

The measurement error is combined of two types of errors—the systematic error (S) and the random error (R)—formula 2.

$$\hat{E} = \hat{S} + \hat{R} \tag{2}$$

The systematic error is usually caused by the miscalibration of the measurement instruments and is the same for all measurements taken by the instrument. The mean of the systematic error is expected to be non-zero, and causes skewness of the measurement results. In order to minimize the systematic errors the measurement instruments are calibrated—adjusted to show the correct results when measuring entities of known properties.

The random error, on the other hand, is different for each measurement taken. The mean value of the random measurement error is expected to be zero and it causes the distribution of the measurement results to be wider.

In general it seems easy to distinguish between these two types of errors, but in practice it might be very difficult. In Sect. 5 we show examples of this problem.

3.1 Impact of Measurement Error on Predictions—Standard Error of the Estimate

Software measures are usually used within various models designed for monitoring and forecasting [13, 14]. When using these measures within prediction models, the other major source of errors come from estimation error. Every estimation method involves an estimation error, which comes from the simple fact that the real quantity generally differs from its estimated quantity.

Taking an example of simple prediction model with single predictor (x) and predicted variable (y) have a linear relationship between them ($y \sim m * x$, where m = 2), two scenarios of prediction model can be shown as in Fig. 1.

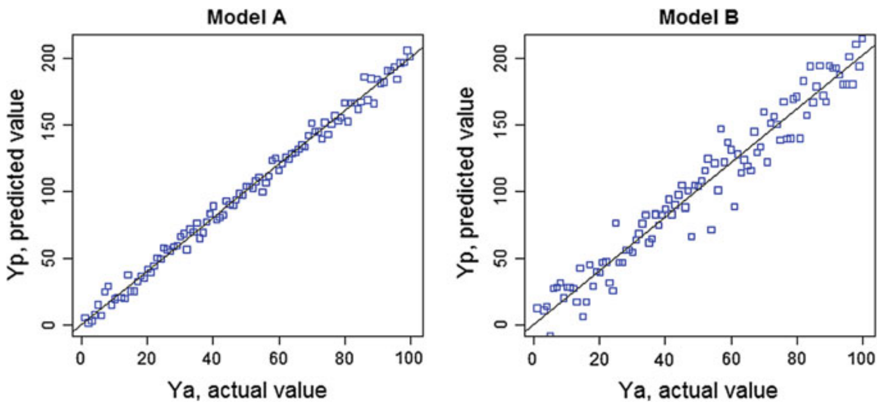


Fig. 1 Regression models with different accuracy of prediction

As evident from Fig. 1, the prediction model A seems more accurate then prediction model B, or in other terms the estimation error is expected to be lower for model A compared to mode B.

The standard error of the estimate (or SEE) is the measure of accuracy of predictions. For estimations using linear regression minimizes the sum of squared deviations of prediction (also referred to as Sum of Squared Errors, or SSE). Thus the standard error of estimate for method of least squares (linear or non-linear models) is equal to SSE, which can be calculated as:

$$\sigma_{est} = \sqrt{\frac{\Sigma(Y_a - Y_p)^2}{N}} \tag{3}$$

where σ_{est} is the standard error of estimate, Y_a is the actual value, Y_p is the predicted value, and N is the number of observations.

4 Lines of Code (LOC)

In this paper we use the measure of Lines of Code (LOC) as an example to illustrate the concept of measurement error. The measure has been used in practice since 1950 s and there is substantial body of research on it, [3]. It is also used as a input variable to many prediction models—e.g. the Constructive Cost Model (COCOMO) and its newer versions [4].

LOC measure is often also called SLOC (Source Lines of Code) as an acronym and has multiple variations, for example:

1. Physical (Source) Lines of Code—measure of all lines, including comments, but excluding blanks
2. Effective Lines of Code—measure of all lines, excluding: comments, blanks, standalone braces, parentheses.
3. Logical Lines of Code—measure of those lines which form code statements

The variations of the measure are used for specific purposes and can be regarded as measures of the same entity, but with different measurement methods according to the definitions included in ISO/IEC 15939 [12]. We can also observe that these measures are liable to systematic errors in different ways—for example the number of physical lines of code will include comments which do not add to the complexity of the algorithm, but may impact the effort needed to develop the program; the logical lines of code will naturally not be sensitive to the same type of error (i.e. comments).

5 Empirical Evaluation

In order to assess what the effects of different ways of calculating measurement error have on the results, we designed a benchmarking study of calculating LOC on a number of open source projects. For the calculations we chose a set of tools which calculate the effective lines of code and should be comparable. The goal of the evaluation was to evaluate the impact of the systematic and random measurement errors on the result of the measurement process from the perspective of a quality manager.

5.1 Design

For the purpose of the evaluation we randomly chose four measurement instruments:

- Unified CodeCount (UCC), Release 2011.10, http://sunset.usc.edu/research/CODECOUNT/
- Understand, http://www.scitools.com/download/
- Code Analyzer, Version 0.7.0, http://sourceforge.net/projects/codeanalyze-gpl/
- SLOCCount, Version 2.26, http://www.dwheeler.com/sloccount/

We also chose five different source code packages in order to capture variability in the programming styles:

- Linux Kernel, Release 3.13.6
- Mozilla Firefox, 27.0.1
- Open Office, V4.0.1 R1524958
- Android, V4.0.3 R1
- Chrome, V18.0.1025123

The size of the measured programs was more than a few hundred lines of code, which was important for the estimations—the size was too large for a person to count the number of lines of code manually and the estimates were needed.

The process of our benchmarking study was as follows. First we calculated the LOC for all software packages and calculated the median for each package. The median was chosen as it is the value that is in-between the middle data points (or the mid data point in case of odd number of data points) and therefore is can be the value that has been measured by one tool. Once we calculated the median we calculated the absolute error which is the difference between each measurement and the median. Then we calculated the relative error as a ratio between the absolute error and the median, which we use later on for comparison. Then we calculated the systematic error by using a pre-defined program as a measured entity for each measurement instrument. Then we used the measured systematic error to adjust the values of the LOC measurements from the five software packages and repeated the procedure with finding the median and calculating the absolute and the relative errors. We then compared the differences between these two calculations and discussed them.

5.2 Results

The results are presented in Table 1 and are predictably correct—all measurement tools provide different measurement results. We do not know the systematic and the random measurement error for these programs. Therefore it is not possible to distinguish the differences between these two types of errors.

Table 2 shows the results from calculating the absolute relative error using the median as the substitute for the true value of the measurement. We chose median instead of arithmetic mean as it has the property of being the middle value in the data set (if the number of elements is odd). This means that it is a value which exists in the data set whereas the arithmetic mean has the property of being in the middle, but most often it does not exist in reality.

The error values shown in Table 2 indicate that there is a large discrepancy in which tools present the lowest and the highest results—visible for the measurement results for the Android source code.

Table 3 presents the percentage—the relative measurement error for the same measurement results.

The results presented in Table 3 shows that the relative error could be as large as 18.58 % for the example data set (Android code measured using Understand). It is natural that this kind of error is unacceptable for any evaluation and that this kind of magnitude of the error could be caused by a systematic error combined with the total size of the software (i.e. low numbers combined with large systematic error will result in large percentage of relative error).

Table 1 Results from measuring five source code repositories with four different tools

Source code	UCC	Understand	Code analyzer	Universal CLC	Median
Linux Kernel	11 631 288	9 466 175	11 650 363	11 207 899	11 419 931
Mozilla Firefox	5 066 234	4 282 587	5 457 003	4 966 983	5 016 609
Open Office	4 855 090	4 431 717	5 231 869	4 742 033	4 798 562
Android	878 157	878 041	876 795	871 710	877 418
Chrome	5 502 768	4 460 016	6 263 157	5 453 027	5 419 742

Table 2 Absolute measurement error when using median as the estimator of the true value

Source code	UCC	Understand	Code analyzer	Universal CLC
Linux Kernel	211 695	−1 953 419	230 770	−211 695
Mozilla Firefox	49 626	−734 022	440 395	−49 626
Open Office	56 529	−366 845	433 308	−56 529
Android	739	623	−623	−5 708
Chrome	24 871	−1 017 882	785 260	−24 871

Table 3 Relative measurement error when using median as the estimator of the true value

Source code	UCC (%)	Understand (%)	Code analyzer (%)	Universal CLC (%)
Linux Kernel	1.85	17.11	2.02	1.85
Mozilla Firefox	0.99	14.63	8.78	0.99
Open Office	1.18	7.64	9.03	1.18
Android	0.08	0	0.07	0.65
Chrome	0.45	18.58	14.34	0.45

Table 4 Results from measuring 5 source code repositories with 5 different tools

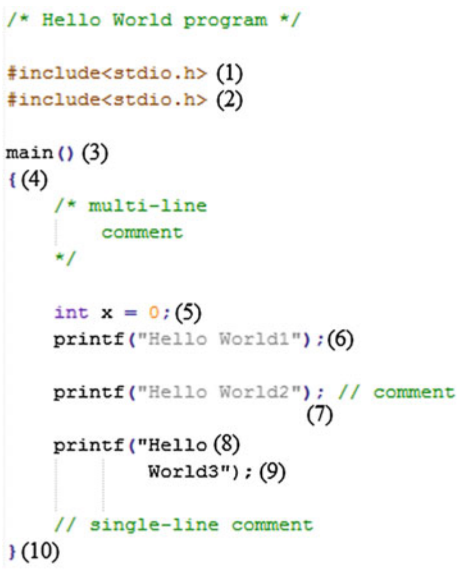
Measurement instrument	Measured LOC	Systematic error	Systematic error (relative) (%)
UCC	10	0	0
Understand	8	2	20
Code Analyzer	10	0	0
Universal CLC	9	1	10

Table 4 shows the results from calibrating the measurement tool on a small C++ program with 21 lines of code as shown in Fig. 2. The program has 10 lines of executable code and the code is written with extra white spaces (e.g. after { after the main procedure). Each line which is counted as a LOC has a number in bracket next to it (e.g. "#include <stdio.h>(1)").

The results shown in Table 4 show that several of the programs provide the results which count the lines of executable code as we intended—e.g. UCC and Code Analyzer. The tool Understand does not even count the curly brackets and therefore can be seen as the one with a negative systematic error—the number of LOC is 8 for that program.

One observation which we can make by comparing the relative error and the systematic error is the difference between these two. For example let's consider the tool Understand, which has a difference of 2 LOC in Table 4 with the calibration results. 2 LOC is 20 % of 10 LOC of the program used for the calibration. In Table 3 that tool shows a relative error in the range of 0.07–18.58 %. This means that knowing an estimate of a systematic error allows to reduce the random error from as much as 18.58–1.42 % (calculated as (20–18.58 %). If we apply the same approach to all measures in Table 3, we get the results as presented in Table 5.

We could see that the tools which have the largest systematic error still have the largest error in total (e.g. Understand). Now, that we know the systematic error and its direction (underestimation) we can take that into the account and change the values of the LOC measurement from Table 1 and redo the calculations which results in the following number of the relative error as presented in Table 6.



```
/* Hello World program */

#include<stdio.h> (1)
#include<stdio.h> (2)

main() (3)
{ (4)
    /* multi-line
        comment
    */

    int x = 0; (5)
    printf("Hello World1"); (6)

    printf("Hello World2"); // comment
                          (7)
    printf("Hello (8)
            World3"); (9)

    // single-line comment
} (10)
```

Fig. 2 Example program used for calibration of the tools

Table 5 Relative measurement error when using median as the estimator of the true value reduced by the systematic error from Table 4

Source code	UCC (%)	Understand (%)	Code analyzer (%)	Universal CLC (%)
Linux Kernel	1.85	2.89	2.02	8.15
Mozilla Firefox	0.99	5.37	8.78	9.01
Open Office	1.18	12.36	9.03	8.82
Android	0.08	19.93	0.07	9.35
Chrome	0.45	1.42	14.34	9.55

Table 6 Relative measurement error when using median as the estimator of the true value reduced by the systematic error from Table 4

Source code (%)	UCC (%)	Understand (%)	Code analyzer (%)	Universal CLC (%)
Linux Kernel	0.08	2.42	0.08	5.91
Mozilla Firefox	4.38	3.00	3.00	3.13
Open Office	7.06	1.80	0.15	0.15
Android	4.39	14.71	4.54	4.39
Chrome	4.31	6.93	8.91	4.31

The results should be compared with Table 3 as they are recalculated in the same way, given the new median (as the LOC measurement for Understand and Universal CLC are increased by 20 % and 10 % respectively). What the results show is that we can decrease the uncertainty (compared to Table 5) as we only have the relative error component in the measurement.

Naturally more studies are needed to use different programming styles when defining the programs used to calibrate the tools, but even this small example shows that the calibration has a significant impact on the measurement error and thus on the estimation formulas where the measure value is used (e.g. in the COCOMO estimation model).

5.3 Threats to Validity

The empirical validation of the results have a number of validity threats as any other empirical study. We use the framework of Wohlin et al. [20].

The main threat to the *external validity* is the fact that we only used open source code in the evaluation. Although this threat can indicate a potential bias, we understand that the coding style differs enough between the open source projects (e.g. code written by different consortia) in order to make claim which can be generalized to other types of programs. The study by Lincke et al. [10] showed the same types of deviations as we observed in our work (but using higher-level, object-oriented metric suite), which was an independent study. This increases the validity of our conclusions as it is in fact a triangulation of the approach. However, we believe that more evaluations on more base measures to explore if our conclusions are valid in more contexts. We also plan to expand this study in the future to allow to draw the calibration code sample from the code base and therefore have consistency between the calibration code and the code base.

The main threat to the *conclusion validity* is the size of the sample—i.e. the number of measurement instruments (code counting tools) and the number of open source programs. We used a simple calibration program in order to establish the baseline for the systematic error in the measurement. Since our main goal was to understand the scale of the measurement error in practice, the small sample size does not lower the validity of our conclusions.

The main threat to the *construct validity* stems from the assumption that there is a difference between systematic and random error in software engineering. Both concepts are established in the measurement theory and they are based on the assumption that there is some degree of non-determinism in the measurement. The non-determinism assumption can be debatable, though, in software engineering (due to automated measurement instruments). Given the fact that the tools used in our study produced a result we treated them as correct (i.e. free from bugs), but this does not have to hold for all measurement instruments. Having bugs in the measurement instruments contributes to the presence of random error and therefore we plan to investigate it more in the next steps. The goal of this study was to investigate whether this assumption is correct and the results of our study show that these concepts are present.

The main threat to the *internal validity* is the choice of measurement tools and programs. They were chosen randomly based on internet search given the criteria that they can calculate the same type of LOC measure. Since there were differences between the measurement tools we believe that the internal validity is not jeopardized; had this not been the case, i.e. there were no differences we would require to expand the study to more tools.

6 Recommendations

Measurement errors are present in almost every measurement taken, including measurements in software engineering. The notion of the errors in software engineering is similar to the same notions in other disciplines and we provide the following recommendations:

- **Calibrate the instrument using small programs**: Calibration is the most important part of measurement and provides the possibility to calculate the errors with the "true" value of the estimator known (which is not the case of measurement of larger entities). Small program allow to manually calculate the values which can be used for calibration.
- **For the calibration, use the same programming style as the code base**: Drawing the sample of the calibration code from the measured code base allows for consistency between the programming styles between the measured code and the calibration code—the same programmers usually write the code in the same way [9].

7 Conclusions

The goal of this paper was to study how the notion of measurement error can be defined for metrics in software engineering to enable correct measurement. In order to study this concept we chose one of the most used and the simplest metrics—Lines of Code. As the metric is designed to measure source code we could link the results of measuring a program to a number. This metric allowed us also to discuss the measurement error based on a small calibration program.

The results showed that both the systematic and random errors are present and that the key to distinguish between these is to use calibration of measurement tools. These results mean that the measurement processes can be significantly improved when using calibration. Instead of using the concepts of standard deviations and average to estimate the LOC in the programs, which are based on assumptions of normal distribution of the metric, calibration provides a better estimate of measurement error and thus a better estimate of the true value of LOC.

In the future we plan to expand our approach by taking into considerations the influence of the assumptions made in this work—one of these assumptions being the consistency in programming styles between the code base and the calibration code. To account for that we plan to draw the sample code for calibration from the measured code base and calculate the influence of certain programming constructs on the measurement. For example we can count the number of lines containing only the curly brackets,or the number of lines containing only white spaces to capture different programming styles.

Acknowledgments The authors would like to thank the doctoral students in the "Measurement in Software Engineering" course at the University of Gothenburg for the discussions on the topic of measurement error and measurement theory.

References

1. Abran, A.: Software Metrics and Software Metrology. Wiley (2010)
2. Albrecht, A.J., Gaffney, J.E.: Software function, source lines of code, and development effort prediction: a software science validation. IEEE Trans. Softw. Eng. **6**, 639–648 (1983)
3. Boehm, B.: Managing software productivity and reuse. Computer **32**(9), 111–113 (1999)
4. Boehm, B., Clark, B., Horowitz, E., Westland, C., Madachy, R., Selby, R.: Cost models for future software life cycle processes: Cocomo 2.0. Ann. Softw. Eng. **1**(1), 57–94 (1995)
5. Briand, L., El Emam, K., Morasca, S.: On the application of measurement theory in software engineering. Empir. Softw. Eng. **1**(1), 61–88 (1996)
6. Briand, L.C., Morasca, S., Basili, V.R.: Property-based software engineering measurement. IEEE Tran. Softw. Eng. **22**(1), 68–86 (1996)
7. Davis, J.S., LeBlanc, R.J.: A study of the applicability of complexity measures. IEEE Trans. Softw. Eng. **14**(9), 1366–1372 (1988)
8. Khoshgoftaar, T.M., Munson, J.C.: The lines of code metric as a predictor of program faults: a critical analysis. In: Fourteenth Annual International on Computer Software and Applications Conference, 1990. COMPSAC90. Proceedings, pp. 408–413. IEEE (1990)
9. Kuzniarz, L., Staron, M.: Inconsistencies in student designs. In: The Proceedings of The 2nd Workshop on Consistency Problems in UML-based Software Development, San Francisco, CA, pp. 9–18. Citeseer (2003)
10. Lincke, R., Lundberg, J., Löwe, W.: Comparing software metrics tools. In: Proceedings of the 2008 International Symposium on Software Testing and Analysis, pp. 131–142. ACM (2008)
11. Nagappan, N., Ball, T.: Use of relative code churn measures to predict system defect density. In: 27th International Conference on Software Engineering, 2005. ICSE 2005. Proceedings, pp. 284–292. IEEE (2005)
12. Organization, I.S., Commission, I.E.: Software and systems engineering, software measurement process. Technical report. ISO/IEC (2007)
13. Rana, R., Staron, M., Berger, C., Hansson, J., Nilsson, M., Torner, F.: Evaluating long-term predictive power of standard reliability growth models on automotive systems. In: 2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE), pp. 228–237. IEEE (2013)
14. Staron, M.: Critical role of measures in decision processes: managerial and technical measures in the context of large software development organizations. Inf. Softw. Technol. **54**(8), 887–899 (2012)
15. Tian, J., Zelkowitz, M.V.: A formal program complexity model and its application. J. Syst. Softw. **17**(3), 253–266 (1992)

16. International Bureau of Weights and Measures: International Vocabulary of Basic and General Terms in Metrology, 2nd edn. International Organization for Standardization, Genve, Switzerland (1993)
17. International Bureau of Weights and Measures: General requirements for the Competence of Testing and Calibration Laboratories, 1st edn. International Organization for Standardization, Genve, Switzerland (2005)
18. International Bureau of Weights and Measures: Systems and software engineering—Systems and software Quality Requirements and Evaluation (SQuaRE)—Guide to SQuaRE, 2nd edn. International Organization for Standardization, Genve, Switzerland (2014)
19. Weyuker, E.J.: Evaluating software complexity measures. IEEE Trans. Softw. Eng. **14**(9), 1357–1365 (1988)
20. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in Software Engineering. Springer (2012)
21. Zuse, H.: A Framework of Software Measurement. Walter de Gruyter, Berlin (1998)

Performance Comparison of CRM Systems Dedicated to Reporting Failures to IT Department

Hubert Zarzycki, Jacek M. Czerniak, Dawid Lakomski and Piotr Kardasz

Abstract This paper contains study to confirm advantage of database systems in the client-side architecture for certain applications. Research investigates whether the client-side architecture systems can be more efficient at the significant load as compared to server-side architectures systems. The experiments were carried out for two types of servers and clients for a variety of tasks and servers load. For the purpose of experiments a two-part database system for reporting failures to IT department has been developed in Visual C# language according to former design assumptions. Then it was compared with the same system developed using PHP in the server-side architecture. The study has proven the advantage of client-side architecture systems in case of servers under high load and confirmed importance of using such architecture for small and medium-sized systems. Additionally, it indicated the necessity to optimize queries in order to accelerate operation of both kinds of applications as well as the need to use optimum network setup of the server.

Keywords Database · MySQL · MsSQL · C# · Visual Studio.NET · PHP · ERP · CRM · Client-side · Server-side

H. Zarzycki
Department of Computer Science, Wroclaw School of Information Technology,
ul. Wejherowska 28, 54-239 Wrocław, Poland
e-mail: hzarzycki@horyzont.eu

J.M. Czerniak (✉) · D. Lakomski
AIRlab Artificial Intelligence and Robotics Laboratory,
Institute of Technology, Casimir the Great University in Bydgoszcz,
ul. Kopernika 1, 85-064 Bydgoszcz, Poland
e-mail: jczerniak@ukw.edu.pl

P. Kardasz
Cluster of Research, Development and Innovation,
ul. Pisudskiego 74, 50-020 Wrocław, Poland
e-mail: p.kardasz@klasterbri.pl

L. Madeyski et al. (eds.), *Software Engineering: Challenges and Solutions*,
Advances in Intelligent Systems and Computing 504,
DOI 10.1007/978-3-319-43606-7_10

1 Introduction

Database systems are now the basis for the activities of each medium or large company. Nowadays, where information is power and the efficiency of information processing often means the success or failure in the market, it is difficult to imagine efficient management [7, 12, 14, 17] without any support from the database computer systems whether in dealing with customers (CRM [4, 13, 25]), financial and accounting software, or the complex ERP systems [6, 9, 13, 15]. Over the years, these systems were created in various architectures and had various forms, which have a close relationship with the development of hardware, operating systems and the computer science as a field of science. These systems were developed starting from relatively simple from relatively simple applications written in text mode, usually in the architecture of client-side (fat client) through complex windowing applications [1, 4, 5, 8, 10] (first CRM and ERP systems) gradually passing on the architecture of server-side (slim client) until relatively new and quite controversial trend of web browser based applications [2, 10, 22]. Currently, complex database systems from a programmer point of view are created basically using the same key assumptions. Usually they are constructed using one of several popular database management systems such as MS SQL, MySQL, Oracle, PostgreSQL, etc. In the vast majority they have a modular design which allows one to meet full scalability requirement, i.e. allows relatively easily to adapt applications to their rapid expansion and to fulfill further customer needs [2, 8]. Most common public database systems operate primarily on server-side architecture. At the present time server-side type of architecture [1, 11, 24] is a dominant solution which forces out client-side type of solutions to the margins of the market. There are many reasons for this, but it seems that the most important are:

- performing complex calculations on server side. It spare user's machine computing power. This allows to adapt applications to a large number of users including those not equipped with the latest hardware,
- reducing overhead on the communications, and thus reduce the network load by eliminating a large number of queries,
- theoretically easy procedure to change the behavior of a particular function by modification of the procedure only on the server side. This eliminates the need for distribution of the new software version to users.

In contrast, the most important advantages of client-side may include:

- there is no need to maintain continuous communication between the client and the server. This is particularly important in the case of bandwidth intensive applications,
- client-side architecture should behave relatively well when the server is overloaded, However, the client computer must be suitably efficient,
- some operating systems and applications are unable to run on server-side architecture. Fat clients can deal with them because they have their own resources.

Despite these strengths of server-side type of solutions applications running in the client-side architecture have not been forced out of the market completely. They cover the market segment where the above-mentioned advantages of server-side architecture are mitigated or even eliminated completely. These are typically medium-sized applications, usually fulfilling auxiliary functions. Examples of such applications are systems reporting failures used to communicate users with the IT departments [11, 16, 21], both within a single organization and to report failures related to the ERP/CRM to the company responsible for the support. Such systems provide a good material to compare both types of architecture in typical applications and use scenarios and to analyze their advantages and disadvantages. In a further part of the paper two systems reporting failures made in two different architectures will be presented along with a comparative analysis of their performance in typical scenarios where performance is modified by server load conditions, task type, server power and client computer power.

2 Failure Reporting Systems

2.1 Failure Reporting System.NET

Failure Reporting System.NET is an application written in C# .NET [2, 3, 19] for the IT department of Drozapol—Profile S.A. company. It's built on the client-side architecture using MySQL database. The application consists of two closely cooperating parts (administration console and end-user application) and has a quasi-modular construction (adding a new module is conducted by adding a new form to the project and updating publication). The application has a number of modules to extend the functionality, among others, these are:

- search engine module,
- module to manually add failure message,
- allocation and management of tasks module (the Head)
- messages publication module,
- administration module.

The application for actual implementation had to fulfill many design assumptions [5]. These included, among others, ease of use and intuitive client application, password security for administration console, implementation of procedures to recognize end users, etc. For the experiments we used the administrative console version 1.42 and the 1.0 version of the client.

2.2 PHP Failure Reporting System

PHP Failure reporting system as the name suggests is written in PHP language [3, 7]. It has the server-side type of architecture and is an application using a Web browser. This system is logically divided into two parts: one used to report failures by users and other password-protected used for administration of failure reporting and user management. The system has an extensive search engine, the possibility delayed registration of failures, and the ability to export reports to PDF. In the specified above company, PHP Failure Reporting System was introduced as first, but executives decided on its modernization [23] because due to performance and other issues it was ill perceived among the users.

3 Purpose and Method of the Experiment

3.1 Aims of the Project

The aim of the experiment was to prove the thesis that the client-side architecture systems can be more efficient in specific applications at the significant load of server as compared to server-side architectures systems in a particular software group.

For the purposes of the experiment 5 basic factors were chosen:

- Architecture type with two levels: client (C#), server (PHP).
- Server Type with two levels: new hardware (S2) or old hardware (S1).
- Client Test Computer with two levels: new hardware (TC1) or old hardware (TC2).
- Server load with three levels: no load, normal load, over load
- Task Type with two levels: search or display.

Therefore, it was decided therefore to examine: the impact of server load on the running time of each application (depending on the speed of the server), and the second to examine the impact of configuration of the user's computer on the application action running time (depending on the server load). Both experiments are designed to simulate the work of both applications in ideal (servers without load) and actual conditions (servers running under load or overload) and thus allow the assessment of the suitability of both application, depending on the environment in which they work.

3.2 Test Platform

Preparing a suitable environment for testing experimentation is very important [20]. The authors using previous experiences [18, 21] proposed a test platform which consists of:

The test platform consists of:

- Server 1: PC with Intel®Core™i7-6700T Processor, 8 GB RAM, 500 GB disk and running Windows 8.1 Professional and software Xampp Lite which includes, among others, Apache, and MySQL database.
- Server 2: PC with Intel®Core™i7-6700K Processor, 8 GB of RAM, disk 2x1TB RAID 10 and running Windows 8.1 Professional version and software Xampp Lite which includes, among others, Apache, and MySQL database, as well as software to man-age the server echoVNC.
- Test computer 1: PC with Intel®Core™i5-6402P Processor, 6 GB of RAM, 500 GB disk and running Windows 10 Professional with administration software.
- Test computer 2: PC with Intel®Core™i5-6500T Processor, 4 GB of RAM, 320 GB disk and running Windows 10 Professional and a typical user software (ERP system client application, user applications, company applications, etc.).
- PHP Failure Reporting System version 0.3 written in PHP for the Department of Computer Science of DROZAPOL—PROFILE SA company.
- Failure Reporting System II version 1.42 written in C# based on .NET technology created in Department of Computer Science of DROZAPOL—PROFILE SA company.
- Everest by Lavalys version Corporate.
- Consume by Microsoft.

3.3 The Experiment

Experiment was carried out in two stages. The first was to compare two studied solutions on two different servers in three states: no load, normal load and overload. In the state of no-load server doesn't run any application; load state is running the Everest Stability Test module (CPU load, RAM memory, hard disk, cache, and chipset motherboard) and overload, additional CPU load with program Consume (option "-cpu-time"). Both applications running on Test Computer 1. The second stage uses only the Server machine 2 and workstation labeled Test Computer 2. The purpose of this stage was to check how similar tasks will perform on regular computer and equipment of older generation. For the tests a working copy of the actual data from the database of failure reporting system were used. The test procedure consisted of measuring the time to display:

(a) search results of the entire database (3257 records)
(b) "daily" reports (22 records)

Stage I

Both parts of table Table 1 presents the results of measurements performed on Server 1 and Server 2 while performing search tasks on the database. Both servers were in the no-load state.

Table 1 Results of the experiment, state no-load during the search

Server 1 in no-load state						
Search time in [s]						
Measurement number	1	2	3	4	5	Average
PHP	6,367	6,272	7,196	6,194	6,379	6,482
C#	2,343	2,59	2,453	2,343	2,451	2,436
Server 2 in no-load state						
Search time in [s]						
Measurement number	1	2	3	4	5	Average
PHP	6,432	6,601	6,96	6,424	7,229	6,729
C#	2,328	2,296	2,281	2,315	2,343	2,313

Table 2 Results of experiment, state no-load during operations on selected records

Server 1 in no-load state						
Time to display the records in the [s]						
Measurement number	1	2	3	4	5	Average
PHP	0,113	0,119	0,103	0,101	0,1	0,107
C#	0,31	0,21	0,31	0,315	0,315	0,29
Server 2 in no-load state						
Time to display the records in the [s]						
Measurement number	1	2	3	4	5	Average
PHP	0,107	0,106	0,108	0,105	0,109	0,107
C#	0,031	0,031	0,031	0,032	0,031	0,031

Table 2 presents the results of measurements performed on Server 1 and Server 2 made during the execution of tasks to display selected records from the database. Both servers were in the no-load state.

Table 3 presents the results of measurements performed on Server 1 and Server 2 while performing search tasks on the database. Both servers were in the normal load state.

The results presented in Table 4 show the measurements performed on Server 1 and Server 2 made during the execution of tasks to display selected records from the database. Both servers were in the normal load state.

The data in the tables signed Table 5 and Table 6 relate to the measurement of execution time, respectively, search for information in the database and display records. Tasks were performed on the test machines Server 1 and Server 2 operating in an overload condition.

Table 3 Results of the experiment, state normal load during the search

Server 1 in normal load state

Search time in [s]

Measurement number	1	2	3	4	5	Average
PHP	10,71	8,082	7,531	6,917	9,43	8,535
C#	2,487	2,515	2,68	2,95	2,5	2,626

Server 2 in normal load state

Search time in [s]

Measurement number	1	2	3	4	5	Average
PHP	8,549	6,423	7,123	5,498	5,984	6,715
C#	2,515	2,5	2,437	2,484	2,45	2,477

Table 4 Results of experiment, state normal load during operations on selected records

Server 1 in normal load state

Time to display the records in the [s]

Measurement number	1	2	3	4	5	Average
PHP	3,04	1,069	1,074	2,095	1,019	1,659
C#	0,171	0,171	0,171	0,187	1,562	0,452

Server 2 in normal load state

Time to display the records in the [s]

Measurement number	1	2	3	4	5	Average
PHP	0,995	0,642	1,054	0,733	0,613	0,807
C#	0,171	0,187	0,328	0,185	0,171	0,208

Table 5 Results of the experiment, state overload during the search

Server 1 in overload state

Search time in [s]

Measurement number	1	2	3	4	5	Average
PHP	25,25	27,62	29,42	23,42	19,1	24,963
C#	3,25	3,32	3,59	2,984	3,965	2,822

Server 2 in overload state

Search time in [s]

Measurement number	1	2	3	4	5	Average
PHP	23,66	26,84	25,87	26,87	22,12	25,073
C#	3,6	3,862	3,421	3,451	3,421	3,551

Table 6 Results of experiment, state overload during operations on selected records

Server 1 in overload state

Time to display the records in the [s]

Measurement number	1	2	3	4	5	Average
PHP	15,11	16,12	14,99	17,1	13,64	15,391
C#	0,859	0,453	0,515	0,84	0,671	0,668

Server 2 in overload state

Time to display the records in the [s]

Measurement number	1	2	3	4	5	Average
PHP	11,25	11,87	10,96	14,85	12,85	12,358
C#	1,5	0,852	1,15	1,2	1,31	1,202

Stage II

For a full spectrum of results both applications were examined on an older generation computer (Test computer No. 2). Measurements were made for the three load states for the Server No. 2 and overload state for the Server No. 1. All the other conditions were the same (search and display selected records). The results of these experiments are given in the last three tables (Tables 7, 8, 9 and 10).

Table 7 Results of the experiment, state no-load during on an older generation computer

Server 2 in no-load state

Search time in [s]

Measurement number	1	2	3	4	5	Average
PHP	12,67	11,73	13,68	13,42	14,5	13,201
C#	14,99	13,57	13,61	13,53	13,58	13,855

Server 2 in no-load state

Time to display the records in [s]

Measurement number	1	2	3	4	5	Average
PHP	0,16	0,162	0,167	0,165	0,17	0,165
C#	2,42	2,4	2,41	2,34	2,34	2,382

Table 8 Results of the experiment, state normal during operations on an older generation computer

Server 2 in normal load state

Search time in [s]

Measurement number	1	2	3	4	5	Average
PHP	13,14	12,85	13,55	13,84	13,57	13,389
C#	11,46	13,64	14,14	14,58	13,23	13,409

Server 2 in normal load state

Time to display the records in [s]

Measurement number	1	2	3	4	5	Average
PHP	0,786	0,713	0,866	0,841	0,71	0,783
C#	2,78	3,31	3,14	2,866	2,473	2,914

Table 9 Results of experiment, state overload during operations on an older generation computer

Server 2 in overload state

Search time in [s]

Measurement number	1	2	3	4	5	Average
PHP	19,74	21,43	16,73	18,43	19,67	19,201
C#	14,74	14,12	14,45	14,27	14,82	14,481

Server 2 in overload state

Time to display the records in [s]

Measurement number	1	2	3	4	5	Average
PHP	8,633	7,543	6,922	10,12	8,42	8,328
C#	3,625	3,737	3,652	3,781	3,412	3,641

Table 10 Results of experiment, state overload during operations on Server 1 and an older generation computer

Server 1 in overload state

Search time in [s]

Measurement number	1	2	3	4	5	Average
PHP	21,07	22,12	19,03	16,27	16,98	19,1
C#	13,4	12,14	15,16	12,87	14,31	13,58

Server 1 in overload state

Time to display the records in [s]

Measurement number	1	2	3	4	5	Average
PHP	11,6	10,24	9,47	11,32	8,94	10,31
C#	2,08	1,83	1,64	2,47	1,75	1,95

4 Analysis of the Results

Preliminary analysis the numbers representing the input parameters indicates, that the results change significantly with the load on the server. As for job type, i.e. search time and display time, one can see a large differences especially for server in no-load state. In fact performing one task can be several times faster than performing the other. The power of the client computer is particularly important for applications written in C#. Without fast client computer application in C# does not perform too well. The power of the server and the server, and the second affects relatively in the slightest on the final results changing them by an average of less than thirty percent. From the point of view of individual measurements there is no statistically significant difference between the performance of the two applications.

Analyzing the results of measurements in Tables 7–10, illustrated in the charts, is easy to see a pattern: the cooperation between the two servers with a relatively fast machine time to generate results for C# based system were lower than those provided with the system written in PHP. This happened both in the case of displaying "daily records" and carrying out more complicated question whose task was to display several thousand of records. However, while the measurements on the no-load machine vary from tenths of seconds as a result of the operation displaying selected records (and thus virtually imperceptible to users), in the case of generating search results these times were, in the case of a PHP program, almost three times higher while the server was busy with program Everest. The times generated by a program written in PHP tested on both servers, when viewing selected records, were twice higher that C# program. An interesting curiosity is the second result of displaying selected records, where the times to generate the results did not differ too much from the results of the previous measurement—no-load state of the server. This may be related to the complexity of the question, the quantity of the necessary comparisons to perform and thus the degree of load on the MySQL database server (which, incidentally, is controlled by a system service, and therefore high priority process). During the last measurement made on the overloaded server results are radically different: while the system written in C# does not show a significant increase in time to display the results in comparison to previous cases, the system implemented on the server side generates a very large measurement time to display the results: from fifteen times longer than in the case of selected records for almost twenty five higher in the case of database search. In practice, this prevents the efficient use of the system (Fig. 1).

The reason for this phenomenon is easy to explain: in the case of system written in C# program performs all calculations associated with a graphical display of records on a computer, database takes only small portions of the data used to populate eg. the components and of course to generate results. In contrast, a system written in PHP is generated entirely on the server side, including both the calculations and generating the query results Fig. 1 (that is, there is a de facto collaboration of two Apache and MySQL servers) and displaying already finished internet page is conducted by user's browser. High load of server will therefore delay the job at the expense of other server processes. In confirmation of these results, one can see that for the old type

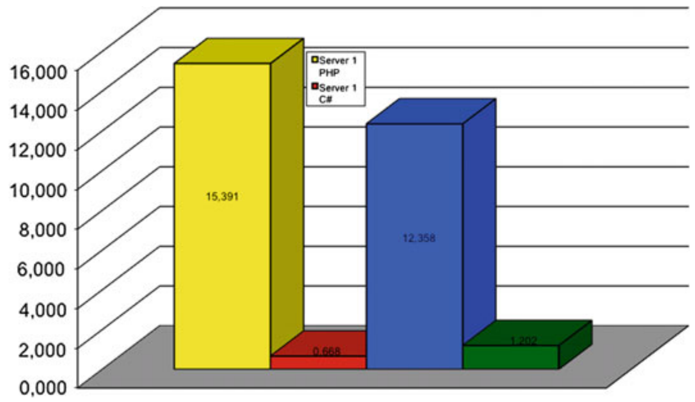


Fig. 1 Average times in [s] of operations on selected records, servers in overload state

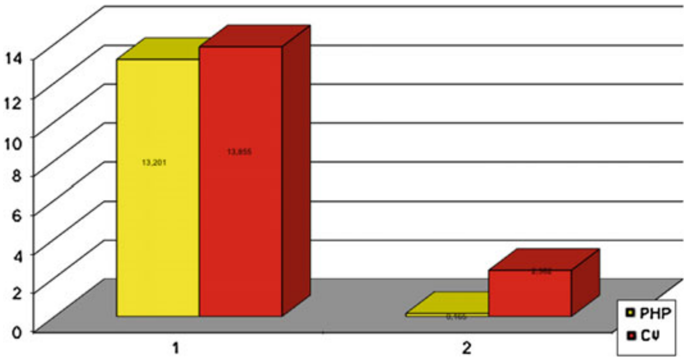


Fig. 2 Average times in [s] of operations on selected records on an older generation computer, servers in no-load state

of computer and overloaded servers 1 and 2 give there is only a slight advantage of application in C# (Fig. 2).

As for the results of stage II—the conclusions are not so obvious. Less powerful machine equipped with a previous generation processor cooperating with servers was used. Results obtained for the display of selected records—see Fig. 2—indicate that the PHP program needed less time than the program created in C# (reached an average of 2 s while PHP had less than 0.2 s). Similar results were obtained, when the server was in loaded state. Only with overload server the times for C# were lower, but not in such a spectacular way as in the previous studies. It can be concluded that for less powerful machines on the user side, the advantage in the form of shorter calculation runtime is largely counteracted. The reason in this case is the design of .NET platform which includes C# language and is a base of executed programs. .NET needs large amount of memory and requires a large amount of other resources. When used with older generation effects are not impressive (Fig. 3).

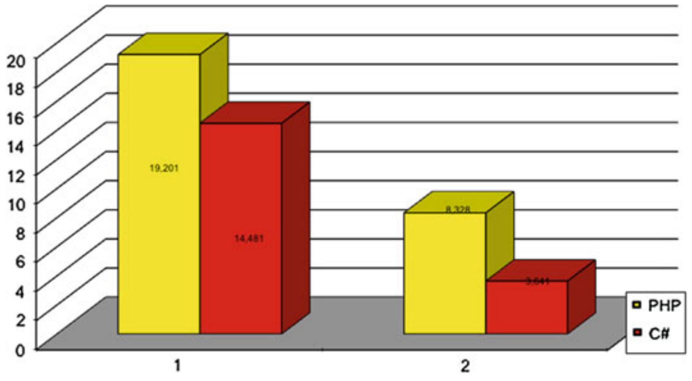


Fig. 3 Average times in [s] of operations on selected records on an older generation computer, servers in overload state

As an interesting fact one can provide the result of additional testing that the authors of this work carried out at the end of the experiment: to the overloaded server with programs Everest and Consume another process Consume was added with "-kernel" parameter (see Fig. 3). This parameter additionally attach more work to kernel. The result was a complete stop of GUI server, which resulted in the stopping of the Apache server application and disabled the display of pages. A big surprise was that the .NET Failure Report System continued its work on a level comparable to that observed when working with overloaded server. This can be explained by lesser resistance of Apache web server to the unstable work of the MySQL server.

5 Conclusions

In summary, one can conclude that, client-side database systems in the case of faster machines are much more efficient than the systems based on server-side type of architecture. However this is not the case on older generation machines. Server-side database systems, if they have access to efficient data centers are more optimal solution than dedicated user machines with limited capabilities. When working under normal load and processing small amounts of data, both types of systems do not differ significantly in terms of performance. However, in case of large portions of data and a relatively less powerful server user-side systems presents a much greater efficiency. In addition, one have to mention that the server-side system require administrator attention to provide robust server security, because these are the systems available through the website which is associated with a higher risk of attacks. It is obvious server-side systems are much less resistant to overload of the server, Denial of Service (DoS) attacks [3–5], or delay on the links. User-side systems are much more economical, they do not need expensive equipment to operate: with small databases it is enough to use regular PC and a free database such as MySQL. Summarizing experimental part and analyzing above observations one can come to the conclusion that there is no universal platform on which to build data processing systems. Both systems have their advantages and disadvantages which manifest themselves in specific situations and specific environments. However, based on literature and experimental research one can suggest to large organizations which can afford their own large server farms (or dedicated servers) and are able to simultaneously handle several hundreds or thousands of queries and also need a system capable of working on less powerful workstation that systems based on web page are best solution for them. The experience of many companies confirm that these are relatively cheap solutions and available from any place without specialized software. An alternative to expensive hardware solutions in this case is renting space on a server or the recently very popular "cloud computing." In this case, however, reckon with the possible overload of these machines, a delay on links and other such issues which may adversely affect the viability of the system.

Systems written in languages such as C# or Java are a quite good solution for companies which have a relatively good machines designed to work with ordinary users and haven't got extensive data center infrastructure. However, they have one or two servers with less impressive hardware specifications, which run multiple services. Statistics published in the country, and for the small and medium enterprises (SME) shows that most companies in this sector does not invest large sums in the development of IT and server products. The advantages of these solutions are very low delays, ease of implementation and use, and in the case of simpler systems their cost is relatively small and is (depending on the number of licenses) in the range from a few hundred up to several thousand PLN.

References

1. Adamczyk, J.: The CRM in a classic and web context (in Polish) (2009)
2. Angryk, R.A., Czerniak, J.: Heuristic algorithm for interpretation of multi-valued attributes in similarity-based fuzzy relational databases. Int. J. Approx. Reason. **51**(8), 895–911 (2010)
3. Apiecionek, L., Czerniak, J.M.: QoS solution for network resource protection. In: Informatics 2013: Proceedings of the Twelfth International Conference on Informatics, pp. 73–76 (2013)
4. Apiecionek, L., Czerniak, J.M., Dobrosielski, W.T.: Quality of services method as a DDoS protection tool. In: Intelligent Systems'2014, Tools, Architectures, Systems, Applications, vol. 2, pp. 323, 225–234 (2015)
5. Apiecionek, L., Czerniak, J.M., Zarzycki, H.: Protection tool for distributed denial of services attack. In: Beyond Databases, Architectures and Structures, BDAS, 2014, vol. 424, 405–414 (2014)
6. Bojar, W., Drelichowski, L., Lewandowski, R., Oszuscik, G., Zarzycki, H.: A comparative analysis of polish economic development across the eu countries with the use of olap tools during 1994–2009. In: Studies and Proceedings of Polish Association for Knowledge Management, vol. 59, pp. 124–147 (2012)
7. Converse, T., Park, J., Morgan, C.: PHP5 i MySql Bible. Helion, Gliwice (2005)
8. Czerniak, J., Zarzycki, H.: Application of rough sets in the presumptive diagnosis of urinary system diseases. Artif. Intell. Secur. Comput. Syst. **752**, 41–51 (2003)
9. Czerniak, J.: Evolutionary approach to data discretization for rough sets theory. Fundamenta Informaticae **92**(1–2), 43–61 (2009)

10. Czerniak, J.M., Apiecionek, L., Zarzycki, H.: Application of ordered fuzzy numbers in a new ofnant algorithm based on ant colony optimization. In: Beyond Databases, Architectures and Structures, BDAS, 2014, vol. (424), pp. 259–270 (2014)
11. Czerniak, J.M., Dobrosielski, W., Zarzycki, H., Apiecionek, L.: A proposal of the new owlant method for determining the distance between terms in ontology. In: Intelligent Systems'2014, Tools, Architectures, Systems, Applications, vol. 2, pp. 323, 235–246 (2015)
12. Czerniak, J., Dobrosielski, W., Apiecionek, L.: Representation of a trend in OFN during fuzzy observance of the water level from the crisis control center. In: Proceedings of the Federated Conference on Computer Science and Information Systems, IEEE Digital Library, ACSIS, vol. 5, pp. 443–447 (2015)
13. Czerniak, J., Macko, M., Ewald, D.: The cutmag as a new hybrid method for multi-edge grinder design optimization. Adv. Intell. Syst. Comput. **401**, 327–337 (2016)
14. Czerniak, J., Smigielski, G., Ewald, D., Paprzycki, M.: New proposed implementation of ABC method to optimization of water capsule flight. In: Proceedings of the Federated Conference on Computer Science and Information Systems, IEEE Digital Library, ACSIS, vol. 5, pp. 489–493 (2015)
15. Dejnaka, A.: CRM. Zarzadzanie relacjami z klientami, Helion, Gliwice (2002)
16. Ewald, D., Czerniak, J.M., Zarzycki, H.: Approach to solve a criteria problem of the ABC algorithm used to the WBDP multicriteria optimization. In: Intelligent Systems'2014, Mathematical Foundations, Theory, Analyses, vol. 1, pp. 322, 129–137 (2015)
17. Kozik, R., Choraś, M., Flizikowski, A., Theocharidou, M., Rosato, V., Rome, E.: Advanced services for critical infrastructures protection. J. Ambient Intell. Hum. Comput. **6**(6), 783–795 (2015). http://dx.doi.org/10.1007/s12652-015-0283-x
18. Owedyk, J., Mathia, Z., Zarzycki, H.: Optimization algorithm supported on minimizing the Kullback-Leibler information divergence in some dynamical systems. Wroclaw Sch. Technol. Sci. Bull. **5**, 12–19 (2015)
19. Solis, D.: Ilustrated C#. Apress (2008)
20. Swierczewski, L.: Intel manycore testing lab—hardware and software environment focused on didactic of development and efficiency testing in software parallelin. Wroclaw Sch. Technol. Sci. Bull. **3**, 32–36 (2013)
21. Walser, K.: Auswirkungen des CRM auf die IT-integration. Eul Verlag (2006)
22. Yuan, S., Chang, W.: Mixed-initiative synthesized learning approach for web-based CRM. Expert Syst. Appl. **20**(2), 187–200 (2001)
23. Zakrzewski, P.: How to prepare the implementation of CRM (in Polish). Magazyn Modern Mark. **7**, 38–42 (2001)
24. Zarzycki, H.: Enterprise resource planning systems application and implementation on the example of simple.erp software package. Adv. Inf. Syst. Manage. AITM **205**, 281–291 (2011)
25. Zarzycki, H., Fronczak, E.: A practical approach to computer system architecture for an agro-food industry information center. PSZW **33**, 248–255 (2010)

Part III
Software Process Improvement

Software Engineering Needs Agile Experimentation: A New Practice and Supporting Tool

Lech Madeyski and Marcin Kawalerowicz

Abstract This article proposes a novel software engineering practice called Agile Experimentation. It aims mostly small experiments in a business driven software engineering environment where a developer is a scarce resource and the impact of the experimentation on the return-of-investment driven software project needs to be minimal. In such environment the tools used for the sake of research need to have virtually no negative impact on the developers, but simultaneously those tools need to collect high quality data to perform sound enough quantitative analyses. In order to fulfill those requirements, and to support the Agile Experimentation practice, we co-developed a tool called NActivitySensor that gathers the data about the developers activities in a widely used Integrated Development Environment—Visual Studio. The proposed Agile Experimentation practice and the developed tool complement each other to support lightweight experimentation in real-world software development settings.

Keywords Agile experimentation · Empirical software engineering · Experimentation in software engineering · NActivitySensor

1 Introduction

Performing experiments in industrial software development environment is difficult and quite expensive [8]. It is usually hard to replicate the software engineering experiments [1]. The same problem is magnified in Software Engineering experiments

L. Madeyski
Faculty of Computer Science and Management, Wroclaw University of Science and Technology, Wyb.Wyspianskiego 27, 50-370 Wrocław, Poland
e-mail: Lech.Madeyski@pwr.edu.pl

M. Kawalerowicz (✉)
Faculty of Electrical Engineering, Automatic Control and Informatics, Opole University of Technology, ul. Sosnkowskiego 31, 45-272 Opole, Poland
e-mail: m.kawalerowicz@po.opole.pl

L. Madeyski et al. (eds.), *Software Engineering: Challenges and Solutions*, Advances in Intelligent Systems and Computing 504,
DOI 10.1007/978-3-319-43606-7_11

conducted in industrial environment. There is usually no budget for conducting the experiment twice, with a classic method (used to date) as well as using the new approach (the one being evaluated). It takes a rich country and research institute to perform controlled replication experiment in real software project with professional developers [23]. It seems like in last years the software engineering scientists realized that there is a need for experimentation. In fact, nowadays it is more and more difficult to publish research papers that do not include an empirical evaluation of new methods, practices, technologies or tools.

While we notice the research activities concerning experimentation in software engineering of the large multinational corporations like IBM (http://research.ibm.com/—213 publications in "Programming Languages and Software Engineering" area), Microsoft (http://research.microsoft.com/–203 research projects in "Software development, programming principles, tools, and languages" area) or Google (http://research.google.com/—177 publications in "Software Systems" area),[1] we do see the lack of research motivation in smaller companies all around the world. Running real world software engineering project is a business with many interested parties, the most important of which is the customer. Unfortunately, the main priority for a customer is the return of investment (ROI), not the experimentation with a new software engineering method. Despite it could be beneficial in later projects. Customers fear that it will cost too much and there will be too much hassle while performing the experiments. For example developers will be pulled away from their actual job and software development tasks they should focus on. The developers themselves fear they will have more responsibilities in they day-to-day work. To overcome such difficulties and to fill the existing gap with regard to lack of software engineering experimentation practices and supporting tools, we call for something we denominated Agile Experimentation (more after simple definition from Merriam-Webster's Learner's Dictionary: "quick, smart, and clever" than after Agile Software Development). It is a way to perform empirical studies and research in real-world software projects with no or minimal impact on people involved in those projects and their schedules. But in the same time what we propose should give researchers the way to perform effective experimentation to reach reliable enough conclusions on a subject being investigated (e.g., the impact of a new practice, method or tool).

In this paper, we start from our motivation, presented in Sect. 2, on which we build our agile experimentation "manifesto" formulated in Sect. 3, where the idea is thoroughly explained. Then we focus on a tool that we co-developed to aid the agile experimentation. It is called NActivitySensor after its predecessor called Activity Sensor [18–20] that was developed under the supervision of one of the authors of this paper. The new tool was built for the Microsoft .NET Framework development, while its predecessor was built for Java. We describe the new tool in detail in Sect. 4 and in Appendix A. Discussion and future work can be found in Sect. 5.

[1]The numbers gathered in May 2016.

2 Motivation

The reasons given by scientists and professionals for not experimenting in software engineering are greatly outlined in the article "Should computer scientists experiment more?" by Tichy [22]. Among them are:

- Traditional scientific method is not applicable.
- The current level of experimentation is good enough.
- Experiments cost too much.
- Demonstration till suffice.
- There is too much noise in the way.
- Progress will slow.
- Technology changes too fast.
- You will never get published.

Tichy fights those excuses one by one in his paper with argumentation that will most likely resonate with average scientist but not a business person. It is because the researcher is most likely to aim at the scientific excellence and businessman is most likely to be interested in ROI. The argument that "experiments cost too much" is the hardest to fight in a ROI-oriented environment such as industrial software development. It is a fact that performing an experiment costs. It might be beneficial in a long term, but in terms of immediate ROI, it is always burdened with an additional cost. It is the reason why so few smaller companies are willing to take the additional experimentation costs into consideration while racing for the customer on highly competitive market. For the same reason there are so few industrial professionals available for experimentation. They are considered to be valuable resource to waste their time for additional tasks like taking part in experiments. The professionals themselves seem to be rather ambivalent to the idea of taking part in experiments. From one point of view, they know that the results are potentially beneficial but from the other, they fear they will have less time for their duties towards the employer.

Thomke in his article "Enlightened Experimentation - The New Imperative for Innovation" names the cost of the experimentation the main damper for the companies to create great products [21]. His list of "Essentials for Enlightened Experimentation" gives four rules for companies to be more innovative:

1. Organize for rapid experimentation.
2. Fail early and often, but avoid mistakes.
3. Anticipate and exploit early information.
4. Combine new and traditional technologies.

3 Agile Experimentation "Manifesto"

We think it is a time to give both the business professionals and researchers ready to use tool-set to align their goals in software engineering. Researchers want to perform controlled experiments that will give reliable enough conclusions leading to improvements in software engineering. Business professionals (e.g., an engineer working on a project or manager supervising it) are mostly interested in return of investment. The work done should make the customer happy and bring benefit for the organization. The results of potential experimentation is interesting for them only if the main goal is met and the additional experimentation cost is acceptably low. We want to give the methods and tools they can use to better meet those requirements. We gathered those methods and tools in popular nowadays form of "manifesto" [2, 3, 6, 7]. We have played with the form and proposed Agile Experimentation "Manifesto". The target of our "manifesto" are both the researcher as well as the practitioner willing to perform experiments. Some of the items might seem obvious for an experienced researcher but they are not necessary so obvious for an average practitioner. Thus the abstraction level of the items. Our "manifesto" contains the following rules:

1. Use small-n and single case experiments rather than large scale experiments to cut costs and enable experimentation.
2. Care about the power of your experiments to reduce waste.
3. Search for the best experiment design that fits your settings.
4. Use friction free tools for data gathering to not interfere with the real-world development environment.
5. Use just-in-time quantitative data rather than late, post project qualitative surveys to enable early informed decisions (on a basis of quantitative data instead of late anecdotal information).

What really hides behind those statements? How are they important for a business professional or a researcher wanting to do experiment in a real-world software engineering project.

3.1 Small-n and Single Case Experiments

As we mentioned earlier in this paper the main driver of a real-live business software engineering project is ROI, which is what the customer is most interested in. He puts his money into the project and expects to get the best possible software in exchange. There is no place for large scale experimentation in such project. There is no money for an experiment with a large number of professional developers. In fact there can be a problem with finding large number of professional developers willing to sacrifice their precious time for any kind of experiment. What can be done to change this situation? There is a way used by pharmacists and medical scientists for years. It is called small-n and single case experimentation [5, 10]. Those are special kind of experiments designed especially for very small samples (small-n) or even one participant (single-case). They are very helpful in clinical trails where the researcher do not have a large set of patients or when you study human behaviour. Often such kind of experiments are done as a low cost pre-studies before large scale and more expensive experiments.

This method took inroads into the software engineering already [24]. The reasoning is simple. Software development team is usually a small group of people (small-n) working on a single software project. In fact every developer working in a project is a case for itself. If psychologists are using small-n and single case experiments to study human behaviour why not use it in software engineering?

We are using this approach to study our extension to TDD (Test Driven Development) that we called CTDD (Continuous Test Driven Development) [17] in a real, industrial software development environment. We have promising results using the agile experimentation to study TDD versus CTDD. In fact the idea for agile experimentation was actively trialed during this experiment. We have used our own principles to design and conduct the TDD versus CTDD experiment. We are intending to publish a paper discussing the results and our experience with agile experimentation as soon as we finish the ongoing experiment.

The idea behind our experiment it to extend the commonly used TDD chain of actions:

write the test → execute it → see if it fails → satisfy the test → run the test → see if it succeeds → refactor → run the test → see if it still succeeds.

The extension involves using continuous testing (unit tests performed on a background thread in the development environment). Thanks to that technique we can eliminate the need of manually starting the tests and thus improve software development. This kind of improvement can be seen as a modification of the existing TDD practice in contrast to combining different software development practices together (e.g., the TDD and pair programming practices [13–15]). In CTDD the chain of actions is shorter:

write the test → see if it fails → satisfy the test → see if it succeeds → refactor → see if it still succeeds.

The reasoning is that a small increase of productivity in an individual developer can make a big impact in the whole developers community. The question is how omitting the need of manually executing the tests will impact the overall performance of the developer. That is what we want to estimate by conducting research following the idea of agile experimentation. We had to our disposal only one small (measured in terms of developers) project where only two developers were contracted. An ideal small-n experiment with two conditions (condition A: TDD and condition B: CTDD). But doing small-n and single case experimentation is challenging in another aspect. Precisely because of the small number of participants the researcher needs to be extra caution to be able to achieve enough statistical power to detect the difference between two conditions (software development practices, TDD and CTDD, in this particular case).

3.2 Care About Power of Your Experiments

The so called A/B experiments [12] are quite popular in the mainstream web development lately. They are used to perform simple online test where a new version of an existing website is created with, e.g., slightly different layout, colours, button orientation or alike and taken online alongside the old version. The old version (A—baseline) is showed to a randomly assigned part of the online audience and the new version (B—with intervention) to the other. The visitors behaviour is then recorded. Is the new layout causing them to stay on the page longer? Are they "liking" the page more if the colours are different? Do they click the "order" button more often in the new version? What is done here is a randomized controlled experiment, using between-subject design, where two groups of subjects are simultaneously tested: one being the control group (A) and one being under "treatment" (B). The subjects are simultaneously tested but assigned randomly to one of the groups, A or B. What is important is the ability to detect statistically significant difference between the treatments, i.e., the statistical power of the experiment is crucial. The power of any test of statistical significance is formally defined as the probability that a false null hypothesis will be rejected if there is no difference between the treatment and the baseline.

Researchers should be interested if the recorded difference between A and B is in fact statistically significant and, last but not least, what is the size of the effect [11, 15, 16]. The power of the test is especially important if performing small-n and single case experiments, see Sect. 3.1. To detect an effect of reasonable size the person performing agile experimentation will have to analyze power taking into consideration the small sample size. Another important aspect is generalization from such experiments. Agile experimentation may impede generalisation, but the threads to external validity can be minimized by the fact that agile experiments are conducted in a real world environment, by real software developers, not in a laboratory environment by inexperienced student subjects as is often the case.

In a simple online A/B test the, so called, random allocation is used. A visitor is randomly put into a group that sees A or B version of a web site. It is enough for such a simple experiment, but what if a researcher performs a more sophisticated small-n or single case experiment? In such case, the more possible experiment arrangements the better. The researcher needs to eliminate all competing explanations for the effect he is observing.

Let us consider an experiment with four phases. Every phase is either a baseline (A) or a treatment condition (B). The moment when to switch between A and B can be randomly chosen. This way the possible arrangements are: switch after the first phase—ABBB, switch after the second phase—AABB, or after the third phase—AAAB. The set of possible arrangements is quite small. What if the researcher decides to use a completely random approach? This means to simply choose every phase randomly? Using this approach he will get 16 (2^4) possible arrangements like this: AAAA, BAAA, ABAA, AABA, AAAB, BBAA, ABBA, AABB and so on. But some of the arrangements are not desirable from the beginning (like AAAA or BBBB). In that case the researcher can decide to gather the phases in blocks. For example always in pairs like AABBAA, AAAABB, BBAAAA, and so on.

Let us consider the TDD versus CTDD experiment. Lets assume we have 8 modules to do. We have 2 software developers so each one may get 4 modules to write. There are 6 possible arrangements within these 4 modules (4 observations, 2 for A and 2 for B): AABB, ABBA, BBAA, BAAB, ABAB, BABA. For 2 developers there is $6 * 6 = 36$ possible arrangements. Maybe switching from the module/package level to the class level (the number of classes is usually larger than the number of packages in Java or solutions in .NET) in the TDD versus CTDD experiment will be give us higher power to detect the difference? Maybe another design of experiment will be better (for example randomized block design)? This kind of a priori consideration can pay off if the researcher strives for reliable results.

3.3 Searching Best Experiment Design

Finding an appropriate design for a given experiment in software engineering is not an easy task. If it is an small-n or single case experiment it might be even harder because it is not common in software engineering. There might be a need to extrapolate the types of experiments from different fields of study onto software engineering. For example, Dugard et al. [4] give a set of example experimental designs. There is a lot to choose from: single-case/small-n one way, single-case randomize block design, small-n repeated measures, small-n repeated measures with replicates, two-way factorial single-case/small-n, single-case AB, single-case ABA, multiple baseline AB, multiple baseline ABA. One design may be better for one kind of experiment and the other for another type. But how to choose the right one? The one that will answer the posed research question in the best way? There is not an easy way to choose a design. There is no tool for searching the available design given project and research constrains. For example in our research about TDD versus CTDD we had a small project that was contracted for 2 developers, having 160 h/month in the period of at least one year, with estimated 37 modules (400 classes) to write. We searched the possible set of designs answering the following important questions [4]:

- Do we have at least two participants? Yes, we have.
- Do we have two conditions to compare? Yes.
- Will each participant receive each of the condition on at least two occasions? Also yes (we have a lot of modules/classes to write and if we consider one as an "occasion" then we are good to go).
- Is it possible to randomly assign conditions to each participant? Yes—we can write a software tool that will randomly assign a class to a given treatment.

"A small-n repeated measure design with replicates" experimental design seemed to be the most suitable for our needs.

3.4 Use Friction Free Tools for Data Gathering

Software developers refer to tools that do not need much attention while in use as "friction free tools". It is because "friction free tools" are not generating "resistance" while in use. Meaning they need no special attention from the developer. It is quite important to use such tools in agile experimentation while gathering data. Agile experimentation tools should work without the developer attention. They have to integrate seamlessly with the developer environment. After simple installation and minimal configuration they should be up and running. Such friction free data gathering software tools need to be quite resilient. If the data gathering relies on network communication and the developer decides to work from home, they have to provide fall back scenarios so the data will not be lost.

What kind of tools will be needed in any given research depends on the planned experiment. There is quite a number of tools already written and available for various range of experiments. That was the case in our empirical evaluation of new approaches to software engineering—TDD versus CTDD. We found very good continuous testing plug-in for IDE that was used in the project under investigation. We needed only to extend it with the data gathering capabilities. It was an open source project so we “forked it”, meaning created our branch of source code and added a small data gathering part. We saved measurements like: when, what test and with what outcome were run, how long it took to complete them. The detailed description of the tool we called AutoTest.NET4CTDD can be found in [17].

The second tool we created was a small randomized block generation. We mentioned it in Sect. 3.2. It was a tool that randomly assign a condition to a given class. It simply assigned a newly created class to TDD or CTDD group by adding a specially formatted line at the beginning of the file. The line was read by AutoTest.NET4CTDD and the plug-in acted accordingly. It automatically turned on or off the continuous testing.

We needed one more tool: reliable event recorder for our IDE. We were quite surprised when we realized there is none available. So in this case we have written it from scratch and open sourced it for the community. The NActivitySensor tool is described in Sect. 4.

3.5 Use Just-in-Time Quantitative Data

We do not advocate discontinuing the traditional research approaches involving semi-quantitative or qualitative data gathered after the experiment using various kinds of surveys. We suggest to enrich this kind of data with a quantitative data based on experiments, both, large and small scale (e.g., small-n, single-case). We strongly believe that such data can be gathered during the research without an additional effort from the researcher and the subject. This data gathering can occur in a just-in-time manner. Meaning the data is gathered in real-time as it happens and is logged immediately for researcher to be used in his convenience. We have build our TDD versus CTDD research tools in that way. Both AutoTest.NET4CTDD[2] and NActivitySensor[3] are storing the data in a central database from where they can be used as they are needed.

[2]Available at https://github.com/ImpressiveCode/ic-AutoTest.NET4CTDD.

[3]Available at https://github.com/ImpressiveCode/ic-NActivitySensor.

4 NActivitySensor

For the purpose of this agile experimentation we created a tool called NActivity-Sensor. It is a "friction free" kind of software tool we described in Sect. 3.4. It is designed to gather the real-time quantitative data while working. NActivitySensor was developed with the third and forth Agile Experimentation rule in mind. It was co-developed at the software development company one of the authors is running. The company is a mostly Microsoft .NET shop that uses C# as his primary language. The main IDE for .NET is Microsoft Visual Studio (VS). Investigation of the VS extensions showed no tool that can be used to record the developer actions in IDE. The data recorded by such tool could then be used in different type of agile experimentation. One of the authors of this paper supervised the creation of similar tool that integrated with Java IDE Eclipse. This tool was called Activity Sensor and developed back in 2006 at Wrocław University of Technology. We decided to name the new tool NActivitySensor. N in the name is often used in the .NET tools and libraries as the indication of the, in most cases, Java based ancestors. NActivitySensor integrates with VS IDE and hooks to Development Tools Environment (DTE) and its "Test subsystem".

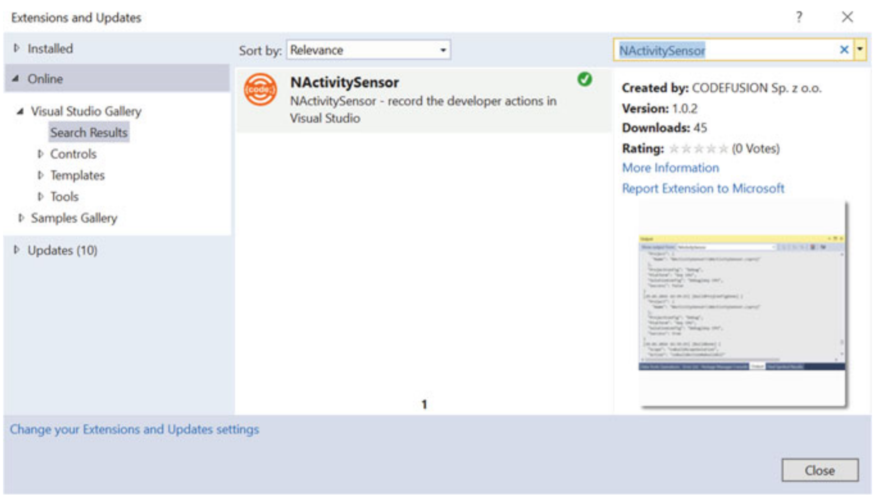
We are currently using the NActivitySensor to gather data for our research on Continuous Test Driven Development (CTDD) [17]. The detailed description of NActivitySensor is given in Appendix A.

5 Discussion and Future Work

The proposed agile experimentation practice and supporting tool are proposed to make experimentation in real-world, industrial settings more widely adapted. The agile experimentation practice and supporting tool set will be refined as part of our further research. We are currently working on a new publication where we are discussing the TDD versus CTDD experimentation. This research is done solely according to the agile experimentation principles. The overall experiences regarding agile experimentation from this ongoing project are promising. We were able to harness the most of our "manifesto" principles to action. In this research we are targeting professional developers in business driven projects. It is not easy to get the permission to experiment in such environment. Thus if permission is granted it is advisable (to say the least) to make the most of it. By using well known small-case and single-n experiments designs (1) and to take care about the power of the experiment up front (2). The act of experimenting should not impact the work of the developers much (4). With this goal in mind the NActivitySensor was developed. It gathers the information about developer activities in the Visual Studio IDE without disturbing developer's work. The data is then gathered immediately for the sake of quantitative research. We will work further on the agile experimentation idea in order to provide a tool for searching the best experiment design for a given project and research constraints (3).

Appendix A—NActivitySensor

This appendix contains description of the details of NActivitySensor Visual Studio Add-in. NActivitySensor is available as an extension [9] at Visual Studio Gallery. Visual Studio Gallery is a tools, controls, and templates distribution platform used in Visual Studio. NActivitySensor is maintained and supported as a free extension by its creators. It can be installed from withing Visual Studio by using "Extension and Updates…" from the "Tools" menu (see Fig. 1). After successful installation NActivitySensor writes the activities log to a new output window with the same name (see Fig. 2). It is possible to configure the extension to write the activity log into a database. The database engine supported in NActivitySensor is Microsoft SQL Server. The database connection string is set in NActivitySensor.dll.config in the extension installation directory (which is `C:Users\Account_Name\AppData\Local\Microsoft\VisualStudio 14.0\Extensions`). The key for the connection string is stored in NActivitySensor.MSSql.ConnectionString. Also it is possible to configure the extension on a project level. In order to do it a configuration file in the Visual Studio solution directory is needed. It needs to be called NActivitySensor.config. Example configuration file is showed on Listing 1.



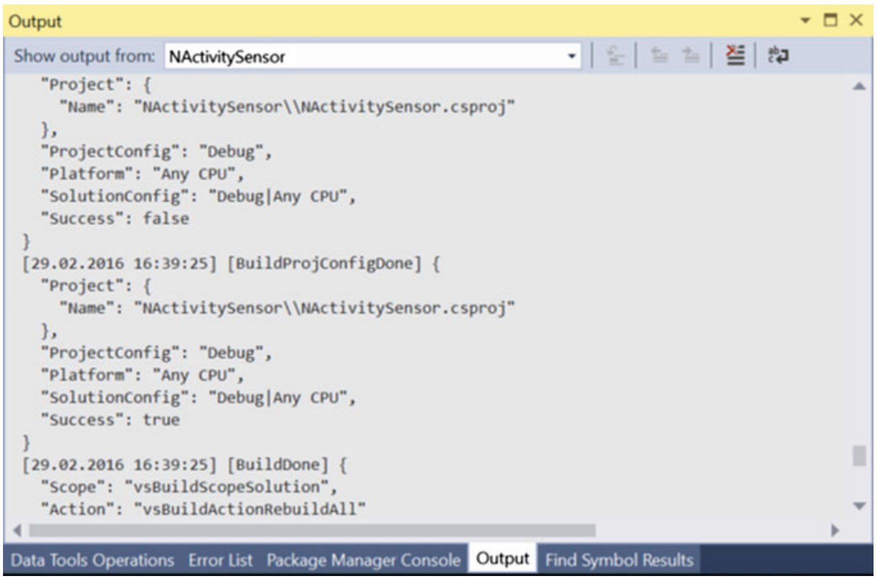
Sort by: Relevance

NActivitySensor

NActivitySensor
NActivitySensor - record the developer actions in Visual Studio

Created by: CODEFUSION Sp. z o.o.
Version: 1.0.2
Downloads: 45
Rating: ☆☆☆☆☆ (0 Votes)
More Information
Report Extension to Microsoft

Fig. 1 NActivitySensor installation in Visual studio 2015



```
  "Project": {
    "Name": "NActivitySensor\\NActivitySensor.csproj"
  },
  "ProjectConfig": "Debug",
  "Platform": "Any CPU",
  "SolutionConfig": "Debug|Any CPU",
  "Success": false
}
[29.02.2016 16:39:25] [BuildProjConfigDone] {
  "Project": {
    "Name": "NActivitySensor\\NActivitySensor.csproj"
  },
  "ProjectConfig": "Debug",
  "Platform": "Any CPU",
  "SolutionConfig": "Debug|Any CPU",
  "Success": true
}
[29.02.2016 16:39:25] [BuildDone] {
  "Scope": "vsBuildScopeSolution",
  "Action": "vsBuildActionRebuildAll"
```

Fig. 2 NActivitySensor output window in Visual studio 2015 IDE

Listing 1 NActivitySensor.config file

```
<?xml version="1.0"?>
<configuration>
  <appSettings>
    <add key="NActivitySensor.MSSql.ConnectionString"
         value="Data Source=192.168.1.102;
Initial Catalog=ExCalcNActivitySensorReports;
User Id=NActivitySensor;
Password=NActivitySensor;
MultipleActiveResultSets=True"/>
  </appSettings>
</configuration>
```

The internal storage format for the activities is JSON. JSON stands for (JavaScript Object Notation) and is a widely used as a data exchange format. It was used because it is both: (1) easy to read and write by a human and a machine and (2) its structure does not have to be defined beforehand. We are using JSON to store various activities with varied format. For example Listing 2 shows the log for the activity ‘DocumentOpened’ (opening a document in IDE) and Listing 3 shows the ‘Build-Begin’ (starting of project building in Visual Studio).

Listing 2 DocumentOpened activity example log

```
[22.03.2016 15:24:21] [DocumentOpened] {
  "Name": "Examples.cs",
  "Kind": "{8E7B96A8-E33D-11D0-A6D5-00C04FB67F6A}",
  "Path": "C:\\Dev\\ClaToDot\\ClaToDot.Demo\\" }
```

Table 1 Microsoft Visual Studio events hooked in NActivitySensor

Event group	Event	Event group	Event
SolutionEvent	SolutionOpened, SolutionBeforeClosing, SolutionRenamed, Solution-QueryClose,SolutionProjectRenamed, SolutionProjectRemoved, SolutionProjectAdded,	TextEditorEvent	LineChanged
BuildEvent	BuildDone, BuildProjConfigDone, BuildBegin, BuildProjConfigBegin	TaskEvent	TaskRemoved, TaskNavigated, TaskModified, TaskAdded
UserEvent	UserInactive, UserActiveAgain	FileItemEvent	FileItemRenamed, FileItemRemoved, FileItemAdded
PluginEvent	Connect, Connection, Disconnection, AddInsUpdate, StartupComplete, BeginShutdown	FindEvent	FindDone
WindowEvent	WindowMoved, WindowCreated, WindowClosing, WindowActivated, WindowPaneUpdated, WindowPaneClearing, WindowPaneAdded	DebuggerEvent	DebuggerExceptionThrown, DebuggerException-NotHandled, DebuggerEnterRunMode, DebuggerEnterDesignMode, DebuggerEnterBreakMode, DebuggerContextChanged
SelectionEvent	SelectionChange	CommandEvent	CommandBeforeExecute, CommandAfterExecute
		DocumentEvent	DocumentClosing, DocumentSaved, DocumentOpened

Listing 3 BuildBegin activity example log

```
[22.03.2016 15:28:27] [BuildBegin] {
  "Scope": "vsBuildScopeSolution",
  "Action": "vsBuildActionRebuildAll" }
```

Table 1 shows all hooked-up events recorded by the NActivitySensor.

The data is stored in the Microsoft SQL Server database. Additionally the data is echoed back into VS output windows as a fall-back for non functioning database or network (it is possible to record the data from the output window to a file).

References

1. Basili, V.R.: What's so hard about replication of software engineering experiments? https://www.cs.umd.edu/~basili/presentations/RESER (2011). Accessed 18 Mar 2016
2. Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R.C., Mellor, S., Schwaber, K., Sutherland, J., Thomas, D.: Manifesto for Agile Software Development (2001)
3. Bonr, J., Farley, D., Kuhn, R., Thompson, M.: The reactive manifesto. http://www.reactivemanifesto.org/ (2014). Accessed 29 Mar 2016
4. Dugard, P., File, P., Todman, J.: Single-case and Small-n Experimental Designs: A Practical Guide to Randomization Tests, 2nd edn. Routledge (2012)
5. Gast, D., Ledford, J.: Single Case Research Methodology: Applications in Special Education and Behavioral Sciences. Taylor & Francis (2014)
6. Guevara, P.C.: Manifesto for minimalist software engineers. http://minifesto.org/ (2013). Accessed 29 Mar 2016
7. Harman, M., Jia, Y., Langdon, W.B.: A Manifesto for higher order mutation testing. In: Proceedings of the Third International Conference on Software Testing, Verification, and Validation Workshops. ICSTW'10, pp. 80–89. IEEE (2010)
8. Juristo, N., Moreno, A.M.: Basics of Software Engineering Experimentation, 1st edn. Springer Publishing Company, Incorporated (2010)
9. Kawalerowicz, M., CODEFUSION: microsoft visual studio extension NActivitySensor. https://visualstudiogallery.msdn.microsoft.com/4675d6fb-2608-48ed-ae0a-320b3a756047 (2013–2016). Accessed 18 Mar 2016
10. Kazdin, A.E.: Single-case Research Designs: Methods for Clinical and Applied Settings. Oxford University Press (2011)
11. Kitchenham, B.A., Madeyski, L., Budgen, D., Keung, J., Brereton, P., Charters, S., Gibbs, S., Pohthong, A.: Robust Statistical Methods for Empirical Software Engineering. Empirical Software Engineering (in press) (2016). http://dx.doi.org/10.1007/s10664-016-9437-5. doi:10.1007/s10664-016-9437-5
12. Kohavi, R., Longbotham, R., Sommerfield, D., Henne, R.M.: Controlled experiments on the web: survey and practical guide. Data Min. Knowl. Discov. **18**(1), 140–181 (2008). http://dx.doi.org/10.1007/s10618-008-0114-1
13. Madeyski, L.: On the effects of pair programming on thoroughness and fault-finding effectiveness of unit tests. In: Münch, J., Abrahamsson, P. (eds.) Product-Focused Software Process Improvement, Lecture Notes in Computer Science, vol. 4589, pp. 207–221. Springer, Berlin, Heidelberg (2007). http://dx.doi.org/10.1007/978-3-540-73460-4_20. doi:10.1007/978-3-540-73460-4_20
14. Madeyski, L.: Impact of pair programming on thoroughness and fault detection effectiveness of unit test suites. Softw. Process: Improve. Pract. **13**(3), 281–295 (2008). http://dx.doi.org/10.1002/spip.382. doi:10.1002/spip.382

15. Madeyski, L.: Test-Driven Development: An Empirical Evaluation of Agile Practice. Springer, Heidelberg, London, New York (2010). http://dx.doi.org/10.1007/978-3-642-04288-1. doi:10.1007/978-3-642-04288-1
16. Madeyski, L., Jureczko, M.: Which process metrics can significantly improve defect prediction models? An empirical study. Softw. Qual. J. **23**(3), 393–422 (2015). http://dx.doi.org/10.1007/s11219-014-9241-7. doi:10.1007/s11219-014-9241-7
17. Madeyski, L., Kawalerowicz, M.: Continuous test-driven developmenta novel agile software development practice and supporting tool. In: Maciaszek, L., Filipe, J. (eds.) ENASE 2013—Proceedings of the 8th International Conference on Evaluation of Novel Approaches to Software Engineering, pp. 260–267 (2013). http://madeyski.e-informatyka.pl/download/Madeyski13ENASE.pdf. doi:10.5220/0004587202600267
18. Madeyski, L., Piechowiak, A.: Exclipse plug-in activity sensor. http://sens.e-informatyka.pl/projekty/activity-sensor/ (2006). Accessed 18 Mar 2016
19. Madeyski, L., Szała, Ł.: Impact of aspect-oriented programming on software development efficiency and design quality: an empirical study. IET Softw. **1**(5), 180–187 (2007). http://dx.doi.org/10.1049/iet-sen:20060071. doi:10.1049/iet-sen:20060071
20. Madeyski, L., Szała, Ł.: The impact of test-driven development on software development productivity—an empirical study. In: Abrahamsson, P., Baddoo, N., Margaria, T., Messnarz, R. (eds.) Software Process Improvement, Lecture Notes in Computer Science, vol. 4764, pp. 200–211. Springer, Berlin, Heidelberg (2007). http://dx.doi.org/10.1007/978-3-540-75381-0_18. doi:10.1007/978-3-540-75381-0_18
21. Thomke, S.: Enlightened experimentation—the new imperative for innovation. Harv. Bus. Rev. **79**(2), 66–75 (2001)
22. Tichy, W.F.: Should computer scientists experiment more? Computer **31**(5), 32–40 (1998)
23. Vokáč, M., Tichy, W., Sjøberg, D.I.K., Arisholm, E., Aldrin, M.: A controlled experiment comparing the maintainability of programs designed with and without design patterns–a replication in a real programming environment. Empir. Softw. Eng. **9**(3), 149–195 (2004)
24. Zendler, A., Horn, E., Schwärtzel, H., Plödereder, E.: Demonstrating the usage of single-case designs in experimental software engineering. Inf. Softw. Technol. **43**(12), 681–691 (2001). http://dx.doi.org/10.1016/S0950-5849(01)00177-X. doi:10.1016/S0950-5849(01)00177-X

An Industrial Survey on Business Analysis Problems and Solutions

Piotr Marciniak and Aleksander Jarzębowicz

Abstract The paper focuses on the problems reported by business analysts which have a negative impact on their work and on the applicability of available business analysis (BA) techniques as solutions to such problems. A unified set of BA techniques was developed on the basis of 3 industrial standards associated with IIBA, REQB and IREB certification schemes. A group of 8 business analysts was surveyed to list problems they encounter in their work activities and assess their frequency. A subset of most frequent problems was further analyzed and solutions were proposed by selecting the most suitable BA techniques. Solution proposals were validated through follow-up discussions with business analysts. The results indicate that the unified set of techniques addresses the problems reported by practitioners and solution proposals are generally accepted as valid, although several techniques can be used interchangeably.

Keywords Requirements engineering • Business analysis • Certification schemes • Industrial standards • Survey

1 Introduction

Since the software crisis in 1960s, requirements engineering (RE) is recognized as one of the crucial aspects of software projects. Also currently, requirements engineering and business analysis (BA—understood as a broader term, which encompasses more activities) strongly influence project's results. The post-mortem

P. Marciniak
Active Code Piotr Marciniak, Gdańsk, Poland
e-mail: chm.pm@hotmail.com

A. Jarzębowicz (✉)
Department of Software Engineering, Faculty of Electronics,
Telecommunications and Informatics, Gdańsk University of Technology,
Gdańsk, Poland
e-mail: olek@eti.pg.gda.pl

L. Madeyski et al. (eds.), *Software Engineering: Challenges and Solutions*,
Advances in Intelligent Systems and Computing 504,
DOI 10.1007/978-3-319-43606-7_12

reviews of software project failures and the reasons behind them reveal that RE/BA issues are among top factors contributing to project success or failure [1, 2].

The importance of this subject is widely recognized, which can be confirmed by issued standards, published books, presence among topics of software engineering conferences and a growing number of certification schemes (and certificates issued) for RE/BA practitioners.

As the research reported in this paper concern Polish software industry, we would like to focus more on the local context. The recognition of RE/BA is also visible in Poland and a trend of growing interest can be noticed. Dedicated job positions of "business analyst", "system analyst" or similar are becoming more common. The available certification paths associated with International Institute of Business Analysis, International Requirements Engineering Board and Requirements Engineering Qualifications Board are becoming more popular (partially thanks to Polish versions of training materials). A significant number of training courses is offered—some dedicated to particular certification exams, while others more general, based on several sources. Another sign of interest are the recently published books entirely dedicated to requirements, either being (promptly) translated from international publications [3] or written by Polish authors [4].

As a result, many sources of knowledge about RE/BA became available and numerous BA techniques dedicated to requirements elicitation, analysis, specification and validation are described in the literature. On the other hand, RE/BA is still perceived as a difficult and error-prone part of the software project and the problems related to e.g. cooperation with stakeholders, obtaining the necessary information or scope creep are quite common. Therefore, the question we would like to ask is how well do available BA techniques address the problems encountered by the business analysts in their everyday work. For this purpose we planned and conducted a research study involving practitioners from Polish IT industry.

2 Related Work

Our research included: identifying frequent problems encountered in BA activities, analyzing state of the art BA techniques and selecting the techniques which provide solutions to particular problems. Two main areas of related work are: surveying the industry about RE/BA related issues and comparing RE/BA techniques.

A number of surveys (based on questionnaires or interviews) about requirements engineering in the industry are available, but most of them focus more on learning about processes and practices actually used [5, 6] than on problems encountered. Davey and Parker [7] provides a summary of surveys related on requirements elicitation problems, but it does not cover other RE/BA areas like requirements analysis or validation. A list of most common requirements problems is included in [8], however it is only based on the author's industrial experience, instead of a wider survey. The most similar approach is reported in [9], which describes a survey on problems and practices of the software industry in Malaysia. It is also worth mentioning that we are not aware of any recent survey research on RE/BA in Polish industry, except [10], which focuses on a particular issue (hidden requirements anti-pattern).

Several works comparing RE/BA techniques gathered from different sources are available [11, 12], however the referenced sources are original papers describing new techniques and/or RE textbooks, no comparison between present industrial standards has been found. A comparison of BABOK, IREB and SWEBOK is provided in [13], but with respect to the general approaches, terms used etc., not techniques included. Selected techniques dedicated to requirements elicitation [14, 15] or prioritization [16] were assessed, but with respect to the predefined criteria or measurements in controlled experiments, not applicability to particular problem situations.

3 Research Study

Our research aimed at addressing the following questions:

- Which BA techniques are recommended by the state-of-the-art sources?
- What problems affecting BA are perceived as most frequent by business analysts?
- Are the available techniques effective in coping with such problems according to business analysts' opinions?

It should be stressed that we sought for the problems from the perspective of business analyst (not e.g. company or customer) and from the practical viewpoint (real life experiences). As for the BA techniques, we decided to focus on areas of requirements development (elicitation, analysis, specification and validation) and to exclude requirements management.

The research was conducted in a number of steps. Each of the steps included several subsequent activities. This process is illustrated in Fig. 1.

3.1 Step 1—Development of a Unified Set of BA Techniques

The prerequisite to fulfill the aim of step 1 was to identify the candidate sources of knowledge and to choose the basis for further work. At first, we tried the international standards. However, the current main requirements engineering standard (ISO/IEEE 29148:2011 [17]) is not very elaborate in regard to this matter. It lists and briefly summarizes several techniques for e.g. elicitation (page 22) or validation (page 31), but mainly focuses on other issues like requirements engineering process or SRS contents. The previous standards (ISO/IEEE 830:1998 [18] and ISO/IEEE 1033:1998 [19]), superseded by 29148, contain even less information about particular techniques.

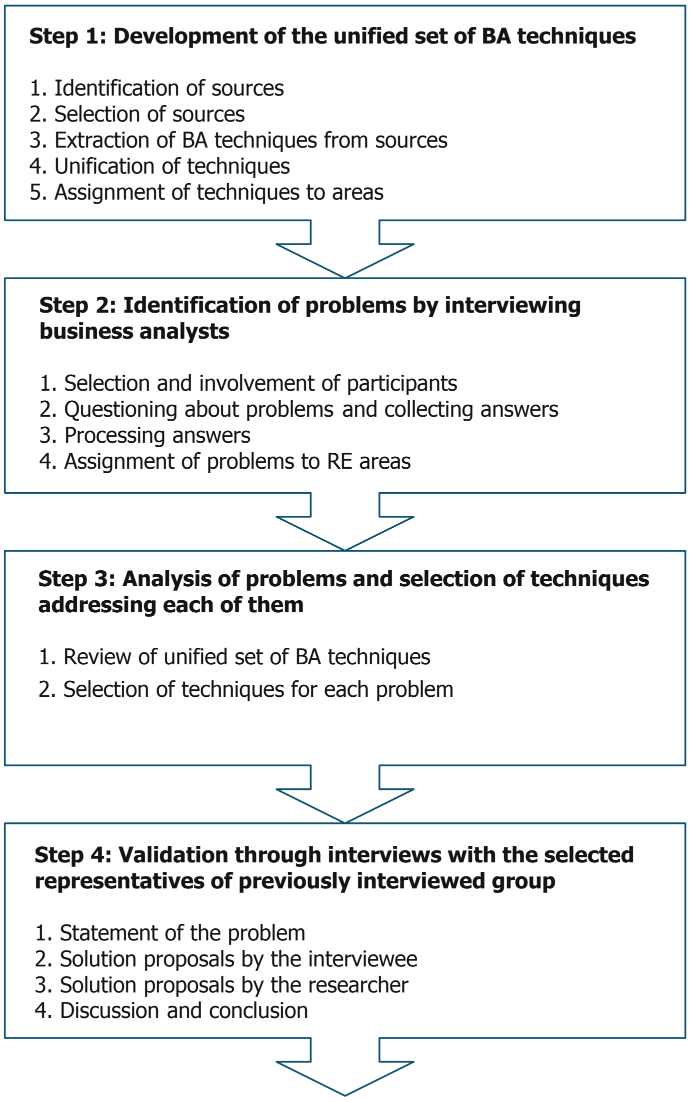

Step 1: Development of the unified set of BA techniques
1. Identification of sources
2. Selection of sources
3. Extraction of BA techniques from sources
4. Unification of techniques
5. Assignment of techniques to areas
Step 2: Identification of problems by interviewing business analysts
1. Selection and involvement of participants
2. Questioning about problems and collecting answers
3. Processing answers
4. Assignment of problems to RE areas
Step 3: Analysis of problems and selection of techniques addressing each of them
1. Review of unified set of BA techniques
2. Selection of techniques for each problem
Step 4: Validation through interviews with the selected representatives of previously interviewed group
1. Statement of the problem
2. Solution proposals by the interviewee
3. Solution proposals by the researcher
4. Discussion and conclusion

Fig. 1 The overview of the research process

We turned to certification schemes and associated "industrial standards" instead. We consider them as representative to the present industrial practice—the growing numbers of certificates issued indicate the interest of practitioners and the syllabi/examination criteria are updated to reflect the current trends. Also, an important factor from the point of view of our research was that many particular BA techniques are described in these sources. We decided not to rely on one certification scheme only, but to analyze several ones, compare them with respect to the recommended BA techniques and to develop a unified list of such techniques based on all analyzed sources. We selected certification approaches established by International Requirements Engineering Board (IREB), Requirements Engineering Qualifications Board (REQB) and International Institute of Business Analysis (IIBA). The following documents describing these 3 approaches were used:

- IIBA BABOK Guide ver. 2 [20],
- IREB CPRE Foundation level syllabus ver. 2.2 [21],
- IREB CPRE Elicitation and Consolidation, Advanced Level syllabus ver. 1.0 [22],
- REQB CPRE Foundation Level syllabus ver. 2.1 [23],
- REQB CPRE Advanced Level Requirements Manager ver. 1.0 [24].

We used the documents which were available at the time and e.g. IREB CPRE Advanced Level Requirements Modeling syllabus was published later. As for BABOK, its current version (3) was published when our work was already in progress (and the standard was not immediately available to us), so we decided to proceed with BABOK 2. Of course, our comparison of 3 approaches can be updated to reflect changes in BABOK contents, but in the interview-based study (Sects. 3.2–3.4) we used techniques from BABOK 2, so we do not introduce such change to this paper for consistency sake.

All the documents were reviewed to identify particular BA techniques. This task was not as straightforward as it may appear. In BABOK most of BA techniques are explicitly listed and described in separate sections, but some techniques e.g. RACI Matrix are not included in the list, only mentioned in the text. The other two sources do not as explicitly focus on techniques. REQB enumerates most of them in tables or as bullet items, but some are only mentioned in accompanying description (e.g. several analysis techniques based on various models are listed in Table 4 on p. 76 [23], but prototyping is only mentioned on p. 79). For IREB, [21] only enumerates the techniques, while descriptions are provided in [22], but these two sets of techniques have some differences. Such differences can also be spotted in REQB sources e.g. [23] lists and describes 11 elicitation techniques, while [24] omits one of them: use cases.

The review of the sources was not just about extracting the name of each technique mentioned in the text—we aimed at developing a unified list of techniques based on 3 sources. It required several actions and decisions to be made. The first and easiest issue were language differences—quite often similar techniques are given different names e.g. Observation (BABOK) and Field Observation (REQB and IREB); Functional Perspective (IREB), Functional Decomposition (BABOK) and Logical Analysis including Functional Decomposition (REQB). Sometimes however, the difference was not just about names e.g. Document Analysis (BABOK) and System Archaeology (IREB) look similar, but the latter includes analysis of existing system code, while the former does not explicitly mention it and is more document-oriented.

Another issue was to decide if some variants of a more general technique should be recognized as separate techniques. For example, Brainstorming is listed as one of elicitation techniques both in BABOK and REQB, while IREB provides several variants of brainstorming and creativity techniques e.g. Method 6-3-5 or 6 Thinking Hats. A similar situation is about reviews because different sources explicitly mention various kinds of this technique (peer review, technical review, informal review etc.). There is no space available here to describe each decision we had to make, in general we tended to merge very detailed variants into one (e.g. Reviews), but we were careful not to step too far (e.g. we distinguish Reviews and Walkthroughs).

The techniques were also assigned to requirements development areas [3, 25]: elicitation, analysis, specification, validation. Specification area differs from the others—neither of sources provides details on SRS contents. BABOK enumerates several kinds of specification documents, IREB refers to IEEE 29148:2011 [17], while REQB to the older standard (IEEE 830:1998 [18]). We compared two latter ones, but it is de facto a comparison between two ISO standards with respect to SRS contents and requirements categories. As example, the resulting set of techniques (only the ones assigned to elicitation area) and their traceability to sources are shown in Table 1.

For each item of the resulting unified set of techniques, an analysis of their applicability according to sources (advantages, limitations) was made. Information from all sources which included a given technique was compiled into a more comprehensive description of its applicability.

3.2 Step 2—Identification of Problems

Step 2 aimed at identification of problems encountered in business analysts' professional experience. It started with gathering a group of analysts and making arrangements for interviews.

The participating analysts represented two companies (4 analysts from each one). For confidentiality sake we will use names Company A and Company B. Company A employs about 140 staff and specializes in outsourcing of IT services and development of web-based solutions for business. Projects are mostly run using agile methodology, by relatively small teams (4–10 persons per project). Company B employs over 350 persons and is mainly active in the financial industry, but includes its own IT department (30 persons) responsible for IT infrastructure, data storage and software development. Software projects are managed using various approaches and methodologies e.g. waterfall model, V model, agile—depending on project's size and other constraints. Project teams vary greatly (from 3 to 20 persons) and some projects, especially maintenance-oriented ones, are rather short-staffed.

These two companies were selected because of their different profiles. A number of business analysts from both companies were initially identified as potential interviewees through the network of professional contacts of one of us. Only persons with designated job position as business analyst and experience in this field were approached. The participants of the study were recruited by contacting them in

Table 1 Requirements elicitation techniques from IREB, REQB and IIBA sources

Technique	IREB	REQB	IIBA
Apprenticing	X	X	
Benchmarking			X
Brainstorming	X	X	X
Context modeling	X	X	X
Contextual inquiry	X		
Customer representative on site		X	
Decision analysis			X
Document analysis	X	X	X
Elevator pitch	X		
Focus groups			X
Functional decomposition	X	X	X
Interface analysis			X
Interviews	X	X	X
Observation	X	X	X
Organization modeling			X
Persona	X	X	
Perspective-based reading	X		
Problem tracking			X
Process analysis/modeling	X	X	X
Prototyping	X	X	X
Questionnaires/surveys	X	X	X
RACI matrix			X
Reuse of requirements	X	X	X
Scenarios	X		X
Self-recording		X	
Stakeholder map/classification	X	X	X
Storyboards	X		
Use cases	X	X	X
User stories	X	X	X
User-Centered Design	X		
Walkthrough	X	X	X
Workshop	X	X	X

person or via email, explaining the rationale and scope of the study and asking them to participate. Initially more people were asked, but some either refused to participate or proved uncooperative. The participation was entirely voluntary, the study was not e.g. endorsed by the management.

The final group consisted of 8 people with a substantial experience in the field of business analysis:

- 5 persons with more than 5 years of experience;
- 3 persons with the experience between 2 and 5 years.

Each of 8 participants was individually asked about the problems encountered in his/her job experience using an open question: “As an analyst, what problems do you encounter most often in your work?”. The participant was supposed to list as many problems as he/she could think of. Also, he/she was asked to evaluate each of the mentioned problems with respect to the frequency of its occurrence. The frequency was measured using an ordinal scale (1—least frequent, 10—most frequent). Answers were collected within two weeks by e-mail.

After collecting all answers, a “data processing” activity was conducted. Answers for an open question usually require some clarification and this case was no exception. In particular, it was essential to merge the answers which reported the same problem, but using different natural language expressions.

For example, one interviewee listed as problems: “Functional changes after user acceptance tests” and “Additional requirements issued during customer-analyst meetings, compared to already agreed and prioritized requirements”, while another one reported “Changes of functionality during the whole project”. These 3 sentences were merged into a more encompassing one: “Changing requirements”.

In all such cases the problem was only listed once (preferably under the most meaningful name), but with a sum of all frequency scores. Sometimes a clarification and/or refinement was required when we had doubts about the meaning of a particular problem or the problem was too generic (e.g. “communication problems”). In such cases, a participant was contacted to clarify doubts and/or provide additional details. Also, the problems that turned out not directly related to RE/BA (but to e.g. company politics or interpersonal issues instead) were rejected or refined.

Next, a classification of problems into the particular areas (elicitation, analysis, specification, validation) was done. A given problem could be assigned to one or more areas (e.g. “Lack of stakeholders’ commitment” problem was assigned to elicitation and validation areas, while “Changing requirements” problem was considered to affect all four areas). Four problems were revealed to belong to requirements management area (excluded from study scope) and were omitted from further analysis.

Some reported problems were quite surprising to us, because their source turned out to be the analyst, not the customer, market situation or other independent factor. For example: “An analyst prematurely assumes that he/she understands stakeholder’s requirements” (resulting in lack of commitment to pursue the issue further), or “An analyst skips recording some requirements during elicitation phase”. At first, we intended to exclude such problems from the further analysis, as it appeared that they stem from analyst’s negligence. If so, then no advanced BA technique, but rather a more responsible approach of the analyst to his/her duties is required. However, follow-up contacts and requests for explanation revealed that the interviewees encountered such problems working together with their fellow analysts (often less experienced ones) and believed that application of a technique capable of preventing such errors would be beneficial. Finally, we decided to treat those problems like all others and try to find appropriate solutions to them.

The final list of problems reported by interviewees included 49 items, together with 86 frequency scores. The top 15 items of the list ordered by summarized

Table 2 Results of interviews—problems with the highest summary scores

#	Problem	Score	E	A	S	V
1	Changing requirements	30	X	X	X	X
2	Too short deadlines to complete BA	30	X	X	X	X
3	Lack of the authorized stakeholders (capable of making decisions)	30	X			
4	The stakeholders are unable to express their needs/requirements	29	X			
5	Analyzing undocumented existing system	28	X	X		
6	Lack of stakeholders' commitment	19	X			X
7	The stakeholders completely do not know what they want	15	X			
8	The stakeholders express requirements which are outside system's scope	14	X			
9	Failure to identify an essential stakeholder	13	X			
10	The stakeholders are unavailable, difficult/delayed contact	13	X			
11	The software developers ignore specified requirements	13			X	
12	Low quality of specified requirements (e.g. incomplete, too generic)	12			X	
13	The stakeholders avoid participating in Verification and Validation activities	12				X
14	Conflicting requirements	12	X	X		
15	Ambiguous requirements' descriptions	12			X	

frequency scores are presented in Table 2. For each problem the areas it concerns (elicitation, analysis, specification, validation) are also marked. It is visible from the summary scores (and the ratio: 49 items—86 scores) that the sets of problems stated by particular interviewees differ from each other. This is obviously the result of an "ad hoc" manner of identifying problems, however it was intentional—we wanted not to restrict the potential outcome by e.g. providing a checklist of problems. We assumed, that if a particular issue is really a frequent problem in the analyst's working activities, then he/she will remember about it and include it in the list.

3.3 Step 3—Solution Proposals

In step 3, the analysis of gathered problems and available BA techniques was planned to propose solution to each problem. However, because of such a significant number of reported problems, we decided to exclude some of them and focus on the ones with higher frequency scores (28 out of 49).

The search for appropriate solutions to problems was based on the guidelines for applying each particular technique (description, pros and cons) compiled from 3

Table 3 Techniques used as solutions to problems

	Elicitation	Analysis	Specification	Validation
Total no. of techniques in the unified set	32	16	35	6
No. of techniques used	11	8	8	2

sources described in Sect. 3.1. The process was iterative, first several candidate techniques were considered, then a selection of the most promising solutions (up to 3 techniques) was made. The process was based on the analysis of issues expressed in natural language, therefore it is hardly possible to describe it in an algorithmic form, with precise decision criteria.

Only some of the techniques were selected as solutions to considered problems. Table 3 shows how many techniques from particular areas were finally used.

3.4 Step 4—Validation

Step 4 focused on validation—finding out whether the solutions developed using guidelines from recognized sources are useful in practice from the business analyst's point of view. Interviews were chosen as the method of validation. Two analysts from the previous group of 8 were contacted. They were among the most active participants who contributed the highest numbers of problems. Also, they worked for different companies (A and B) and held different positions (A—senior analyst, B—junior analyst). Validation interviews were arranged separately with each analyst. During the interview each of 28 considered problems was discussed using the following scheme:

1. The researcher asked the interviewed analyst to come up with proposal of solutions to a given problem.
2. The researcher revealed his own proposal developed in step 3.
3. A comparison of the proposals by both participants took place, followed by a discussion to reach a consensus.

In some cases the proposals of the researcher and the interviewee were exactly the same, so no discussion was necessary, but mostly there were at least partial differences. Incidentally, the interviewee admitted he had no idea which technique to apply for a given problem (literally 2 cases). The outcome of the discussion could be either:

1. The interviewee admitted that the proposals of the researcher are a better solution (or at least not worse—quite often the conclusion was that different techniques can be used as equivalent solutions).
2. The interviewee convinced the researcher that his proposal should be changed or at least extended by applying additional technique.

The researcher took detailed notes documenting the discussions and afterwards summarized the outcomes for each of the problems. An example is presented below.

Name of the problem: *Lack of stakeholders' commitment.*

Author: *Two different solutions can be applied. A more "friendly" approach is to facilitate* ***workshops*** *or* ***brainstorming*** *and therefore to stimulate the stakeholders to be more active and creative in requirements elicitation. These techniques were selected, because they are generally known to stimulate creativity and involve participants. An alternative approach results in a more confrontational way and includes usage of the* ***stakeholders map*** *technique. The map enables the analyst to understand the organizational hierarchy of a company or project and to contact a superior of an uncooperative stakeholders who can deal with them or find a replacement.*

Analyst A: *The first analyst was inclined to apply* ***workshops*** *or* ***brainstorming*** *with an emphasis on choosing the ones with a more attractive form. He said that an approach which includes a creative way of eliciting requirements and is considered fun would be more profitable than a standard "boring" meeting. During the discussion, the researcher presented his solutions including the "unpleasant way" with using a* ***stakeholders map****, but the analyst disagreed with that approach.*

Analyst B: *The second analyst also proposed* ***workshops*** *and* ***brainstorming****, but he also suggested using* ***prototypes*** *as a way to capture stakeholders' attention and as result effectively elicit requirements. After hearing researchers' proposals, the analyst agreed that they are suitable.*

Conclusion: *In all 3 cases, the preferable solution was to stimulate stakeholders' initiative by using creativity-based techniques like* ***workshops****,* ***brainstorming*** *and* ***prototyping****. The researcher decided to add* ***prototyping*** *to the short list of suitable solutions to this problem. On the other hand, the idea of using* ***stakeholders map*** *was discarded as a result of validation. The researcher was convinced by the argument of Analyst A, that it could result in a negative attitude and harm relationships between the project team and the stakeholders.*

Depending on the outcomes of the interviews, the following course of action could be taken to incorporate validation results into the proposed set of solutions:

1. No change—validation confirmed that the proposal is sound, no counter-proposals were issued by the interviewees (12 problems).
2. Extension—another technique was added as part of the solution, especially if it reinforced the techniques already included in the solution (14 problems).
3. Alteration—the initial solution proposal was modified by substituting one or more techniques with others, as suggested by one or both interviewees (2 problems).

All the techniques proposed by the interviewees and included as extensions or alterations could be found in the unified set based on 3 certification sources, there was no case that would require modification of the outcome of step 1. The example results of validated solutions to problems are included in Table 4. The table also

Table 4 Techniques assigned as solutions (examples)

Problem	Techniques assigned (after validation)
Analyzing undocumented existing system	Document analysis, observation, "Stakeholders" section of SRS
Changing requirements	Requirement diagram, cost-value prioritization, prototyping
Failure to identify an essential stakeholder	Stakeholder map, RACI matrix, Process modeling
Lack of authorized stakeholders (capable of making decisions)	Stakeholder map, RACI matrix
Lack of stakeholders' commitment	Stakeholder map, brainstorming, workshop, Prototyping
Low quality of specified requirements (e.g. incomplete, too generic)	Non-functional requirements analysis, user stories, scenarios, process modeling
The stakeholders completely do not know what they want	Brainstorming, interviews, workshop
The stakeholders express requirements which are outside system's scope	Scope modeling, process modeling
The stakeholders are unable to express their needs/requirements	Observation
Too short deadlines to complete BA	Timeboxing, MoSCoW prioritization

shows changes resulting from validation—the techniques added to the initial proposal and removed from it are distinguished by underline and strikethrough respectively.

4 Conclusions and Further Work

The review of the state of the art RE/BA knowledge sources resulted in a large set of recommended BA techniques. A side effect of this work is the observation that apart from the "core" established and well known techniques (like prototyping, questionnaires or observations), the reviewed industrial standards recommend different techniques as tools for business analysts.

The study uncovered a number of problems expressed by a group of business analysts and related to their work activities. For each of frequent problems a solution in the form of one or more BA techniques was proposed. Validation shows that the set of BA techniques developed by unifying contents of 3 selected sources was sufficient to address each of the problems considered. Moreover, the guidelines on applicability of particular techniques compiled from 3 sources allowed to select the right techniques (only 2 out of 28 proposals were rejected by business analysts participating in validation). The substantial number of proposals which were extended with additional techniques as result of validation suggest that the set of available techniques includes many items which can be used interchangeably, as replacements for each other.

The main threat to the validity of the results is a relatively small group of study participants and the fact that they all come from Polish software industry. The reported small-scale study (conducted as part of the M.Sc. thesis [26]) was designed from the beginning to target Polish IT sector. We do not make any assumptions whether conditions of business analyst's work in Poland are significantly different than elsewhere or not—we simply report our results. Within the scope of our study, we made some effort to include representative participants (different companies and profiles of BA practitioners). Another potential threat is the subjectivity of assessments made by interviewees about problems' frequencies and applicability of solutions, however subjectivity is an integral part of the selected survey approach.

As for future work, we consider reviewing additional sources e.g. SWEBOK or PMI Guide—even if the current set of techniques seems to be "sufficient", more guidelines and hints about which one to apply would be helpful (especially in case of similar techniques). Also, a study involving a larger number of participants and companies (preferably from different countries) is a possible direction of research. As our approach of asking open questions about problems proved to have its drawbacks (low similarity of problems reported), it is worth to consider using a combined approach—a list of problems (based on literature analysis) available to the participant, together with the opportunity to add problems not present on the list.

References

1. The Standish Group International: Chaos Report 2014 (2014)
2. Charette, R.N.: Why software fails. IEEE Spectrum **42**(9), 42–49 (2005)
3. Wiegers, K., Beatty, J.: Software, 1st edn. Microsoft Press (2013)
4. Chrabski, B., Zmitrowicz, K.: Inżynieria Wymagań w Praktyce (in Polish—Requirements Engineering in Practice). Wydawnictwo Naukowe PWN, Warsaw (2015)
5. Neill, C., Laplante, P.: Requirements engineering: the state of the practice. IEEE Softw. **20**(6), 40–45 (2003)
6. Sommerville, I., Ransom, J.: An empirical study of industrial requirements engineering process assessment and improvement. ACM Trans. Softw. Eng. Methodol. (TOSEM) **14**(1), 85–117 (2005)
7. Davey, B., Parker, K.: Requirements elicitation problems: a literature analysis. Issues Inf. Sci. Inf. Technol. **12**, 71–82 (2015)
8. Firesmith, D.: Common requirements problems, their negative consequences and the industry best practices to help solve them. J. Object Technol. **6**(1), 17–33 (2007)
9. Solemon, B., Sahibuddin, S., Ghani, A.: Requirements engineering problems and practices in software companies: an industrial survey. In: Ślęzak, D., Kim, T.H., Kiumi, A., Jiang, T., Verner, J., Abrahão, S. (eds.) Advances in Software Engineering, Communications in Computer and Information Science, vol. 59, pp. 70–77. Springer, Heidelberg (2009)
10. Bobkowska, A., Wyrzykowski, K.: Model Działania Analityka Biznesowego w Administracji Publicznej w Celu Przeciwdziałania Ukrytym Wymaganiom. In: Proceedings of 7th TIAPISZ Conference (Technologie informatyczne w administracji publicznej i służbie zdrowia), Warsaw (2015)

11. Cheng, B., Atlee, J.: Research directions in requirements engineering. In: International Conference on Software Engineering (ICSE'07), IEEE Computer Society, pp. 285–303. IEEE Computer Society, Washington DC (2007)
12. Yousuf, M., Asger, M.: Comparison of various requirements elicitation techniques. Int. J. Comput. Appl. **116**(4) (2015)
13. Aoyama, M., Nakatani, T., Saito, S., Suzuki, M., Fujita, K., Nakazaki, H., Suzuki, R.: A model and architecture of REBOK (requirements engineering body of knowledge) and its evaluation. In: Proceedings of 17th Asia Pacific Software Engineering Conference. IEEE (2010)
14. Besrour, S., Bin Ab Rahim, L., Dominic, P.: Assessment and evaluation of requirements elicitation techniques using analysis determination requirements framework. In: 2014 International Conference on Computer and Information Sciences, pp. 1–6 (2014)
15. Wellsandt, S., Hribernik, K., Thoben, K.: Qualitative comparison of requirements elicitation techniques that are used to collect feedback information about product use. In: Proceedings of 24th CIRP Design Conference, pp. 212–217 (2014)
16. Vestola, M.: A Comparison of Nine Basic Techniques for Requirements Prioritization. Helsinki University of Technology (2010)
17. ISO/IEC/IEEE Standard 29148-2011. Systems and Software Engineering—Life Cycle Processes—Requirements Engineering (2011)
18. IEEE Standard 830-1998. IEEE Recommended Practice for Software Requirements Specifications (1998)
19. IEEE Standard 1233-1998. IEEE Guide for Developing System Requirements Specifications (1998)
20. International Institute of Business Analysis: A Guide to the Business Analysis Body of Knowledge (BABOK Guide) v2.0 (2009)
21. International Requirements Engineering Board: IREB CPRE Foundation Level Syllabus ver. 2.2 (2015)
22. International Requirements Engineering Board: IREB CPRE: Elicitation and Consolidation, Advanced Level Syllabus ver. 1.0 (2012)
23. Requirements Engineering Qualifications Board: REQB CPRE Foundation Level Syllabus ver. 2.1 (2014)
24. Requirements Engineering Qualifications Board: REQB CPRE Advanced Level Requirements Manager ver. 1.0 (2011)
25. IEEE: A Guide to Software Engineering Body of Knowledge 3.0 (2014)
26. Marciniak, P.: The Role of Business Analyst in IT Companies. M.Sc. thesis, Gdańsk University of Technology (2015)

Beyond Software Architecture Knowledge Management Tools

Andrzej Zalewski

Abstract Research into architecture decision-making tools has so far been focused on validating the research concepts underlying architectural knowledge management in a practical context, rather than on delivering tools that are suitable for conditions of the real-world software development. At the same time, much research effort has been devoted to the exploration of architects' expectations with regard to architecture decision-making tools. The main idea promoted here is that instead of separate architecture knowledge management tools we need an architecture knowledge management infrastructure that connects producers and consumers of architectural knowledge and easily integrates with tools they use for software development. It should be designed so that it seamlessly integrates with the software development environment. The study of the architecture of such a system reveals important challenges facing academia and the software industry with respect to the popularisation of architecture knowledge management.

Keywords Architectural knowledge management • Software architecture decisions • Architecture decision making tools

1 Introduction

The recent survey by Capilla et al. [1] shows that the impressive research on software architectural knowledge (AK) and architecture decision (AD) making [2] has not resulted in a broad adoption of AK management practices by the software industry. The factors that hinder the wider industrial application of AK management can be traced back to the complicated nature and properties of AK management and AD making [3], to the lack of a convincing business case for including AK management as a day-to-day practice, to the reluctance of the participants of the

A. Zalewski (✉)
Institute of Automatic Control and Computational Engineering, Warsaw University of Technology, Warsaw, Poland
e-mail: a.zalewski@elka.pw.edu.pl

L. Madeyski et al. (eds.), *Software Engineering: Challenges and Solutions*,
Advances in Intelligent Systems and Computing 504,
DOI 10.1007/978-3-319-43606-7_13

software development process to expand their duties with AK management, or to the lack of adequate tools [1]. All these factors form a vicious circle that has to be broken at some point in order to resolve all the issues.

The lack of appropriate tools that support AK management [1] seems to be of primary importance because it strengthens the other factors itemised above. Lack of adequate tools makes it generally difficult to include AK management into software development processes, increases the cost of AK management, hinders to observe the benefits resulting from AK management as well as makes developers reluctant to apply practices, which they perceive as immature for the industrial use due to lack of tool support.

The primary idea promoted in this paper is that in order to introduce AK management into real-world software development organisation, we need to build an entire AK management infrastructure, which enables flow of AK between producers and consumers, and at the same time seamlessly integrates with the various tools supporting real-world software development.

A top-level architecture of such an AK management system has been envisaged in this paper. Although, the architecture is presented and discussed on rather a high abstraction level, it reveals the challenges that have to be met both by academia and the software industry in order to make AK management a common software development practice, similar to requirements analysis or testing, for example.

The rest of the paper is organised as follows: Sect. 2 presents shortly related work, the set of architectural drivers is presented in Sect. 3, Sect. 4 describes the envisaged architecture of the AK management system, which is discussed against the background of related work in Sect. 5, Sect. 6 provides a summary and outlines further research.

2 Related Work

The research on AK management tools has been summarised in the book by Ali Babar et al. [2] and papers by Capilla et al. [1], Tang et al. [4], Manteuffel et al. [5], Shahin et al. [6]. The tools developed during recent 10 years are either stand-alone systems, such as Archium, PAKME, ADDSS, Eagle, Adkwik, ADDM, ADDSS v. 2.0/2.1, SAW, ADvISE and RGT or plugins into: modelling tool (AREL, Decision Architect), software development kit (SEURAT) or popular office software (Knowledge Architect).

The engagement of ABB company in the development and validation of Decision Architect [4, 7] clearly marks the shift from tools developed in order to provide a test-bed for AK management concepts to AK management systems that are suitable for industrial applications.

3 Architectural Drivers for an Industrial Architecture Knowledge Management System

The drivers that shape the architecture of an AK Management System that can be easily accepted in the software industry reflect the important facets of modern software development practices.

A. AK management systems should support a variety of implementation frameworks, technologies and modelling tools etc.

A variety of tools are used in modern software development projects, namely, modelling/architecting tools, software development kits, test support tools, requirements management tools (compare [8] for example) etc. It is also remarkable that in large developments many software development technologies as well as programming frameworks and libraries are employed. This makes real-world software development environments heterogeneous in terms of the applied software technologies and tools.

"Software architecture is an abstraction of a software system, which is perfectly suitable for communicating, reasoning, and learning about important system elements and properties" [9], therefore, software architecture is perceived as a communication medium between the project stakeholders. This role should be supported by the AK management tools, which should provide a medium for the exchange of AK between producers and consumers, who may be using many different tools and implementation technologies.

B. Architects are neither the only producers nor the only consumers of AK

Although architects usually play an important role in the creation and consumption of AK, they are obviously not the only stakeholders participating in these activities. van Heesch et al. [10] provide a list of stakeholders who have concerns related to ADs. Apart from architects, the list also includes reviewers, managers, customers, requirements engineers, new project members and domain experts who are in generally consumers of AK knowledge. Based on real world observations, I would argue that the list of producers and consumers of AK is much longer (an asterisk indicates producers and consumers of AK added to the above list in this paper):

- Producers: architects, developers*, vendors of software development frameworks/technologies/libraries*;
- Consumers: architects, requirements engineers, developers*, quality assurance team members (testers, reviewers), various managers (change, development, top-level), clients, new project members, domain experts.

While consumers generally use AK for their own purposes (e.g. QA team members use it for architecture assessment or developing test cases, while new members need AK in order to get acquainted with the architecture of a system being developed etc.), the list of producers requires more attention. In many development projects there are no architects at all, and architectural decisions are made at the coding level by the team of programmers. They focus on producing design knowledge (compare categories of AK in Tang et al. [4]), though they must also make architectural decisions, and hence, produce reasoning AK that usually remains tacit.

If an architect is involved in the project, developers can also use reasoning knowledge in order to consult their choices with the architecture established by the architects. On the other hand, architects and QA need design knowledge to perform architecture assessment and to verify that the developed software is consistent with the crafted architecture.

C. **AK is not created entirely during the development projects, but also brought in with the chosen programming frameworks/technologies/libraries/component sets**

Modern software is extremely rarely developed from scratch. Software development starts nowadays from configuring the development landscape by choosing software development kits (not necessary a single one!), programming frameworks, libraries or sets of the components. Such crucial choices belong to the reasoning AK, but substantial amounts of architectural knowledge are also contained in the chosen programming frameworks, libraries or components. Why should we capture this AK again from scratch? Architectural knowledge could be captured while developing programming frameworks and delivered as a predefined AK. It could take, for example, the form of a set of templates of architectural decisions comprising all the architectural choices available in a given framework.

D. **AK management system has to scale well between development organisations of different sizes**

In order to make AK management a common industrial practice, it should become available and affordable for every development organisation. This means that AK management systems should be highly versatile, easy to deploy and to integrate into the tools used by a given development organisation. It should also be easy to scale the scope of AK system's deployment to the specific conditions of a given software development organisation. This implies a highly modular structure with plug-ins enabling the AK capability in the existing software development and modelling tools.

E. **AK management system is distributed**

Software development is an activity distributed among various roles and various people, more and more often between various locations. The role of AK management systems is to integrate these distributed activities.

4 The Architecture of Architecture Knowledge Management Infrastructure

The proposed AK management system (Fig. 1) should comprise the following architectural elements:

- Modelling tool plug-in—enables architectural decision making functionality similar to, for example, Decision Architect or AREL, integrated with the regular functionality of a modelling tool (e.g. UML diagram drawing). It should also support downloading the predefined architectural knowledge supplied with development frameworks, libraries etc. into the modelling tool for further processing (e.g. capturing design or reasoning knowledge). On the other hand, it allows AK stored in other modelling tools and software development kit (SDK) to be accessed and searched.
- Software Development Environment (SDE) plug-in—enables the developers to: (1) document AK and facilitates the use of the predefined AK delivered together with software development frameworks, ready components or libraries, (2) access and search through AK stored in the same SDE or elsewhere.
- AK broker—the core of the entire architecture, enables the flow and exchange of AK between the entities of a system. Its role is also to impose AK exchange standards on other architectural elements.
- AK stand-alone browser—it allows to look into the AK stored in the system without a modelling tool or SDE. This may be useful for testers, clients and team newcomers.

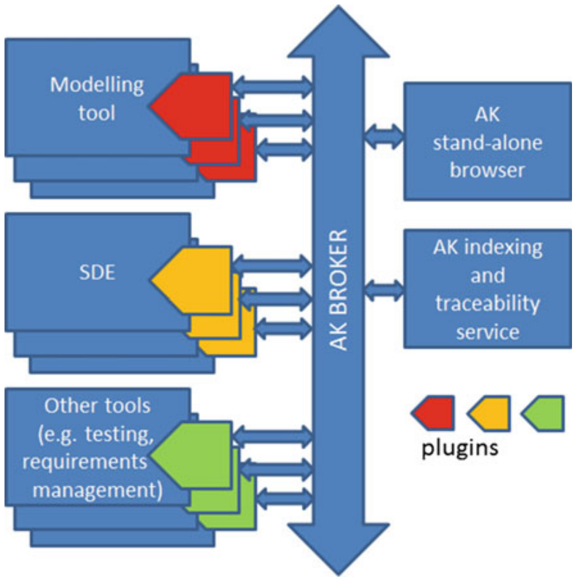

Modelling tool
SDE
Other tools (e.g. testing, requirements management)
AK BROKER
AK stand-alone browser
AK indexing and traceability service
plugins

Fig. 1 The proposed architecture for AK management system

- AK indexing and traceability server—it plays a role of a search engine and enables traceability links to be established and explored between various AK elements and design artefacts. It would turn out to be necessary in large software development environments, in which searching through multiple, distributed stores of AK would impede the performance of the AK management system.
- Plug-ins to other systems employed to support software development (testing management, requirements management systems, for example)—as they are used by the consumers of AK they should enable to look into the AK, directly from the tools, into which plugin is installed.

AK will actually be distributed among the repositories of modelling tools and software development kits. As a result, AK is stored near the place, in which it is created. Distributed AK is accessed via an AK broker, which can also be named the Architectural Knowledge Bus.

The following configurations of the above architecture are imaginable:

- Small development organisation, single development team—installs only a plug-in into the SDK and uses its features to capture AK, mainly by exploiting the predefined AK delivered with the chosen programming frameworks, COTS or libraries;
- Medium-size development organisation, e.g. small software houses—install plug-ins into the SDKs that they use, install plug-ins into modelling tools and the AK broker;
- Large development organisations—install all the components of the AK management system.

5 Discussion

The presented architecture follows the well-known system integration pattern—broker (bus). It also consumes the well-established ideas of AK management plugins and stand-alone AK management tools (see: Sect. 2). The novel idea is that we should go beyond a single AK management tool by developing an entire AK management infrastructure, which connects various tools, which support software development, in order to ensure AK flow between producers and consumers. The novelty is also the concept of predefined architectural knowledge, which should be delivered (and even sold) together with the software development frameworks, software libraries etc.

Let us look how the envisaged system addresses the barriers to industrial adoption of AK listed by Capilla et al. [1]:

- “B1. Lack of motivation”—this has been addressed to some extent—a system compatible with the existing software development environment can encourage the introduction of the AK management on an organisation scale;

- "B2. Lack of adequate tools"—naturally, the success of the AK management system depends hugely on its detailed functionality. Here, only issues connected with the integration with established tools and practices are addressed. Let us observe that integration between heterogeneous tools requires developing standards for documentation and the exchange of AK knowledge;
- "B3. What to capture?", "B4. Effort in capturing AK", "B5. Disrupting the design flow"—using the predefined AK should simplify the capture of AK in comparison with a "from scratch" approach. This should minimise additional effort and clarify what should be documented and what should not. However, this raises another challenge, as such a knowledge base has to be developed and delivered by the vendors or developers of implementation frameworks etc.
- "B6. Lack of stakeholder understanding"—this is unfortunately out of the range of any AK management tool.

Let us observe that the proposed AK knowledge management support follows three out of the four recommendations found in Capilla et al. [1]:

- "R1. Encourage and convince stakeholders"—compare barriers B1, B4 below.
- "R3. Lean approaches for capturing AK"—the use of the predefined AK knowledge will substantially speed up AD making, by providing templates of AD comprising at least definitions of typical problems and available solutions, together with recommendations for which option to choose in certain conditions;
- "R4. Embed the design rationale with current modelling approaches and tools"—this is achieved through plugins into SDKs and modelling tools.

In order to develop the AK management system for which the architecture has been sketched in this paper, the following changes has to be met:

- Developing a standard for AK storage, retrieval and exchange, the challenge is that it should support a variety of forms of AK representation. This would lead to either a generic solution, which would impede developing tools exploiting the details of AK elements, or to specialised, varied solutions that cannot be integrated;
- Making the above standard accepted by the industry, which is probably even more challenging, as there currently exist many competing AD representation concepts [11];
- Modelling tools and SDK should be ready for AK plug-ins, which fortunately is becoming common (compare, for example, leading modelling tools—Enterprise Architect, IBM Rational, Eclipse);
- Participation of commercial industry and open source development communities in capturing the predefined AK included in the development frameworks etc. has to be ensured.

The above challenges fall under the category of managerial and organisational problems. They precede many technical and theoretical problems that will have to be resolved when standards of AK documentation and exchange are established and accepted, and the support of the industry is assured. The list of such problems includes: ensuring support for architectural decision making, defining mechanism enabling search through AK distributed among system's nodes, defining the scope and form, in which predefined AK is delivered as well as mechanisms enabling its further use, technical details of AK capturing mechanisms included in SDK and modelling tool plugins.

6 Conclusion, Research and Development Outlook

The architecture of an AK management system suitable for software industry has been envisaged. It turned out to follow the well-known systems integration pattern, namely, a broker or service bus ensuring the flow of AK between producers and consumers.

The proposed architecture is aimed at supporting the capture of AK at the level of both architectural and detailed (code) design as well as the consumption of AK by various stakeholders. The AK may be presented to the consumers, traditionally, using textual representation or visualisation techniques such as those presented in [11] or for example the diagrammatic notation such as MAD proposed in [12, 13]. This is an important step, as architects are usually kept away from the detailed design developed on the source coding level. The observation that important parts of AK are brought in with the choice of the software development frameworks, if properly exploited, may enable the development of tools that substantially reduce the amount of work necessary to document AK.

Finally, the success of AK management systems requires tight and effective collaboration between academia and industry in developing AK representation and exchange standards, as well as the components of the envisaged AK management system. This seems to be a challenge for the next 10 years of AK knowledge management.

References

1. Capilla, R., Jansen, A., Tang, A., Avgeriou, P., Ali Babar, M.: 10 years of software architecture knowledge management: practice and future. J. Syst. Softw. (Available on line 2015) (in press)
2. Ali Babar, M., Dingsøyr, T., Lago, P., van Vliet, H.: Architecture knowledge management. Theory and Practice. Springer, Berlin, Heidelberg (2009)
3. Zalewski, A., Kijas, S.: Architecture decision-making in support of complexity control. In: Software Architecture; 4th European Conference, ECSA 2010, Copenhagen, Denmark, 23–26 Aug 2010, LNCS, vol. 6285, pp. 501–504 (2010)
4. Tang, A., Avgeriou, P., Jansen, A., Capilla, R., Ali Babar, M.: A comparative study of architecture knowledge management tools. J. Syst. Softw. **83**(3), 352–370 (2010)
5. Manteuffel, Ch., Tofan, D., Avgeriou, P., Koziolek, H., Goldschmidt, T.: Decision architect—a decision documentation tool for industry. J. Syst. Softw. **112**, 181–198 (2016)

6. Shahin, M., Peng, L., Khayyambashi, M.R.: Architectural design decision: existing models and tools. In: Joint Working IEEE/IFIP Conference on Software Architecture, 2009 and European Conference on Software Architecture. WICSA/ECSA 2009, pp. 293–296. IEEE (2009)
7. Manteuffel, C., Tofan, D., Koziolek, H., Goldschmidt, T., Avgeriou, P.: Industrial implementation of a documentation framework for architectural decisions. In: Software Architecture (WICSA), 2014 IEEE/IFIP Conference on, pp. 225–234 (2014)
8. Jansen, A., de Vries, T., Avgeriou, P., van Veelen, M.: Sharing the architectural knowledge of quantitative analysis. In: QoSA 2008, LNCS 5281, pp. 220–234. Springer (2008)
9. Weinreich, R., Buchgeher, G.: Towards supporting the software architecture life cycle. J. Syst. Softw. **85**(3), 546–561 (2012)
10. van Heesch, U., Avgeriou, P., Hilliard, R.: A documentation framework for architecture decisions. J. Syst. Softw. **85**(4), 795–820 (2012)
11. Shahin, M., Liang, P., Ali Babar, M.: A systematic review of software architecture visualization techniques. J. Syst. Softw. **94**, 161–185 (2014)
12. Zalewski, A., Kijas, S., Sokołowska, D.: Capturing architecture evolution with maps of architectural decisions 2.0. In: ECSA 2011, Essen, Germany, 13–16 Sept 2011. Lecture Notes in Computer Science, vol. 6903, pp. 83–96 (2011)
13. Zalewski, A., Ludzia, M.: Diagrammatic modeling of architectural decisions. In: Software Architecture, Second European Conference, ECSA 2008 Paphos, Cyprus, 29 Sept–1 Oct, Proceedings. Lecture Notes in Computer Science, vol. 5292, pp. 350–353 (2008)

Model of RUP Processes Maturity Assessment in IT Organizations

Włodzimierz Wysocki, Cezary Orłowski, Artur Ziółkowski and Grzegorz Bocewicz

Abstract A crucial problem in modern software engineering is maturity assessment of organizations developing software. We propose utilizing a process model of RUP development methodology, as a pattern for comparing it with the examined process. Percent values of accordance coefficient determine the task accordance of the examined process with the pattern of activities flow. The above mentioned RUP model concept is based on a multi-agent based simulation (MABS). It presents goals and behaviors of agents as well as components of the agent system environment. To confirm the usefulness of the method for assessment of organization's maturity, a two-fold experiment was undertaken. The results confirm the usefulness of the model in maturity assessment. First part consisted of tuning the simulation internal parameters to the development process. In the second part accordance coefficient values of the sample project were obtained by comparing it with the results of the simulated model.

Keywords IT organization maturity • Software development processes • Rational unified process • Maturity assessment • Software process simulation • Multi-agent • MABS • JADE

W. Wysocki (✉) · G. Bocewicz
Technical University of Koszalin, Koszalin, Poland
e-mail: wlodzimierz.wysocki@tu.koszalin.pl; wlodek.wysocki@gmail.com

G. Bocewicz
e-mail: bocewicz@ie.tu.koszalin.pl

C. Orłowski (✉) · A. Ziółkowski (✉)
WSB University, Gdańsk, Poland
e-mail: corlowski@wsb.gda.pl

A. Ziółkowski
e-mail: Artur.Ziolkowski@wsb.gda.pl; aziolkowski@wsb.gda.pl

L. Madeyski et al. (eds.), *Software Engineering: Challenges and Solutions*,
Advances in Intelligent Systems and Computing 504,
DOI 10.1007/978-3-319-43606-7_14

1 Introduction

In recent years we have seen a rise in interest regarding transformation problems of IT organizations from the traditional Rational Unified Processes RUP [5], to agile methodologies, such as SCRUM [9], XP [1]. Interest in the processes of transformation is the result of a large number of failed projects implemented by these organizations. Hence, IT organizations are looking for solutions to ensure the effectiveness of agile transformation. Conducted research of transformation processes is focused on the analysis of transformation paths, (technology, process, design and organizational culture) and the methods used to determine an organization's readiness for starting the process of agile transformation [6–8]. In the latter case, it is assumed that the achievement of the organization's particular state of readiness is a necessary condition for the start of the transformation process. Analysis of the state of preparedness is the challenge for the authors of this article.

This analysis considers the relationship between the maturity of the development processes and its impact on the readiness state of organizations for agile transformation. The use of an agent-object system is proposed for analysis of an example RUP development methodology. RUP describes both development and management processes, as well as determines their performance and the utilization in the various phases and iterations of the development process. Process roles are defined as well. Hence, RUP is an excellent example for the description of the development process, as well as a formal description, which can be the basis for its simulation using agent systems.

This integration of RUP and the agent system formed the basis for construction of the agent-object system simulating the workflow of the development team. In the first phase, it was assumed that the research system will be used by project managers to predict scheduling of software development projects. It turned out, however, that in the course of work on the issues of transformation of IT organizations, from the point of view of managers of those IT organizations, examining the state of readiness utilizing the agent-object system is more valuable. Therefore, the authors have proposed to treat the forecasts of the agent-object system as a measure of organizational readiness for the processes of agile transformation.

Because of that, the article is divided into three main parts. In Sect. 2 we presented an agent-object model of the RUP process. In Sect. 3 we present the experiment confirming the usefulness of this method. In Sect. 4 we summarize the results of the experiment and draw conclusions.

2 Agent-Object RUP Process Model

For the analysis of the project and its development processes, compatible with the RUP model, it is proposed to use the agent-object system.

The developed system is based on the model of design environment, which consists of:

- project plan model (based on RUP standards),
- artefact repository,
- development team model.

At the outset, the model was built, and was subjected to the process of implementation using the Jade environment. The simulation model has been designed as a multi-agent system [11]. Members of the project team are mapped as agents performing the tasks assigned to them, in corresponding project roles, according to the project plan and activities flow.

The task involves processing the input artefacts located in the repository into output artefacts and saving them to the repository. Fig. 1 shows the agent-object system model.

Inside the model, dynamic components are represented by the agents placed in an agent environment, containing passive objects to which agents have access and which perform operations that comprise the mapping of the software development process.

Agents can be approximated as a stand-alone computer programs, operating independently of each other, having the ability to communicate among themselves, observing the environment in which they are located, as well as decision-making, and having an impact on the environment. Classes of agents and objects in the system simulation:

Person Agents. Project team model is a collection of person agents

$$G = \{g_i | i = 1 \ldots p\} \tag{1}$$

where the agent is characterized by the following sequence:

$$g_i = (\{r_{i,k} | k = 1 \ldots s\}, \{b_{i,k} | k = 1 \ldots r\}) \tag{2}$$

which defines the role set of an agent, as well as its behaviour set.

Person agent behaviours are supporting:

- searching for types of tasks to be performed, that are appropriate to their assigned roles in the project
- creating and executing tasks based on the searched type of task
- saving information about the work done in the Worktime Register
- cooperation with other agents for joint execution of tasks

Simulator Agent. Controls the execution of the project plan. It is responsible for:

- initializing of the agent environment and the creation of agents
- switching of activities and iterations in the project plan
- storing information about changes to current activities and iterations in Activity Change Register

- ending the simulation, the removal of agents, and storing the course and final state of the simulation

Simulation Clock Agent. It determines the simulation time and synchronizes operation of all agents in the system. One simulation clock cycle represents 15 min of project time. This value has been chosen since the shortest work period intended for the task takes about 15 min.

Project Plan Object. This is the model input data. Information stored in the Project Plan was initially discussed in the description of the inputs and outputs of the simulation model.

Project plan model is understood as a set of iterations performed during the project:

$$PP = \{I_i | i = 1 \ldots n\} \tag{3}$$

where: I_i—ith iteration of project plan PP

According to RUP terminology, iteration I_i consist of activities $O_{i,k}$ defined as subsets of elementary tasks E:

$$I_i = \{O_{i,k} | k = 1 \ldots m\} \tag{4}$$

where: $O_{i,k} \subset E$

Elementary task set E contains base (indivisible) operations which can be implemented in a given development environment, i.e. Find Actors And Use Cases. Task $e_l \in E$ is described by a sequence: $e_l = (r_l, ai_l, ao_l, d_l)$ defining the role of an agent (r_l), which can accomplish the task, a collection of input artefacts required to start the task ($ai_l \subseteq A$), a collection of artefacts obtained after the completion of the task ($ao_l \subseteq A$), the duration of the task (d_l—measured in units of agreed time, u.a.t.).

Development Team Object. This is the model's input described above. Based on information about the members of the project team Agent Simulator creates the agents representing team members.

Artefact Repository Object. It is a part of the model state. The repository is responsible for storing instances of artefacts by type, assigning artefacts unique identifiers, creating specified type of an artefact instance, and search for instances of artefacts according to the specified type.

In the model, the ability to carry out the elementary tasks e_l depends on access to the required artefacts ai_l. The types and number of available artefacts in the system simulation is determined as a pair defined in the repository:

$$R = (A, \varphi) \tag{5}$$

where: A—is a collection of design environment artefacts, $\varphi(a, t) = \varphi_{a,t}$—function specifying the number of type a artefacts $a \in A$ at a discrete point in time t.

Worktime Register Object. It is a part of the process state. The register stores information about which type of a task was performed by the agent, as well as when it began the task and when it finished.

Activity Change Register Object. This is the model output and an element of model state. The register stores information about the changes in the flow of the project. Information about the active iteration of the project, current activities, and simulation time at which the change occurred are stored.

In the context of the above definition, simulation system *SS* can be represented as a sequence containing the project plan *PP* defined on the set of elementary tasks *E*, artefact repository *R* and agent set *G*:

$$SS = (PP, E, R, G) \tag{6}$$

Behaviour of the simulation system is a set of work performed during the project simulation:

$$S = \{W_i^S | i = 1 \ldots n\} \tag{7}$$

where: W_i^S—*i*th task of simulation *S*

The simulation is related to the schedule for completion of elementary tasks:

$$X^s = \left\{x_{i,j,k} | i = 1 \ldots n, j = 1 \ldots m, k = 1 \ldots z\right\} \tag{8}$$

where: $x_{i,j,k}$—and start time of the *i*th task, in regard to activity *j*th and elementary task *k*th

Based on the schedule of elementary tasks, a schedule of activities can be defined, which is used as a process pattern to examine the accordance of the actual design process with the RUP model:

$$\overline{X^S} = \{x_{i,j} | i = 1 \ldots n, j = 1 \ldots m\} \tag{9}$$

where: $x_{i,j} = min\left\{x_{i,j,1} \ldots x_{i,j,z}\right\}$.

2.1 RUP Process Implementation in a Agent-Object System

Fig. 2 shows utilization of agent-object system as a simulation of RUP. Two sets of information are passed as input of the model, according to RUP. The first relates to the project team, the second, project plan.

Each person has a specific name, assigned design roles and periods of employment in the project. The period of employment is made up of a start date, end date, and weekly hours of work that person performs for the project.

The project plan *PP* includes a range of information and the iterations of each phase of the project. The scope of the project is represented as a number of use cases calculated in accordance with the method for estimating the effort of the project in Use Case Points. The size of the project at the input of the simulation is a scaled result of estimating UCP, so that one point corresponds to one Use Case artefact in the system simulation.

For each iteration of the development project, a name, phase and growing percentage targets for disciplines are determined. Iteration also includes the activities workflow of the project. The flow is properly configured and tailored to the needs of the project. The internal parameters of the simulation are responsible for the duration of the different types of tasks.

At the output of the model groups of variables are defined and aggregated to the RUP process state and process pattern $\overline{X^S}$. The state of the simulated project is marked on the diagram so that it can be modified according to the current state of the real project.

This allows the use of simulation to support the decision of the project manager not only at the beginning of the project, but also in the process of software development.

2.2 Implementation of the Agent-Object Model System in a Software Environment

JADE was chosen for the implementation of the agent-object system—it is written in Java as open source code and licensed under GNU. The system is actively developed by Telecom Italia and the international community [2, 4].

Multi-agent system had to be adapted for use as a runtime multi-agent simulation (MABS) [10]. The JADE system lacks the solution for simulation of time. Therefore we use a proprietary solution of the central timing simulation with a fixed step, which is an extension of the concept of clock simulation in the Jadex package [3]. Simulation clock has been extended to determine the order of sending time information to groups of agents, with the introduction of a multi-phase clock cycle. This solution has reduced entropy and produced a better result reproducibility of RUP process simulation.

3 Application of the Agent-Object System to an Actual RUP Project

The aim of the experiment was to use the agent-object system to assess the maturity of the actual development process that is in accordance with RUP.

The experiment consisted of two phases. The first stage was planned to measure the size of the project, preparation of information about completed tasks and tuning parameters of the internal model to the actual process.

Second stage was planned to use the fine-tuned simulation as a pattern to assess the maturity of the development process in the tested process.

The input for the experiment was detailed data on the course of the development process of a large information system in the field of insurance. The system was created as an extension of a previous version, with significant changes in architecture and functionality. The development process lasted for 5 years, the project team employed to design consisted of 30–210 people. The number of tasks performed during the development of the system was over 24,000. The duration of the project simulation was about 40 min.

3.1 The First Stage of the Experiment Using the Agent-Object Model to Assess RUP Process Maturity

The aim of the first stage is to obtain a process pattern matching the RUP process. To do this, size of the project was estimated and internal parameters of the simulation were adjusted.

Estimating the size of the project using the UCP method. Preliminary estimation of the size of the project in order to agree on the scope, time and budget are carried out using the Use Case Points method. The total size of the system is: 6480 UCP. Based on the analysis of documents produced after the meetings and requirement workshops it was estimated that the size of the modernization of the system is 549 UCP. Therefore, most of the requirements of the system were obtained from a previous version of the software and the project size is 5931 UCP.

The total size of the system agrees with this calculated effort, which amounts to 319,130 man-hours. The coefficient calculated from this data is 10.2 h for one UCP point, which is the correct value for experienced project teams.

The result was consistent data provided in Table 1, which complemented with the project plan and information about the project team, facilitated simulation of the examined project.

Measuring the effort of project tasks. Available records of the work done during the project had only text descriptions of the work performed. On their basis categories were introduced, from which it was possible to determine the structure of effort and it allowed for tuning of the internal parameters of the model.

Table 1 Project size input data

Input variable	Value
The size of the project according to the UCP method	6480
The size of the new part of the system according to the UCP method	549
The size of the requirements recovered from a previous version of the project according to UCP method	5931

Table 2 Results of the first part of the experiment

Output variable	Actual value	Simulation result	Relative error (%)
Number of artefacts	15517	14945	3.70
Effort of project	319130	308703	3.30

Fifteen categories of project tasks were introduced and according to them, the structure of labour was determined. With this structure, internal parameters defining the processing times of task types in RUP can be estimated.

Tuning the number of artefacts in the model. The time to complete tasks and simulation effort depends on the number artefacts in input. Therefore, the first step was tuning simulation's internal parameters, in order to get the number of artefacts created in the simulation close to the tested project. There are results provided in Table 2.

The effort of tasks by category. Each category was assigned appropriate activity types. Internal parameters defining the execution time of task types have been selected, in order to simulate the effort close to the real project values. The results are given in Table 3.

Table 3 Effort of tasks by category results

Category	Actual man-hours	Simulated man-hours	Relative error (%)
Architecture project	1240	1274	2.7
Architecture prototype	4593	4474	2.6
System integration	1792	2405	25.5
Consultations	3244	–	–
Configuration management	2579	2699	4.4
Implementation and bug fixing	159419	158780	0.4
Design	7353	7350	0.0
Analytical project	5906	5261	10.9
Database project	4614	4499	2.5
Graphic interface project	3282	2226	32.2
Testing	57738	56891	1.5
Use case model	45377	40618	10.5
Requirements workshops and interviews	6900	13518	49.0
Project management	9573	8708	9.0
Other work	5520	–	–
Total	319130	308703	3.3

3.2 The Second Phase of the Experiment—Utilizing Agent-Object Model to Assess RUP Process Maturity

In the second stage of the experiment we used results of the RUP simulation process as a pattern for comparison with the course of the tested project.

The tested project is defined as a set of task performed during the project:

$$P = \{W_i^P | i = 1 \ldots n\} \tag{10}$$

where: W_i^P—*i*th task of tested project *P*, is defined analogously to the simulation results (7).

The project is related to its schedule:

$$X^P = \{x_{i,k} | i = 1 \ldots n, k = 1 \ldots m\} \tag{11}$$

where: $x_{i,k}$—start time of *i*th task, *k*th activity of project *P*

The accordance of the project task with the simulated task is marked: ≡ and defined as such:

$$W_i^P \equiv W_i^S \; if \; \exists o_i \in O_i^S, \text{for which } (x_{i,j}^P \geq \overline{x_{i,J}^S}) \wedge (x_{i,j}^P \leq \overline{x_{i,J}^S} + d_{i,j}^S) \tag{12}$$

where: O_i^S—activity set, $d_{i,j}^S$—timespan of *i*th task, with regards to *j*th activity in the simulation result.

Activity accordance *Z* is defined as the percentage ratio of the number of tasks in accordance L_z the total number of tasks L_w. There are results provided in Table 4.

$$Z = 100\,\% \frac{L_z}{L_w} \tag{13}$$

Detailed comparison results of the process tested with the process pattern are presented in Fig. 3. The schedule shown in the figure is approximated for readability. The application comparing the test process with the pattern allows to check the cause of non-accordance.

In the figure sample tasks are indicated by a number in the circle. The reasons for the inconsistencies are presented below:

1. At the beginning of the inception phase the task of developing the system architecture was performed, which is against the flow of RUP process.

Table 4 The results of the comparison between tested process and the process pattern

Variable	Description	Value
Lz	Number of tasks in accordance	33396
Lw	Total number of tasks	51701
Z	Activity accordance coefficient	65 %

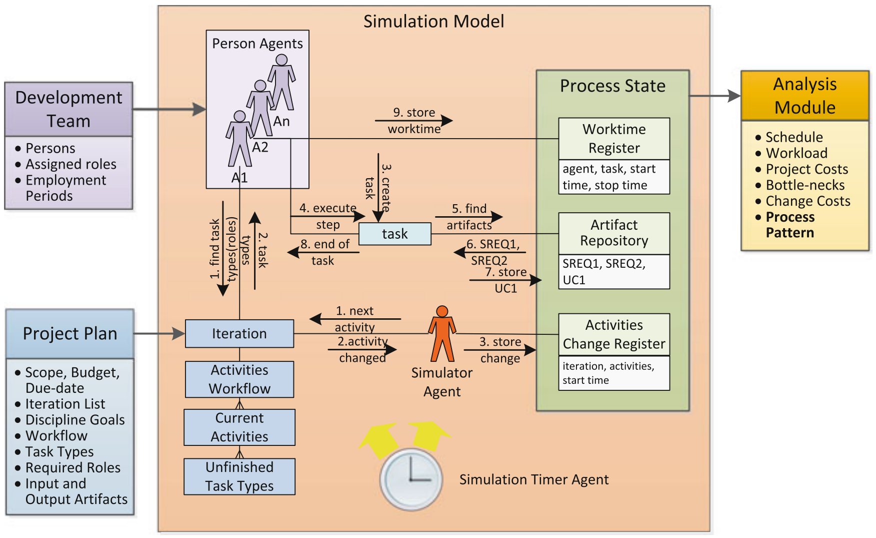


Fig. 1 The agent-object system model for project status analysis of the RUP process

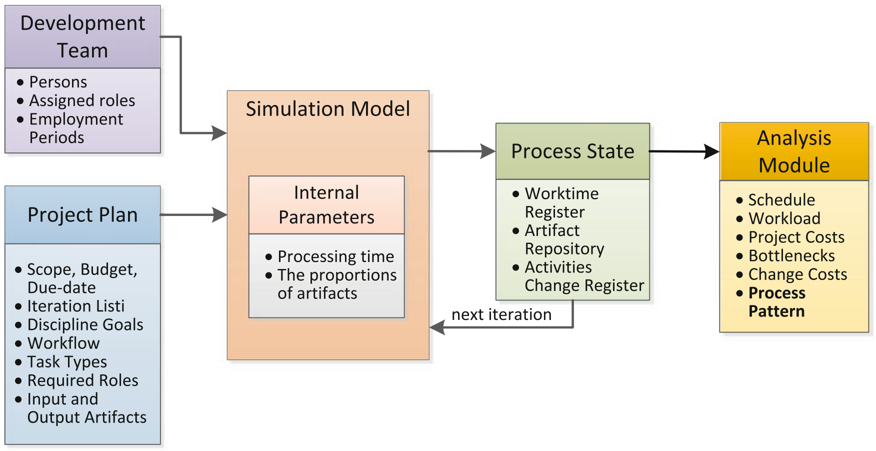


Fig. 2 Utilization of the agent-object system in the RUP process

2. In the latter part of the inception phase the task of designing the database was performed, which is against the flow of RUP process.
3. At the beginning of the first iteration in the construction stage, the task of project management was started, which is consistent with the flow of RUP, but management activities were not performed at this point in simulation.

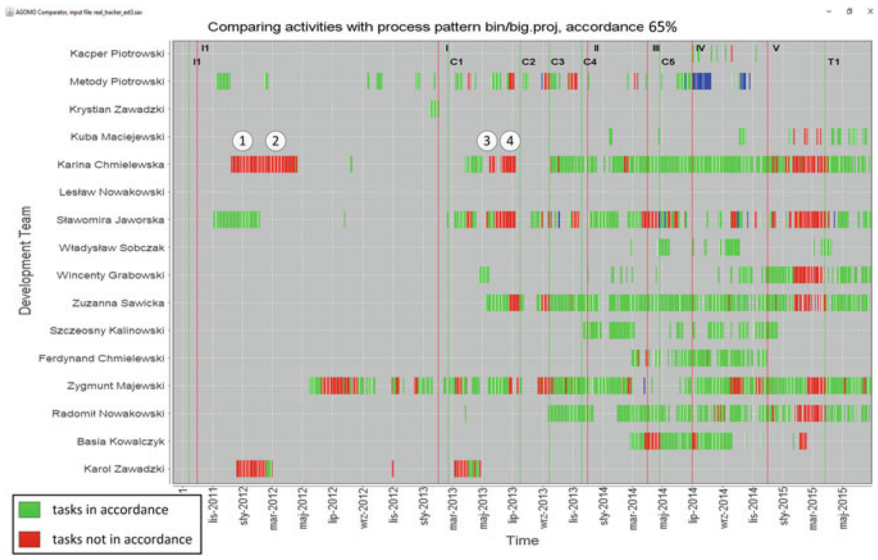


Fig. 3 Magnified fragment of comparison between the actual development process and the result of the RUP process model

4. In a further iteration of the construction the task of design was started, which is in line with the flow of RUP, but management activities were not performed at this time of simulation.

4 Conclusions

The aim of the research was to use the agent-object system developed by the authors to analyse the RUP process in an IT organization. It was assumed, that the use of the system will show the maturity of an IT organization using RUP, for the eventual transformation process, based on the internal parameters of the simulation. The article presents the design of the model, showing its verification environment, as well as the use of the system in the in implementation of an actual IT project. The research experiment, the aim of which was verification of the developed model consisted of two phases. Preliminary, under which the system was used for fine-tuning, and the second, which was used as a pattern for comparison with the course of the test project.

The results in the first phase of the experiment showed that the parameters of the project: the size of the project measured by the UCP method, effort of the project activities, the number of artefacts and effort by category of tasks, has created conditions to tune the system so that the relative error was small. It can therefore be assumed that the proposed internal parameters of the simulation can be used to assess the readiness of the organization of the process agile transformation.

The results of the experiment in the second phase indicated that the development processes ended earlier than the actual ones analysed. Therefore, there is a large discrepancy in activities at the end of the development process.

The agent-object system proposed in the article, allows the analysis of causes, for which investigated task is not in accordance with the process pattern. For example, the non-accordance of tasks at the beginning of the process arises from the fact, that those tasks concern the system architecture design and implementation of the system architecture, which is contrary to the description of the RUP process, since this work must be carried out at the stage of elaboration phase.

The evaluation method of the RUP process using the agent-object system was checked not only at the level of synthetic assessment in the form of coefficient of accordance which shows the overall assessment of RUP process maturity but also allowing for the preparation of detailed analysis

Therefore, in view of further assessment studies of an organization's readiness for agile processes of transformation, it must be considered how parameters typical for a mature organization's performance may provide a basis for classifying it in assessing the readiness of the organization.

The results obtained in the first and second phase of the experiment indicate on one hand, the ability to forecast the state of the project using the tuning of the agent-object system parameters, yielding a small relative forecast error, but on the other do not create the conditions for accurate assessments of RUP processes.

This may mean that since the organization is not able to predict the state of its processes so its maturity is low in synthetic terms.

It may also mean that the adopted internal parameters of the simulation can be a preliminary measure of an organization's readiness for agile transformation, which means that the readiness state of the organization analysed in article is low.

This does not mean, however, that if the organization is not mature, the risk of agile transformation is high. Such research was not conducted by the authors.

They assumed that the maturity analysis for the start of agile transformation should be carried out before the management decides to implement an such a transformation.

References

1. Beck, K., Andres, C.: Extreme Programming Explained: Embrace Change, 2nd edn. Addison-Wesley, Boston, MA (2005)
2. Bellifemine, F.L.: Developing Multi-agent Systems with Jade. John Wiley, Hoboken, NJ (2007)
3. Braubach, L., Pokahr A.: The Jadex Project: Simulation. In Multiagent Systems and Applications, pp. 107–28. Springer (2013)
4. Jade—Java Agent Development Framework. http://jade.tilab.com/. Accessed 21 May 2016

5. Kroll, P., Kruchten, P.: The Rational Unified Process Made Easy? A Practitioner's Guide to the RUP. Addison-Wesley, Boston (2003)
6. Orłowski, C., Deręgowski, T., Kurzawski, M., Ziółkowski, A.: A model for shaping IT project management process to raise the readiness of an IT organization for agile transformation. (in Polish). In: XIX Conference on Innovation in Production Management and Engineering, Zakopane 2016, the Polish Association of Production Management (PTZP) (2016)
7. Orłowski, C., Deręgowski, T., Kurzawski, M., Ossowska, K., Ziółkowski, A.: The use of complexity measures of the project to assess the state of evolution of the IT organization. (in Polish). In: XIX Conference on Innovation in Production Management and Engineering, Zakopane 2016, the Polish Association of Production Management (PTZP) (2016)
8. Ossowska, K., Orłowski, C., Ziółkowski, A., Deręgowski, T., Kurzawski, M.: The aggregate evaluation measure of the organization's state of maturity in the process of its evolution. (in Polish). In: XIX Conference on Innovation in Production Management and Engineering, Zakopane 2016, the Polish Association of Production Management (PTZP) (2016)
9. Schwaber, K., Beedle M.: Agile Software Development with Scrum. Series in Agile Software Development. Prentice Hall, Upper Saddle River (2002)
10. Uhrmacher, A., Weyns D. (eds.): Multi-Agent Systems: Simulation and Applications. Computational Analysis, Synthesis, and Design of Dynamic Models Series. Boca Raton: CRC Press/Taylor and Francis (2009)
11. Wooldridge, M.J.: An Introduction to Multiagent Systems, 2nd edn. Wiley, Chichester (2009)

A Collaborative Approach to Developing a Part-Time Ph.D. Program in IT Architecture with Industry Cooperation

Jan Werewka

Abstract Ph.D. studies seem to become more and more attractive to industry professionals. Government policies in most countries advocate close mutual beneficial cooperation between industry and universities. In case of IT professionals, Ph.D. studies should be focused on strategic domains, advances in software engineering, and software system engineering. Moreover, development of IT architectures is crucial for software companies, and should become a vital area in the program of Ph.D. studies. This paper focuses on establishing a Ph.D. program in computer science in the context of collaboration with IT companies. The paper begins by discussing comparison of different characteristics of part-time and full-time studies. The motivation model includes main stakeholders, drivers, assumptions, principles and assessment methods that are the basis for the proposed Ph.D. project. Possible successful completion of the project is investigated through force field and SWOT analysis. The evaluation of the Ph.D. program takes into account preconditions for applicants, selection of the area of studies and its quality. The paper concludes that some attitude changes at universities and in the IT sector are necessary to carry out the project successfully.

Keywords Ph.D. studies · IT architecture · Software architecture · Computer science · Education

1 Introduction

The paper presents a proposal for Ph.D. studies for IT professionals interested in the topics of IT architecture and development of software systems. The IT architecture concept needs to be considered broadly and includes developing enterprise, business, application, and infrastructure architectures. The paper considers establishing

J. Werewka (✉)
Department of Applied Computer Science, AGH University of Science and Technology, al. Mickiewicza 30, 30-059 Kraków, Poland
e-mail: Jan.Werewka@agh.edu.pl

L. Madeyski et al. (eds.), *Software Engineering: Challenges and Solutions*,
Advances in Intelligent Systems and Computing 504,
DOI 10.1007/978-3-319-43606-7_15

Table 1 Comparison between ITAPhD and RefPhD studies

Feature	ITAPhD	RefPhD
Activities per week	Weekends	Workdays
Ph.D. students as teachers	10 h/Year	60 h/Year
Interaction with taught students	Support, teaching	Regular teaching
Length of the studies	3 years	4 years
Total hours 1st/2nd/3rd/4th pro year	224/224/65/0	555 + (5ECTS)
Student numbers	Min. 20	10 this year
Ph.D. thesis fee	≈150 %	No fee during studies
Language of instruction	English (domestic)	Domestic
Tuition fee for national students per year	100 %/100 %/50 %	0 %

part-time Ph.D. studies focusing on IT Architecture (abbreviated as ITAPhD; IT Architecture Ph.D. studies) and considers the following aspects: (1) How the studies should be organized in order to fit the needs of employees in the IT sector? (2) How to identify Ph.D. studies that are especially relevant for cooperation with industry? (3) How to achieve high quality of the studies? (4) How the cooperation between the industry and the university should be structured to give significant benefits to both sides?

An important question concerns why mainstream computer science Ph.D. studies do not develop IT architecture scientists. An example specialization are Ph.D. studies on trustworthy systems. In [1], the field of trustworthy systems is analyzed, and Ph.D. programs in this area in the US are examined. The discussion considers what is needed to produce competent professionals able to protect health of current cyber systems. This example shows that specialization of Ph.D. studies in computer science is possible and can be important for companies from the IT sector.

To better understand the problem, Table 1 shows a basic comparison between ITAPhD and selected existing full-time Ph.D. programs in computer science (abbreviated as RefPhD; Reference Ph.D.). The area of the referenced studies is described in [2]. The success of ITAPhD will lead to research concentrating on strategic solutions for the IT sectors, and educating industry experts with a scientific background. The fee for Ph.D. studies depends on different factors. In Table 1, it is assumed that the standard fee for a Ph.D. student for the first year is 100 %. This corresponds to a monetary value of €2,000, which is a reference value for further calculations.

2 Related Work

The problem of the suitability of Ph.D. studies was analyzed from different points of view. In the “Bologna Declaration” [3] the European Union (EU) promotes consistency and mobility related to higher education. In the declaration, a framework is proposed for achieving compatibility by introducing a clear division of studies into 3 levels: undergraduate, graduate and doctoral degree studies. The report [4] proposes a set of recommendations to strengthen US graduate (including Ph.D.) education in partnership with industry and the government. The Polish Ministry of Science and Higher Education insists on extensive cooperation between educational establishments and industry, which should be followed by the commercialization of learning outcomes. This cooperation should also be developed in the field of doctoral studies in order to produce experts in the field of IT, and alliances should be fostered with industry to support such cooperation. The Scientific Board of the Polish Information Processing Society published a book [5] on professional IT Qualifications. This work describes a model of proposed solutions for both IT professionals and users of information technology.

Research in paper [6] focused on explaining the elements of the process of developing interest in obtaining a Ph.D. degree in engineering. Certain universities have developed or intend to develop education research centers and Ph.D. programs [7] in engineering basing on the Collaboratory for Engineering Education Research (CLEERhub.org) project with the goal of exchanging information. Another project, ELLEIEC [8] concentrated on Electrical and Information Engineering, and investigated the position of Ph.D. students in Europe with the aim to explain distinctions between academic institution regulations and requirements for obtaining Ph.D. degrees. The goal of the mentioned project was also to clarify important topics [9] aimed at facilitating mobility and exchange programs related to Ph.D. studies in Europe. Another example presents an innovative project supported by the European Union [10]. The project promotes quality research in aeronautics and international Ph.D. education on the base of a collaborative research network. Its participants comprise of leading universities, research centers, and the Air Traffic Management industry. The IT sector requires innovative leaders with robust research and theoretical foundations who can effectively contribute to the global development of economy [11]. The mentioned paper highlights possible gaps in the availability of Ph.D. educational opportunities related to industry specialists working full time and simultaneously attending Ph.D. studies. Domain and also non-domain characteristics [12] of Ph.D. studies are of great importance for industry professionals. This means that part-time Ph.D. studies should be attractive for professionals not only in the area of their research, but also with respect to non-domain features, such as organization efficiency.

3 Importance of the IT Architect Profession

Part-time Ph.D. studies are mostly designated for employees from the IT branch, therefore, they should concentrate on research aspects including new solutions in software system engineering, developing efficient software, or system and enterprise architectures. In IT companies, system architecture design has strategic importance in delivering valuable software products. This brings up the following essential questions: which competences should architects have, and, on the basis of

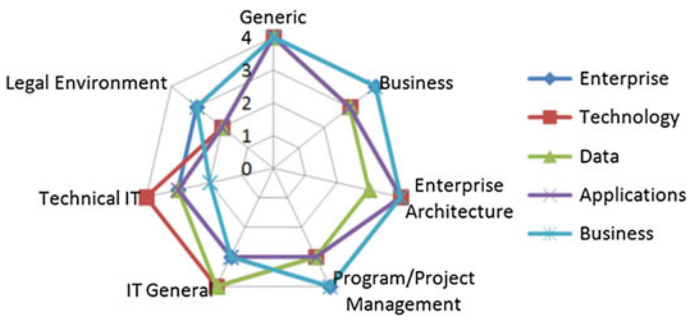


Fig. 1 Averaged IT architects' competence scale

their competences, where should they be located in decision-making structures? The study [13] presents a systematic approach to assessing and developing vital competences that could be leveraged in software companies. Another issue is the definition of the process of career path development, as well as the definition of the organizational structure supporting this process.

In IT companies, the methodologies of development and the assessment of competencies may be built on the basis of their own solutions or existing standards. The normative methodology of the Architecture Skills Framework (ASF), which is part of a broader TOGAF methodology [14], is an example of a standard reference solution. Figure 1 presents the average levels of competencies calculated for the ASF TOGAF set within a category for individual architecture roles.

Universities providing education in the broadly defined field of computer science should investigate the development paths of architects in companies in order to take into account the needs of the industry. The proposed approach [13] was verified at an IT company where it had been implemented by means of the ORRCA architectural governance methodology.

4 Motivation Model of Part-Time Ph.D. Studies

ArchiMate notation [15] has been used for defining a motivation model for the establishment of ITAPhD studies because the elements of the notation seem to be self-explanatory and the notation is not very extensive.

In the first instance, it is important to know the major stakeholders and their drivers (Fig. 2). We can distinguish the main stakeholders and their drivers as follows: IT industry employees who intend to gain scientific and practical knowledge in order to have better competences and employee market value; Ph.D. students who need an effective and well-organized way to access valuable knowledge; university professors who gain additional support for research and feedback from industry; faculties which gain new education qualities and a competitive position in

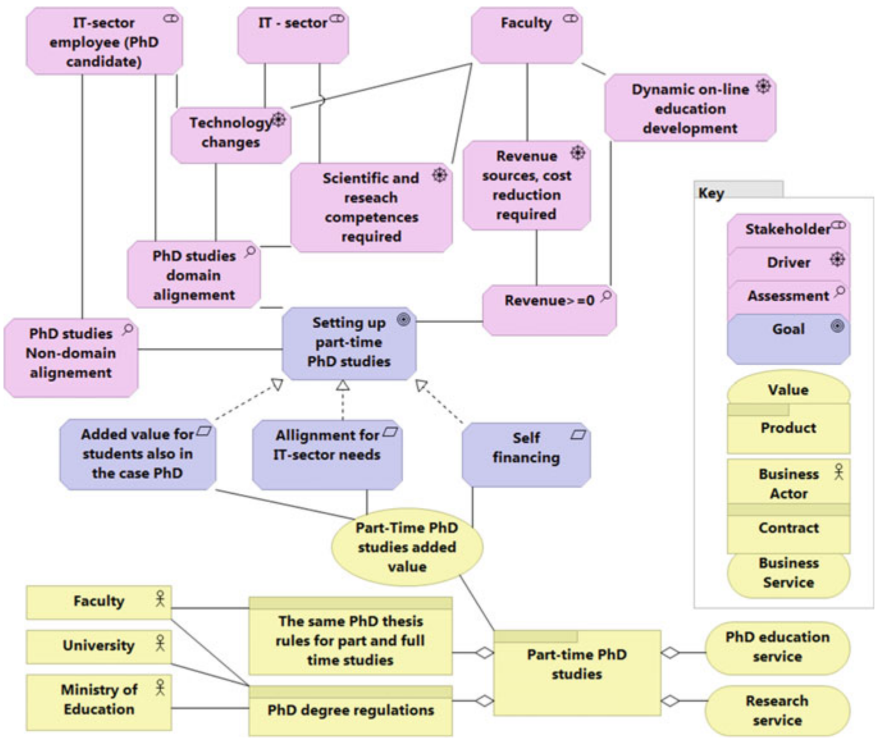

IT-sector employee (PhD candidate)
IT - sector
Faculty
Dynamic on-line education development
Technology changes
Scientific and reseach competences required
Revenue sources, cost reduction required
PhD studies domain alignement
Revenue>=0
Setting up part-time PhD studies
PhD studies Non-domain alignement
Added value for students also in the case PhD
Allignment for IT-sector needs
Self financing
Part-Time PhD studies added value
Faculty
University
Ministry of Education
The same PhD thesis rules for part and full time studies
PhD degree regulations
Part-time PhD studies
PhD education service
Research service
Key
Stakeholder
Driver
Assessment
Goal
Value
Product
Business Actor
Contract
Business Service

Fig. 2 Part-time Ph.D. studies motivation and business layers

relation to IT sector demands; IT companies which gain scientific software experts who can make proper analysis and decisions; IT sector vendors whose technological solutions are promoted by education; the Ministry of Science and Higher Education, which supports co-operation with industry and the commercialization of research results.

A motivation model was developed in ArchiMate (Fig. 2) for the ITAPhD A properly defined motivation model will support the development of the business layer of the proposed Ph.D. studies.

5 Force Field and SWOT Analysis for Offering Ph.D. Studies

For the introduction of a new type of doctoral studies, a force field and SWOT analysis was performed.

5.1 Force Field Analysis

The driving forces are: new Ministry of Science and Higher Education regulations, designed to support co-operation with industry and commercialize research results; the fact that science needs investors from outside the budget sector, while entrepreneurs seek cooperation with scientists; new opportunities; MOOCs technology and e-learning supporting remote work; domestic and foreign competition; the acceleration of change in global IT technology; industrial sector expectations; the creation of additional sources of revenue.

Restraining forces are: resistance to change and the fact that University staff and industry managers are accustomed to old ways; passivity, as university staff and industry managers only concentrate on current problems; the risk of loss of position or image; lack of knowledge of what the competition is doing, the needs of the industry, and the economic situation; the need for financing because projects cannot be self-financing; low IT sector interest.

5.2 SWOT Analysis

SWOT analysis takes into consideration internal strengths and weaknesses, external opportunities, and threats.

Table 2 presents internal strengths and weaknesses and Table 3 shows external opportunities and threats for the proposed Ph.D. study.

In the considered case, the weight of distinguished strengths, weaknesses, opportunities and threats are determined based on discussion with experts. In the next step, influence matrix coefficients are determined, based on four (yes, no) questions:

Table 2 Strengths and weaknesses of organization of Ph.D. studies

	Strengths	Weight		Weaknesses	Weight
s_1	Experience in having full-time Ph.D. studies	0.3	w_1	Available scientific resources are overloaded with other tasks	0.3
s_2	Experience in supervising of Ph.D. thesis	0.3	w_2	Scientific research partially does not correspond to IT sector needs	0.2
s_3	Competences in computer science research	0.2	w_3	Dispersed scientific knowledge on different IT research fields	0.2
s_4	Experience in cooperation with IT sector	0.1	w_4	Inertia and resistance to change	0.2
s_5	Available e-learning system with base functions	0.1	w_5	No specialized distance learning courses	0.1

Table 3 Opportunities and threats of organization of Ph.D. studies

	Opportunities	Weight		Threats	Weight
o_1	Large number of IT companies in the neighborhood	0.3	t_1	Companies concentrating only on business activities	0.3
o_2	IT Sector specialists seek to gain knowledge on IT architecture and be distinguished by Ph.D. degrees	0.2	t_2	Companies own research groups concentrating on specialized tasks which find cooperation with university inefficient	0.2
o_3	Increasing trends for cooperation between industry and universities	0.2	t_3	Strong research confidentially in IT sector companies	0.2
o_4	Positive attitude towards internationalization of studies	0.2	t_4	Broad study domain (instead more specialization e.g. trustworthy IT systems)	0.2
o_5	Extension of HR cooperation to Ph.D. level	0.1	t_5	Narrow study domain (instead e.g. interdisciplinary studies)	0.1

- Can identified strengths take advantage of the opportunities?
- Can identified weaknesses preclude taking advantage of the opportunities?
- Can identified strengths overcome threats?
- Can identified weaknesses exacerbate threats?

Table 4 presents the SWOT influence matrix with the values 1 (yes), 0 (no).

The weighted coefficients of strengths, weaknesses, opportunities, and threats are denoted correspondingly as s_i, w_i, o_i, t_i.

The influence of strengths on opportunities, strengths on threats, weaknesses on opportunities, and weaknesses on threats is denoted by matrix elements correspondingly denoted as soij, stij, woij, wtij.

Table 4 SWOT influence matrix

		s_1	s_2	s_3	s_4	s_5		w_1	w_2	w_3	w_4	w_5
	so_{ij}	**0.3**	**0.3**	**0.2**	**0.1**	**0.1**	**wo_{ij}**	**0.3**	**0.2**	**0.2**	**0.2**	**0.1**
o_1	**0.3**	1	0	1	1	0	**0.3**	0	1	1	0	0
o_2	**0.2**	1	1	1	0	1	**0.2**	1	1	1	1	1
o_3	**0.2**	0	0	1	1	0	**0.2**	1	1	1	1	0
o_4	**0.2**	0	1	0	0	1	**0.2**	0	0	1	0	1
o_5	**0.1**	0	1	1	1	0	**0.1**	0	0	1	0	0
	st_{ij}	**0.3**	**0.3**	**0.2**	**0.1**	**0.1**	**wt_{ij}**	**0.3**	**0.2**	**0.2**	**0.2**	**0.1**
t_1	**0.3**	0	0	0	1	0	**0.3**	1	1	0	1	0
t_2	**0.2**	0	0	1	0	0	**0.2**	1	1	0	1	0
t_3	**0.2**	0	0	1	1	0	**0.2**	1	1	0	1	0
t_4	**0.2**	1	1	1	0	0	**0.2**	0	0	0	0	0
t_5	**0.1**	1	1	1	0	0	**0.1**	0	0	0	0	0

The estimated values of strategies are obtained by the corresponding formulas:

$$\mathrm{SO_v} = \sum_{i,j} s_i so_{i,j} o_j \tag{1}$$

$$\mathrm{ST_v} = \sum_{i,j} s_i st_{i,j} t_j \tag{2}$$

$$\mathrm{WO_v} = \sum_{i,j} w_i wo_{i,j} o_j \tag{3}$$

$$\mathrm{WT_v} = \sum_{i,j} w_i wt_{i,j} t_j \tag{4}$$

The calculation (1–4) of strategies for best strategy determination was based on the subjective value of coefficients for strengths, weaknesses, opportunities, weaknesses and the influence of dependencies between them. Analysis of the current situation shows that conservative ($\mathrm{WO_v} = 5.6$) and aggressive ($\mathrm{SO_v} = 5.4$) strategies seem be the best suited to the project. 4 classic SWOT strategies are considered for the project, presented in Table 5. The aggressive strategy is based on growth and attack and uses strengths to maximize opportunities. The conservative strategy is based on competition and defense and uses strengths to minimize threats. The competitive strategy is based on improvement and attack and minimizes weakness by taking advantage of opportunities. The defensive strategy is based on change and retreat to minimize weakness and avoid threats.

The SWOT analysis was performed under some subjective assumptions regarding IT-sector needs relating to ITAPhD studies. The subjective opinion is based on the author's experience of being CEO for 17 years of a large software

Table 5 SWOT strategies

Aggressive strategy	Conservative strategy
Use strengths to take advantage of opportunities. $\mathrm{SO_v} = 5.4$	Overcome weaknesses by taking advantage of opportunities. $\mathrm{WO_v} = 5.6$
• Open part-time Ph.D. studies • Start closer cooperation with IT companies • Prepare programs suitable for industry specialists • Use e-learning • Eventually combine selected courses from the 1st and 2nd year to reduce costs	• Start recruiting later. (officially open studies) • Start preparation and cooperation with industry for the next year • Prepare more for e-learning

Competitive strategy	Defensive strategy
Use strengths to avoid threats. $\mathrm{ST_v} = 3.6$	Minimize weaknesses and avoid threats. $\mathrm{WT_v} = 4.5$
• Compete with others by combining part time and full-time studies (reduces cost of studies, but studies' goals will be only partially met) • Join efforts with other similar Ph.D. studies	• Do not start the studies • Start cooperation with industry

development company, discussion with IT experts which resulted in varying (positive and negative) opinions regarding Ph.D. study needs, and the author's experience in leading post graduate studies. For the proper alignment of Ph.D. studies to IT-sector needs, additional surveys will be run. Risk will be mitigated by developing some collaboration models discussed in a later section.

6 Evaluation of Ph.D. Studies

The evaluation of Ph.D. studies is based on: preconditions for their inception, the area of studies (width and depth), and their quality, including considerations of the value of the program to the IT industry.

6.1 Preconditions for Applicants

The candidate must be a graduate of an M.Sc. program in the field of technical sciences, economics, or natural science, and should demonstrate necessary potential, willingness and determination to carry out research and education at the doctoral level. The applicant should be interested in research in the field of IT, be willing to use a systematic approach in his work, be able to formulate research problems and hypothesis, and exhibit experience in software development or in the application of information technologies. For applicants with significant curriculum differences, additional classes are proposed to obtain 6 ECTS points. This may be achieved by attending selected classes at the graduate level or IT Architecture Academy courses [16], that offer knowledge included in the approaches proposed by the Open Group, SEI and ISAQB.

6.2 Selecting the Areas for Ph.D. Studies

The areas selected for the ITAPhD study program should be closely related to solutions that are of current value for the IT sector and based on well-defined standards; for example, the IEEE Computer Society established a Guide to the Software Engineering Body of Knowledge (SWEBOK Guide [17]). An important SWEBOK goal is to offer curricula to undergraduate, graduate, and continuing education students. 15 SWEBOK KAs (Knowledge Areas) are distinguished.

Finally, three main areas are selected for the ITAPhD studies:

- Software system engineering related to systems and software. In this area, the following issues may be considered: System research; Ontologies of systems;

Table 6 Basic research landscape for Ph.D. studies

Layers	Research results
Business	Product commercialization
	Publications
	Knowledge services
Application	Methods development
	Software development
	Software development environment
Infrastructure	System Integration and building
	System support and Maintenance
	Infrastructure development

Methods, tools and experimental studies related to requirements, designing, architecture, verification and validation, maintenance and evolution of software.

- Software engineering related to the development and improvement of software development processes.
- Enterprise architectures related to: designing IT systems for enterprises, corporate architectures and application concepts in various branches of industry, inter-enterprise cooperation, collaborative development, virtual enterprise, corporate architecture design, service-oriented architecture (SOA), model-driven architecture (MDA), cloud computing, big data analytics (BDA), component-oriented architecture, systems integration, integrated manufacturing systems, industrial IT.

For evaluation and comparison purposes, a Ph.D. studies landscape is proposed. The landscape consists of segments (columns) and products and services (rows). The products and services can be divided into three groups: research results (Table 6), education, and research workshops.

6.3 *Quality of Ph.D. Studies*

The quality requirements are similar to full-time studies, as students are required to obtain a positive evaluation from their scientific coordinator for each academic year, must pass mandatory and selected optional subjects, and must receive formal approval of their selected doctoral research. Students should pass all the required doctoral exams in the third year of studies.

The performance measures of scientific entities in Poland are based on regulations from the Ministry of Science and Higher Education consisting of 4 basic criteria: scientific and/or creative achievements; scientific potential; concrete benefits of the scientific activity; intangible benefits of the scientific activity. In literature, this issue is investigated in greater depth [18]. Publishing of scientific papers is required in order to obtain formal approval of the conducted doctoral research and to defend a Ph.D. thesis successfully. Each year the Ministry publishes the number of credits assigned for publications in scientific journals.

It is important that part-time Ph.D. Studies should not waste students' time and should be finally completed after a formal approval of the thesis. The proposed metrics for the efficiency of studies could be the completion rate and the number of years taken to complete.

7 Collaboration Models

Many attractive non-academic career paths exist for Ph.D. graduates. An important issue is to align curricula at universities to the needs of the IT industry. For universities, the development of curricula and forms of education is important to attain the appropriate attractiveness of studies and compliance with the needs of the IT industry. This is important because the level of the adjustment of studies to the requirements of the labor markets is considered during university assessment and accreditation. Talent management has been introduced in many IT companies, an example of which is given in the paper [19], which proposes career paths for IT architects.

The proposed ITAPhD studies should use different collaboration models. The main models are discussed below.

Collaboration with industry—cooperation models:

- Industry experts working on topics suitable for Ph.D. theses. Having talented IT sector specialists on Ph.D. studies is important in order to bring industry cooperation to a new level.
- Knowledge exchange or common project realization in cases where industry and universities are interested in the same research. Students taking Ph.D. studies should collaborate on challenging projects
- Ph.D. study programs are adapted for industry needs. This adaptation can be achieved with the support of other university units.

Working in the industry and undertaking Ph.D. research not related to the current work will be difficult.

Collaboration with on-line learning organizations and librarian institutions:

- Virtual team cooperation supported by e-learning
- Video teaching materials from university staff
- Up-to-date scientific materials from libraries

There is also an intention to collaborate with international organizations in order to broaden the education offer. Collaboration with other universities and institutions could include:

- Assistance from experts (teachers, video lectures)
- Support for Ph.D. paths from professors

- Clusters. An interesting template for this may be the Cluster Hiring Initiative [20], which was designed to foster collaborative research, education, and outreach by creating new interdisciplinary areas of knowledge that are not restricted by the boundaries of existing academic departments

Other forms of collaboration should be proposed, such as the invitation of international students (e.g. tuition is in English) and interdisciplinary cooperation in the faculty.

8 Conclusions

The proposed part-time Ph.D. study program should be attractive for professionals from the IT branch, which means they must have good organization, be effective, and be based on new technological solutions and innovative research. The considered part-time program [21] allows their participants to reconcile professional activity with the possibility of developing new abilities of significant value. The course's orientation towards the IT sector's needs ensures that many issues that would-be Ph.D. students face will be included in the course curriculum.

Studies quality can be accomplished through proper organization and definition, and setting high levels of excellence. Some shifts in attitude at universities and in the IT sector are necessary to carry out the ITAPhD project successfully. Establishing Ph.D. studies faces two main collaboration problems. For university researchers, it may be difficult to collaborate with stakeholders and concentrate on values important to industry that coincide with university performance measures. For professionals from the industry, it may be difficult to contribute to research projects without obtaining instant results. The proposed solutions are designed to be internationally recognized.

The Ph.D. studies initiative has the following features: the official confirmation of the program and offer of Ph.D. studies; developing cooperation with industry in the fields of interest, taking into account possible candidate profiles and expectations relating to research; using e-learning platform including prepared video courses.

References

1. Yasinsac, A., Irvine, C.: Help! is there a trustworthy-systems doctor in the house? IEEE Secur. Priv. 73–77 (2013)
2. Tadeusiewicz, R.: Programme of the Ph.D. Studies at the Faculty of Electrical Engineering, Automatics, Computer Science and Electronics. AGH-UST, Attyka, Kraków (2011) (in Polish)
3. Bologna Declaration. http://en.wikipedia.org/wiki/Bologna_declaration

4. The Path Forward The Future of Graduate Education in the United States. Council of Graduate Schools and Educational Testing Service, p. 71. http://www.fgereport.org/rsc/pdf/CFGE_report.pdf (2010)
5. Szyjewski, Z. (ed.): Professional IT qualifications, Scientific Board of Polish Information Processing Society, p. 280 (in Polish). http://zbc.ksiaznica.szczecin.pl/dlibra/docmetadata?id=33572 (2015). ISBN 9788394269104
6. Howell Smith, M.C.: It's not what you think: a theory for understanding the lack of interest among domestic students in the engineering Ph.D. In: 41st ASEE/IEEE Frontiers in Education Conference (2011)
7. Smith, K.A., Streveler, R.A.: Special session—connecting and expanding the Emerging Engineering Education Research (EER) and Engineering Education Innovation (EEI) Communities. In: 41st ASEE/IEEE Frontiers in Education Conference (2011)
8. Bonnaud, O., Fremont, H., Thiriet, J.-M., Yahoui, H.: Ph.D. in Electrical and Information Engineering in Europe: towards a harmonization including LifeLong Learning. In: International Conference on Information Technology Based Higher Education and Training (ITHET) (2012)
9. Bonnaud, O., Fremont, H., Thiriet, J.-M.: On the way of harmonization of Ph.D. in Europe in Electrical and Information Engineering: status and recommendations. In: 23rd EAEEIE Annual Conference, p. 5 (2012)
10. Valdés, R.A., Moreno, J.C., García, F.J.S.E.: Educating engineering Ph.D. students for a Global World. In: Global Engineering Education Conference (EDUCON) (2012)
11. Djuricic, A., Grady, H.M., Graham, W.G.: The information economy: educational opportunities for industry-based professionals. In: IEEE International Professional Communication Conference (2008)
12. Werewka, J., Turek, M.: Computer science Ph.D. program evaluation proposal based on domain and non-domain characteristics. In: Świątek, J., Borzemski, L., Grzech, A., Wilimowska, Z. (eds.) Information Systems Architecture and Technology: Proceedings of 36th International Conference on Information Systems Architecture and Technology—ISAT 2015—Part III, pp. 177–187. Springer International Publishing. http://link.springer.com/chapter/10.1007/978-3-319-28564-1_15 (2016)
13. Jamróz, K., Pitulej, D., Werewka, J.: Adapting enterprise architecture at a software development company and the resultant benefits. In: Avgeriou, P., Zdun, U. (eds.) ECSA 2014, LNCS 8627, pp. 170–185 (2014)
14. TOGAF® Version 9.1, Open Group Standard, The Open Group, 2009–2011, p. 692
15. The Open Group: ArchiMate 2.1 Specification. http://pubs.opengroup.org/architecture/archimate2-doc/toc.html (2013). Accessed 18 May 2016
16. IT Architecture Academy. http://www.it-architecture.agh.edu.pl
17. Guide to the Software Engineering Body of Knowledge, Version 3.0, SWEBOK®. In: Bourque, P., (Dick) Fairley, R.E. (eds.) A Project of the IEEE Computer Society, (2014) p. 335
18. Koczkodaj, W., Kułakowski, K., Ligęza, A.: On the quality evaluation of scientific entities in Poland supported by consistency-driven pairwise comparisons method. Scientometrics (2014)
19. Turek, M., Werewka, J.: Motivation modeling and metrics evaluation of IT Architect certification programs. In: Kozielski, S., Dariusz, M., Małysiak-Mrozek, B. (eds.) Beyond Databases, Architectures, and Structures, pp. 463–472. Springer International Publishing (2015)
20. Cluster Hiring Initiative. University of Wisconsin-Madison. http://clusters.wisc.edu
21. IT Ph.D. Studies.http://www.it-phd-studies.agh.edu.pl/home (2016). Accessed 20 May 2016

Author Index

L. Madeyski et al. (eds.), *Software Engineering: Challenges and Solutions*,
Advances in Intelligent Systems and Computing 504,
DOI 10.1007/978-3-319-43606-7