Preface

This volume contains papers presented at the Fifth Workshop on Software Quality Analysis, Monitoring, Improvement, and Applications (SQAMIA 2016). SQAMIA 2016 was held during August 29 – 31, 2016., at the Faculty of Informatics of Eötvös Loránd University, Budapest, Hungary.

SQAMIA 2016 continued the tradition of successful SQAMIA workshops previously held in Novi Sad, Serbia (in 2012 and 2013), Lovran, Croatia (2014), and Maribor, Slovenia (2015). The first SQAMIA workshop was organized within the 5th Balkan Conference in Informatics (BCI 2012). In 2013, SQAMIA became a standalone event intended to be an annual gathering of researchers and practitioners in the field of software quality.

The main objective of the SQAMIA series of workshops is to provide a forum for presentation, discussion and dissemination of the latest scientific achievements in the area of software quality, and to promote and improve interaction and collaboration among scientists and young researchers from the region and beyond. The workshop especially welcomes position papers, papers describing work in progress, tool demonstration papers, technical reports, and papers designed to provoke debate on present knowledge, open questions, and future research trends in software quality.

The SQAMIA 2016 workshop consisted of regular sessions with technical contributions reviewed and selected by an international program committee, as well as of one invited talk by Prof. Kevin Hammond. In total 12 papers were accepted and published in this proceedings volume. All published papers were triple reviewed. We would like to gratefully thank all PC members for submitting careful and timely opinions on the papers.

Our special thanks are also addressed to the steering committee members: Tihana Galinac Grbac (Croatia), Marjan Heričko (Slovenia), and Hannu Jaakkola (Finland), for helping to greatly improve the quality of the workshop. We extend special thanks to the SQAMIA 2016 Organizing Committee from the Faculty of Informatics of Eötvös Loránd University, Budapest, and the Department of Mathematics and Informatics, Faculty of Sciences, University of Novi Sad, especially to its chair Tamás Kozsik for his hard work and dedication to make this workshop the best it can be.

The workshop is financially partially supported by the EU COST Action IC1202: Timing Analysis on Code-Level (TACLe).

And last, but not least, we thank all the participants of SQAMIA 2016 for their contributions that made all the work that went into SQAMIA 2016 worthwhile.

August 2016

Zoran Budimac
Zoltán Horváth
Tamás Kozsik
Workshop Organization

General Chair
Zoltán Horváth (Eötvös Loránd University, Hungary)

Program Chair
Zoran Budimac (University of Novi Sad, Serbia)

Program Committee
Nuno Antunes (University of Coimbra, Portugal)
Tihana Galinac Grbac (co-chair, University of Rijeka, Croatia)
Marjan Heričko (co-chair, University of Maribor, Slovenia)
Zoltán Horváth (co-chair, Eötvös Loránd University, Hungary)
Mirjana Ivanović (co-chair, University of Novi Sad, Serbia)
Hannu Jaakkola (co-chair, Tampere University of Technology, Finland)
Harri Keto (Tampere University of Technology, Finland)
Vladimir Kurbalija (University of Novi Sad, Serbia)
Anastas Mishev (University of Sts. Cyril and Methodius, FYR Macedonia)
Zoltan Porkolab (Eötvös Loránd University, Hungary)
Valentino Vranić (Slovak University of Technology in Bratislava, Slovakia)

Additional Reviewers
Cristiana Areias (Instituto Politécnico de Coimbra, ISEC, DEIS, Coimbra, Portugal)
Tânia Basso (School of Technology - University of Campinas (FT-UNICAMP), Portugal)

Organizing Chair
Tamás Kozsik (Eötvös Loránd University, Hungary)

Organizing Committee
Szilvia Ducerf (Eötvös Loránd University, Hungary)
Gordana Rakić (University of Novi Sad, Serbia)
Judit Juhász (Altagra Business Services, Gödöllő Hungary)
Zoltán Porkoláb (Eötvös Loránd University, Hungary)

Organizing Institution
Eötvös Loránd University, Budapest, Hungary

Steering Committee
Zoran Budimac (University of Novi Sad, Serbia)
Tihana Galinac Grbac (University of Rijeka, Croatia)
Marjan Heričko (University of Maribor, Slovenia)
Zoltán Horváth (Eötvös Loránd University, Hungary)
Hannu Jaakkola (Tampere University of Technology, Finland)

Technical Editors
Doni Pracner (University of Novi Sad, Serbia)
Gordana Rakić (University of Novi Sad, Serbia)

Sponsoring Institutions of SQAMIA 2016
SQAMIA 2016 was partially financially supported by:
EU COST Action IC1202 Timing Analysis on Code-Level (TACLe)
# Table of Contents

- Tool to Measure and Refactor Complex UML Models ............................................................ 1  
  Tamás Ambrus, Domonkos Asztalos, Zsófia Borbély and Melinda Tóth

- Product Evaluation Through Contractor and In-House Metrics ................................................. 9  
  Lucija Brezočnik, Črtomir Majer

- Software Metrics for Agent Technologies and Possible Educational Usage ................................. 17  
  Mirjana Ivanović, Marcin Paprzycki, Maria Ganzha, Costin Badica, Amelia Badica

- Why Information Systems Modelling Is Difficult .................................................................. 29  
  Hannu Jaakkola, Jaak Henno, Tatjana Welzer Družovec, Bernhard Thalheim, Jukka Mäkelä

- Pharmaceutical Software Quality Assurance System Architecture ........................................... 41  
  Bojana Koteska, Ljupco Pejov, and Anastas Mishev

- Assessing the Impact of Untraceable Bugs on the Quality of Software Defect Prediction Datasets ........ 47  
  Goran Mauša, Tibana Galinac Grbac

- XML Schema Quality Index in the Multimedia Content Publishing Domain .............................. 57  
  Maja Pušnik, Marjan Heričko, Boštjan Šumak, Gordana Rakić

- Combining Agile and Traditional Methodologies in Medical Information Systems Development Process .......... 65  
  Petar Rajković, Ivan Petković, Aleksandar Milenković, Dragan Jankovic

- How is Effort Estimated in Agile Software Development Projects? ........................................... 73  
  Tina Schweighofer, Andrej Kline, Luka Pavlič, Marjan Heričko

- First Results of WCET Estimation in SSQSA Framework ...................................................... 81  
  Natasa Sukur, Nemanja Milošević, Saša Pešić, Josef Kolek, Gordana Rakić, Zoran Budimac

- Towards the Code Clone Analysis in Heterogeneous Software Products ..................................... 89  
  Tijana Vislavski, Zoran Budimac, Gordana Rakić

- Monitoring an OOP Course Through Assignments in a Distributed Pair Programming System ........ 97  
  Stelios Xinogalos, Maya Satratzemi, Despina Tsompanoudi, Alexander Chatzigeorgiou
1. INTRODUCTION

Using UML modeling for designing a software product is heavily used by the industry. However, the tool support for model-based development has not reached the same level as the tool support for source code-based development. Our goal was to provide a tool to support refactoring and static analysis of UML models, which were developed in the open source modeling framework, Papyrus [Papyrus 2014].

There are tools, such as EMF Refactor [Arendt et al. 2010], that targets refactoring of EMF models. This tool provides an extensible framework for defining EMF model transformations and also model metric calculations. Several class refactorings and metrics have been defined in EMF Refactor. Therefore, we based our tool on this framework.

The main contributions of our work are the followings. (i) We have built a refactoring tool for Papyrus models based on EMF Refactor. (ii) We have implemented several state machine based refactorings. (iii) We have defined well-known model complexity metrics for state machines and introduced some new metrics to measure the models. (iv) We have implemented bad smell detectors and refactorings to reduce the complexity of the models.

The rest of this paper is structured as follows. In Section 2 we briefly introduce EMF Refactor and Papyrus. Section 3 illustrates the usage of our tool with an example at first, and then Sections 4, 5 and 6 present all of the features. Finally, Section 7 presents some related work and Section 8 concludes the paper.
2. BACKGROUND

We chose our product to be an Eclipse extension due to several decisions. Therefore it can build upon two other extensions: EMF Refactor and Papyrus. Since EMF Refactor is open source we can contribute our improvements when we reach a bigger milestone in this project.

2.1 Papyrus

[Papyrus 2014] is an open source model-based Eclipse extension. It can show the UML diagrams in a view in Eclipse. It also provides an other view for the semantic elements only, this is a kind of outline named model explorer. These two help users to edit all types of UML diagrams.

Model explorer and GUI editor work synchronously, meaning:
- clicking on an element on GUI should select the same element in model explorer,
- selecting an element in model explorer should select the same element on GUI,
- the appearing context menu should equal for elements in both views.

Modifying the underlying model programatically can cause differences in the separate views that must be handled manually.

2.2 EMF Refactor

[EMFRefactor 2011] is an open source tool environment for supporting model quality assurance process. It supports metrics, refactorings and smells of models based on EMF (Eclipse Modeling Framework). It basically builds upon the org.eclipse.ltk.core.refactoring Java package which supports semantic preserving workspace transformations [Arendt et al. 2010].

There are many predefined metrics, refactorings and smells for class diagrams [Arendt and Taentzer 2010]. This results a stable, useful tool to design reliable, easy-to-understand class diagrams. Preference pages are provided to the refactorings and also to the metrics and smells. These preference pages contain a list about the defined refactorings, etc. For the metrics, each item is related to a category. Preference pages come from UML elements (like 'Class'), it is also easy to expand the tool with self defined ones (e.g. 'State'). The aim of the preference pages is that users can choose items they want to work with. For example, marking a refactoring means that it can occur in the suitable context menu of a model element. Context menu appears if the user clicks (with the right mouse button) on an element.

The context menu filters accessible items based on the selected elements automatically. For example, while editing a state chart, class diagram refactorings are omitted from the list. It does not guarantee passing preconditions though, it makes only suggestions depending on the type of the selected elements.

The result of the selected metrics appears in a view named Metric Result View. This view does not follow the model changes, in order to have up to date measurements users need to re-run metrics on the changed model. Users can run metrics from the context menu of a selected element: in this case a subset of the selected metrics (based on the type of the selected element) will be evaluated. The result is a list which contains the name and value of the metrics.

New metrics, refactorings and smells can be added using the proper extension points.

3. TOOL DEMONSTRATION

It is hard to decide whether the quality of a model is adequate. Although we can estimate the understandability and maintainability by looking at it, metrics are essential tools of quality measurement. For a UML model, we can define numbers that describe it in details, such as number of elements, number of connections, etc., and have conclusions by examining the connections between them. This may be a very exhaustive process, thus it seems to be a good idea to use a tool for that.

The tool we have been developing is an extension of EMF Refactor. By using our tool, the user can use
predefined metrics that may contain information about the quality and complexity of a model. Metrics may show us bad design if the number is too large: by defining a threshold, besides detecting smells in the model, we also can eliminate them as they are in connection with specific refactorings, this way we can improve the quality of the model.

This tool also gives us proof that we improved the quality as the predefined metrics may show lower numbers and smells may disappear.

3.1 Example

To demonstrate our tool we use the example presented in [Sunyé et al. 2001]. A phone state diagram was described where the user can start a call by dialing a number, the callee can answer it if not busy and the two participants can talk until hanging up the phone. In the example, we got a flat state machine, therefore after we created the diagram that can be seen on Figure 3, we noticed that its quality can be improved: there are a lot of transitions with the same trigger that all go into state Idle. This problem can also be detected by using smells. If we select the project and open Properties window, we can set the proper thresholds of smells in EMF Quality Assurance/Smells configuration (Figure 1).

A suitable smell is Hot State (Incoming Transitions), that marks those states that have more incoming transitions than the threshold. If we set up the configuration as in Figure 1, then calculate all smells (right click on the model, EMF Quality Assurance/Find Configured Model Smells), it finds Idle as a smelly state (Figure 2). We can eliminate it in two refactoring steps: group the belonging states into a composite one and fold their transitions into a single one. You can see the result of these steps in Figure 4. By eliminating the similar transitions, we have got a simpler, clearer diagram. Moreover, the result can be measured: although the deepness of the state chart has increased, less transitions means less cyclomatic complexity, which is a good approximation for the minimum number of test cases needed.
Fig. 3. Flat state machine of mobile phone calls.

Fig. 4. State machine of mobile phone calls after a **Group states** and a **Fold outgoings** refactoring. Active states are grouped together and all transitions of the substates of Active composite state are folded into a single transition.

4. REFACTORINGS

Refactorings are operations that restructure models as follows: they modify the internal structure of the models, though the functionalities of the models remain the same. Models before and after refactoring are functionally equivalent since solely nonfunctional attributes change. A fully defined refactoring consists of preconditions, postconditions, main-scenario (changes of the model) and a window for additional parameters.

Many refactorings are provided by EMF Refactor. All of them can be used on class diagrams, e.g. add parameter, create subclass, create superclass, move attribute, move operation, pull up attribute, pull up operation, push down attribute, push down operation, remove empty associated class, remove empty subclass, remove empty superclass, remove parameter, rename class, rename operation and several compositional refactoring operations. One of our goals was to extend these and visualize them properly.
4.1 Visualization

The existing class diagram refactorings modify only the EMF model, the result is not visible in the Papyrus diagram, therefore our first goal was to add this feature. This involved programmatically creating and deleting views of model elements simultaneously with the EMF model changes. The main aspects were not only to refresh the Papyrus model, but also to support undoing in a way that every change can be reverted in one step. To achieve that, transactions are supported in EMF Refactor which means that it detects the changes during the refactoring process and stores them in a stack, providing an undoable composite command. Unfortunately, EMF and Papyrus changes cannot be made in the same transaction due to multithreading problems, thus we implemented a solution where atomic actions (add, remove) are caught, and we modify the diagram by handling these. We try to keep the consistency of the model and the corresponding diagram.

4.2 New refactorings

Since EMF Refactor defines refactorings only for class diagrams, our other goal was to create refactorings for state machines. State machines provide many opportunities to refactor model elements causing small and consistent changes. Most of the refactorings we implemented may be found in [Sunyé et al. 2001], that contains the pre- and postconditions of all refactorings. In the article, postconditions differ from the ones defined in EMF Refactor, they must be checked after the refactoring process to guarantee that the refactoring made the specific changes.

In order to refactor successfully, our refactorings first check the preconditions, then pop up a window, that contains a short description of the selected refactoring and the input fields for the parameters – some of the refactorings needs additional parameters, e.g. name of the composite state which will be created. After that, it checks the conditions that refer to the parameters, then executes the proper changes. If any of the conditions fails, the execution is aborted and the error list is shown.

The added state machine refactorings are:
- Group States: it can be used by selecting many states to put them into a composite state instead of moving them and their transitions manually,
- Fold Entry Action: to replace a set of incoming actions to an entry action,
- Unfold Entry Action: to replace an entry action by a set of actions attached to all incoming transitions,
- Fold Exit Action: to replace a set of outgoing actions to an exit action,
- Unfold Exit Action: to replace an exit action by a set of actions attached to all outgoing transitions,
- Fold Outgoing Transitions: to replace all transitions leaving from the states of a composite to a transition leaving from the composite state,
- Unfold Outgoing Transitions: the opposite of the Fold Outgoing Transitions,
- Move State into Composite: to move a state into a composite state,
- Move State out of Composite: to move a state out of a composite state,
- Same Label: copy the effect, trigger and guard of a selected transition.

The refactorings are executed regarding the predefined conditions to keep semantics and they also modify the Papyrus diagram.

We also implemented two new important class diagram refactorings:
- Merge Associations: associations of class A may be merged if they are of the same type and they are connected to all subclasses of class B,
- Split Associations: the opposite of the Merge Associations refactoring.
5. METRICS

Metrics are able to increase the quality of the system and save development costs as they might find faults earlier in the development process. Metrics return numbers based on the properties of a model from which we may deduce the quality of the model. For example, a state machine with numerous transitions may be hard to understand as the transitions may have many oscillations. On the other hand, states embedded in each other, with a deep hierarchy, might also be confusing. According to this, by calculating the metrics we get an other advantage: we can detect model smells, furthermore, some of them can be repaired automatically. We describe this in Section 6.

In EMF Refactor there are many class, model, operation and package metrics defined. Our goal was to create state and state machine metrics. State metrics measure the state and the properties of its transitions, while state machine metrics calculate numbers of the whole state machine. We added 10 state (Table I.) and 16 state machine (Table II.) well-known metrics, all of them measure the complexity of the model. These metrics can be easily calculated as they have simple definitions: they describe the model by the number of items and connections in the model.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>st_entryActivity</td>
<td>Number of entry activity</td>
<td>st_doActivity</td>
<td>Number of do activity (0 or 1)</td>
</tr>
<tr>
<td>st_exitActivity</td>
<td>Number of exit activity (0 or 1)</td>
<td>st_NOT</td>
<td>Number of outgoing transitions</td>
</tr>
<tr>
<td>st_NTS</td>
<td>Number of states that are direct target states of exiting transitions from this state</td>
<td>st_NSSS</td>
<td>Number of states that are direct source states of entering transitions into this state</td>
</tr>
<tr>
<td>st_NDEIT</td>
<td>Number of different events on the incoming transitions</td>
<td>st_NITS</td>
<td>The total number of transitions where both the source state and the target state are within the enclosure of this composite state</td>
</tr>
<tr>
<td>st_SNL</td>
<td>State nesting level</td>
<td>st_NIT</td>
<td>Number of incoming transitions</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NS</td>
<td>Number of states</td>
<td>NSS</td>
<td>Number of simple states</td>
</tr>
<tr>
<td>NCS</td>
<td>Number of composite states</td>
<td>NSMS</td>
<td>Number of submachine states</td>
</tr>
<tr>
<td>NR</td>
<td>Number of regions</td>
<td>NPS</td>
<td>Number of pseudostates</td>
</tr>
<tr>
<td>NT</td>
<td>Number of transitions</td>
<td>NDE</td>
<td>Number of different events, signals</td>
</tr>
<tr>
<td>NE</td>
<td>Number of events, signals</td>
<td>UUE</td>
<td>Number of unused events</td>
</tr>
<tr>
<td>NG</td>
<td>Number of guards</td>
<td>NA</td>
<td>Number of activities</td>
</tr>
<tr>
<td>NTA</td>
<td>Number of effects (transition activities)</td>
<td>MAXSNL</td>
<td>Maximum value of state nesting level</td>
</tr>
<tr>
<td>CC</td>
<td>Cyclomatic complexity (Transitions - States + 2)</td>
<td>NEIPO</td>
<td>Number of equal input-parameters in sibling operations</td>
</tr>
</tbody>
</table>

6. BAD SMELL DETECTION

As mentioned earlier, model quality is hard to measure, but with simple heuristics – smells – we can detect poorly designed parts. Moreover, in some cases we can offer solutions to improve them using specific refactorings.

Useful smells consist of a checker part and a modification part. Checker part may contain metrics and conditions: we can define semantics for the values of the metrics. Categorizing the values means that
we can decide whether or not a model is good or smelly. The modification part may contain refactorings to eliminate the specified smells.

EMF Refactor defines 27 smells for class diagrams, about half of them is rather a well-formedness constraint than a real smell: unnamed or equally named elements. Most of them provides useful refactorings to eliminate the bad smells.

In our extension, we implemented four important metric-based smells for state machines:
- **Hot State (Incoming)**: a threshold for the number of incoming transitions,
- **Hot State (Outgoing)**: a threshold for the number of outgoing transitions,
- **Deep-nesting**: a threshold for the average nesting of states,
- **Action chaining**: a threshold for transitions, its main responsibility is to recognize whether too many entry and exit actions would be executed in a row.

We can also detect unnamed or unreachable states and unused events.

Defining the thresholds of the smells is not easy ([Arcelli et al. 2013](#)), they may vary in the different projects. We defined the smells and the default thresholds based on the experience of our researchers and the reference values used in the state-of-the-art. If the users find these values inappropriate to their models, they can modify them in our tool manually.

### 6.1 Smell based refactorings

It is not always obvious which refactorings can help to eliminate bad smells. As we presented in Section 3, a state with a large number of incoming transitions can be simplified in two steps: group the states where the transitions come from, then fold the outgoing transitions of the new composite state. Having a large number of outgoing transitions is more complex. An idea is to describe the smelly state more precisely, this way the substates and details may explain the different scenarios, but unfortunately it also increases the complexity. In this topic further research is needed.

Deep-nesting can be improved by using specific Move State out of Composite refactorings. A further step could be to detect "unnecessary" composite states which increase the complexity more by nesting than decrease it by folding transitions. In connection with that, an important aspect is that by using composite states, code duplication is reduced as common entry and exit actions do not have to be duplicated in substates.

Finally, action chaining is a very complex problem. It depends on the specific actions if it can be reduced or fully eliminated. Though detection is useful and shows a possible bad design, it may be better to be handled manually.

### 7. RELATED WORK

The literature regarding the metrics and refactorings of UML models is extensive. Since our interest is tool developments for executable UML models, our review of the related work is focused on the tool-related topics.

The most well-known model measurement tool is [SDMetrics 2002](#). It is a standalone Java application having a large set of predefined metrics for the most relevant UML diagrams. SDMetrics also supports the definition of new metrics and design rules, and is able report the violation of design rules. Although several metrics mentioned earlier in this paper were first implemented in SDMetrics by our project team, we have decided to use EMF Refactor because of the refactoring support and the easy integration with Eclipse.

The recent developments show an increased interest in the combination of metrics evaluation and refactoring services in a single toolchain. A good starting point for the review is the survey published by [Mishbhauddin and Alshayeb 2015](#). The survey refers to 13 publications (including EMF Refactor) about state chart measurements and refactorings. An other publication dealing with state charts and
providing full automatic refactorings is [Ruhroth et al. 2009]. It is based on RMC Tool, a quality circle tool for software models. [Dobrzanski 2005] presents five different tools that describe or implement model refactorings. Refactoring Browser for UML focuses on correctly applied refactorings, while SMW Toolkit describes new refactorings without validation. The goal of Odyssey project is improving understandability by defining smells for class diagrams.

Compared to these researches our tool aims to support model driven development in the Eclipse framework based on Papyrus and EMF Refactor. We want to provide a tool that is built into the daily used modeling environment of the users, and there is no need to use a separate tool: the users can develop, maintain, refactor, measure and analyse their models in the same environment. One more reason that made us to choose EMF Refactor is the txtUML toolchain developed by our research group. In the txtUML toolchain executable UML models can be defined textually and the framework is able to generate executable code and Papyrus diagrams as well [Dévai et al. 2014]. Naturally, we can use our tool with the generated diagrams, nevertheless it would be a great advancement to measure and refactor them before the generation, using only the textual definition of state machines. We want our tool to be an important part of the txtUML toolchain as well.

8. SUMMARY

We presented a tool that is able to measure the complexity of state charts and execute transformations to reduce their complexity. Besides implementing metrics, smells and refactorings in connection with state machines, we extended the original functionality of the EMF Refactor tool with the feature of Papyrus visualization.

We plan to implement more refactorings and smells in order to improve automation of model quality assurance. An important point of view is that we defined only metric-based smells, but in EMF Refactor, graph-based smells are also supported. Our plan is also to validate these refactorings to ensure the consistency of the model.

REFERENCES


Product Evaluation Through Contractor and In-House Metrics

LUCIJA BREZOČNIK AND ČRTOMIR MAJER, University of Maribor

Agile methods are gaining in popularity and have already become mainstream in software development due to their ability to produce new functionalities faster, and with higher customer satisfaction. Agile methods require different measurement practices compared to traditional ones. Effort estimation, progress monitoring, improving performance and quality are becoming important as valuable advice for project management. The project team is forced to make objective measures to minimise costs and risks with rising quality at the same time. In this paper, we merge two aspects of agile method evaluation (the contractor and the client view), propose AIM acronym and discuss two important concepts in order to perform objective measurements: “Agile Contractor Evaluation” (ACE) and “Agile In-House Metrics” (AIM). We examine what type of measurements should be conducted during agile software development methods and why.

Categories and Subject Descriptors: D.2.8 [Software Engineering]: Metrics—Performance measures; Process metrics; Product metrics
General Terms: agile software development, agile metrics, agile contractor evaluation
Additional Key Words and Phrases: agile estimation

1. INTRODUCTION

The transition from the waterfall development process and its variations to an agile one poses a challenge for many companies [Green 2015, Laanti et al. 2011, Schatz and Abdelshafi 2005, Lawrence and Yslas 2006]. Examples of organisations that have successfully carried out the transition are Cisco [Cisco 2011], Adobe [Green 2015], Nokia [Laanti et al. 2011], Microsoft [Denning 2015], and IBM [IBM 2012]. However, not all companies have the same aspirations when it comes to the reason they want to introduce an agile approach [VersionOne 2016]. The most common reasons include: to accelerate product delivery, to enhance the ability to manage changing priorities, to increase productivity, to enhance software quality, etc. However, the metrics of success need to be selected wisely. Based on the recently released 10th Annual State of Agile Report [VersionOne 2016], the main metrics are presented in Figure 1.

A majority of agile methods share an important aspect in terms of development planning. Each determines the preparation of prioritised features that need to be done (e.g. Product Backlog in Scrum). Developers pull features from the list in relation to the capacity that is currently available, i.e. if a company uses Scrum, the pull of features is used only at the beginning of each iteration (sprint). In the case of Kanban, the pull of features is continuous. Because agile methods are about team and teamwork, they all determine some kind of regular interaction between the development team and management, and communication within the development team.

The agile metrics are widespread in agile companies in order to monitor work and tend to make improvements inside them. In this paper, we will discuss the metrics that are used in agile software development from the point of “Agile Contractor Evaluation” (ACE) and “Agile In-House Metrics” (AIM). ACE covers the approaches for monitoring the progress of agile contractors, while AIM

This work is supported by the Widget Corporation Grant #312-001.
Author's address: L. Brezočnik, Faculty of Electrical Engineering and Computer Science, University of Maribor, Smetanova 17, 2000 Maribor, Slovenia; email: lucija.brezocnik@um.si; Č. Majer, Faculty of Electrical Engineering and Computer Science, University of Maribor, Smetanova 17, 2000 Maribor, Slovenia; email: cromir.majer1@um.si.

Copyright © by the paper's authors. Copying permitted only for private and academic purpose.
approaches are used for monitoring work and workflow within the company. All metrics discussed in this paper were obtained on the basis of preliminary research.

During this paper, we will address the following two research questions:

- RQ1: What are the metrics and their usage in agile project management within the company?
- RQ2: How to evaluate an agile contractor?

2. RELATED WORK

The topic of software metrics has been covered well in several scientific papers. In this section, we mention only a few that are directly relevant to our research.

In the paper “Model of Agile Software Measurement: A Case Study” [Gustafsson 2011] it is stated that metrics are obtained in the form of a research method case study. Some authors (e.g. [Drury-Grogan 2014]) have conducted an interview with the intention of identifying iteration objectives and critical decisions, which refer to the golden triangle. A commonly used research method is also surveyed, e.g. described in [Lei et al. 2015]. The main purpose of it is to identify the effect of agile metrics Scrum and Kanban on software development projects backed up by statistical analysis. Some research done focuses only on identifying metrics for the improvement of specific aspects of the business, e.g. business value [Hartmann and Dymond 2006]. In [Qumer and Henderson-Sellers 2008] the authors have proposed a framework to support the evaluation, adoption and improvement of agile methods in practice. Some authors have tackled the problem of metrics in a comprehensive way and carried out a systematic literature review [Kupiainen et al. 2015]. However, in none of the analysed articles were the AIM and ACE metrics collectively accounted. This is why we committed ourselves to this article.

3. AGILE IN-HOUSE METRICS (AIM)

With the term “Agile In-house Metrics” we are referring to metrics that are measured in software development companies. With them, we are trying to predict delivery times, costs and delivery capacity, so that all initially planned features will be realised. We present the most commonly recommended metrics and allocate them in the following four main sections [Gustafsson 2011, Kammelar 2015]: lean metrics, business value metrics, quality metrics, and predictability and team productivity metrics.
3.1 Lean metrics

The Lean software development is one of the most popular agile development methods [Poppendieck 2015]. It promotes seven software development principles, which are very similar to those in lean manufacturing, initially used by Toyota [Gustafsson 2011]. Lean focuses on optimising flow efficiency across the entire value stream [Poppendieck 2015]. Lean principles are [Poppendieck 2015]: eliminate waste, amplify Learning, decide as Late as Possible, deliver as Fast as Possible, empower the Team, build Quality in, and see the Whole.

The most commonly used lean metrics are the following: Lead time is measured by elapsed time, such as minutes, or hours for one feature to go through the entire process from start to finish [Mujtaba et al. 2010]. Optimal Lead time is as short and stable as possible. It can be used to schedule high priority features and plan demo dates with customers [Kupiainen et al. 2015, Polk 2011] or for monitoring work in progress to predict a project schedule. On the other hand, we need to be careful if the Lead time is used as a metric for productivity. Bug fixes and quality improvement demand time.

Work in Progress (WIP) represents the set of features development team is working on [Middleton et al. 2007]. For controlling the mentioned Lead time, WIP constraints are used.

The Queue is another metric and is defined in [Staron and Meding 2011] as the “number of units remaining to be developed/processed by a given phase or activity.” Unmanaged queues have an effect on Lead time and workflow such as increased Lead time and ineffective flow, respectively. Cycle Time is measured by the amount of time per unit, where the unit means, for example, hours/feature. In other words, it is the amount of time it takes for a specific feature to be completed [Polk 2011].

3.2 Business Value Metrics

The business value metrics are intended for long-term business process monitoring. Every business tries to get a high return on investment (ROI) and profit for any investment or at least to break even. It is important to highlight that ROI may not always be measured in the financial aspect. Earned Business Value (EBV) [Javdani 2013] metrics may be used in conjunction with the ROI, where estimated ROI is prorated to the features of each iteration in the financial aspect.

One of the commonly recommended metrics for business value is also the Customer Satisfaction Survey, where we ask the customers if they are satisfied with the product and Business Value Delivered. The latter can be interpreted by delivered features per timeframe [Kupiainen et al. 2015, Abbas 2010]. To estimate future expenses in software development, the Average Cost per Functions (ACPF) [Gustafsson 2011] metric is recommended.

3.3 Quality Metrics

As presented in the paper “Metrics and Models in Software Quality Engineering” [Kan 2002], the quality metrics are classified as Product metrics, Process metrics, or Project metrics. The product metrics monitor the characteristics of the product, e.g. size, complexity, and quality. The process metrics cover defect handling and can be used for software development improvement and maintenance. The project characteristic, e.g. cost, productivity, and schedule, are measured by project metrics. The following is a presentation of the most proposed metrics for quality.

The Technical Debt is a metric that is recommended by the Agile community [Behrens 2009]. It measures the amount of hours that are required to fix the software related issues [Kupiainen et al. 2015]. In the paper “Model of Agile Software Measurement: A Case Study” [Gustafsson 2011] says that is optimal to translate Technical Debt to money. In such an approach, it is hard to be accurate.

The Defect Count, another quality metric, counts defects in each iteration. It is important to highlight that defects can be viewed from different perspectives. One way is to use Faults-Slip-Through (FST) metrics [Damm et al. 2006] as a measurement of a number of defects that have not been found when it was most cost-effective. FST could also be used as a predictor of fault-proneness [Afzal 2010].
3.4 Predictability and team productivity metrics

Agile methods emphasise the predictable performance of the monitored process [Hayes et al. 2014]. Commonly recommended metrics in this section are as follows.

The Velocity can be measured directly or indirectly [Padmini et al. 2015]. The direct approach is done using the monitoring volume of the work done by a given team in a given timeframe. It is important to stress that velocity is not measured by the ratio between the completed and committed stories (i.e. stories that were committed by developers to be completed in the iteration). If the development team realises that their work is primarily measured with it, they will use the following approach: “Tell me how you measure me, and I will tell you how I will behave” [Goldratt 1990]. Another negative part of the improper measurement of velocity as a productivity measure is the degradation of the quality of the developed software [Gustafsson 2011]. An indirect way of velocity measurement is done by bug correction time.

Running Automated Tests is the metrics for measuring a team’s productivity by the size of the product [Gustafsson 2011]. It comprises all the features that are covered by automated tests.

4. AGILE CONTRACTOR EVALUATION (ACE)

In comparison to AIM, where measures are taken and evaluated inside a software development company, ACE describes ways to monitor the progress of agile contractors. The US Department of Defense [Hayes 2014] proposes the following list of activities for successful contractor evaluation during the project.

4.1 Software Size

For monitoring progress in traditional software development, the size of the product is often an acceptable indicator for eventual costs and schedule. Agile methods tend to fix costs and schedule using timebox planning or work in progress limits, leaving software size as the only variable [Lei et al. 2015]. The goal is to maximise the delivery of new functionalities to the customer, depending on work capacity via the story prioritisation (high priority stories first). In the agile world, software size is typically represented in story points or delivered user stories. Each iteration is planned with a subset of the most important user stories, that are planned to be completed according to team capacity. Therefore, minimising the difference between the number of story points planned and the number of stories delivered becomes the main focus. Size is used to help the team understand its capacity and serves as a basis for performance diagnosing. If a team has continuing problems with delivered story points, that indicates the capability of the team is not realistic and/or that the scope of work is not well understood [Hayes 2014].

4.2 Effort and Staffing

It is important to track effort and staffing since they tend to be the primary cost drivers. Using agile methods, there is a notable change in the staff utilisation pattern. During the early phases of development, a risk of slow ramp-up in staffing (the time until the team is fully formed) can appear. There also seems to be a decline in the number of participants towards the end of the development cycle, leading to possible complications [Hayes 2014].

4.3 Schedule

Tracking velocity over iterations and releases provide the basis for understanding the pace of delivery. It also helps with estimating the completion time of the remaining work in a backlog. With schedule analysis, the client can measure confidence in an agile contractor, regarding the timely completion of the agreed scope of work, and is thus able to identify potential risks early in the process. Confidence is a result of trust and previous experience with the contractor. A precise schedule is not a priority in
agile development, therefore focusing on things like schedule performance is less meaningful, in contrast to traditional approaches [Hayes 2014].

4.4 Quality and Customer Satisfaction

Agile methods put a lot of attention on the product quality, right from the beginning of a project. The acceptance criteria for each user story should be precisely defined, so that every product increment passes through multiple stages in the quality assurance process (unit testing, integration testing, UI testing, acceptance testing etc.), mitigating the risk of defects in a production environment. A customer plays a prominent role of prioritising user stories as well as providing timely feedback at the completion of each sprint. Defects, discovered and fixed during the quality assurance process inside each iteration/story provide the basis for quality metrics. This problem can also be addressed with the usage of Software Defect prediction techniques like presented in the paper “A Systematic Data Collection Procedure for Software Defect Prediction” [Mauša 2015].

The cumulative flow diagram is a powerful visualisation tool for early quality problem recognition. Figure 2 shows an example of such a diagram with an explanation of how to read different measures. It also displays two unhealthy states of a project due to inability to complete a development task, because of defects or other issues (A) and the poor rate at which new work is taken into the development process (B).

Patterns similar to (A) can lead to Technical Debt, which threatens the successful realisation of a project. Ideally, we want consistent and narrow bands for all measures, like the ones for validated and assigned categories. Pattern B shows that the development team is sitting idle or working on unplanned tasks out of iteration scope. Poor performance during one stage of the development process affects all the subsequent stages, which can be seen as leading to emerging gaps in the cumulative flow diagram [Hayes 2014].

![Fig. 2. How to read Cumulative Flow Diagram](image)

4.5 Requirements

Requirements, typically represented as user stories, often change over time. Agile methods are very flexible about changes. In some environments, requirements in the backlog are less detailed, thus not providing all the necessary information for implementation. That information is acquired via a direct conversation with users or their surrogates which prolong the time of activity completion. In other environments, user stories are fully defined, including information about “done criteria” and test cases used for verifying the product increments. Metrics that reveal how well-defined are the requirements, include: a number of changes in user stories, the number of items in a backlog (where “done criteria”
needs to be specified), the cycle time between a customer identifying the new story as a top priority, and similar measures [Hayes 2014].

4.6 Delivery and Progress

Agile methods give product delivery a higher priority compared to other attributes such as costs, durations, and schedule. Frequent delivery of working software renders a more direct view of progress, allowing early opportunities to refine the final product and assure that the development team is moving toward the desired goal [Hayes 2014].

5. AGILE ESTIMATION

We presented agile methods and metrics used for project management. As can be seen, every project needs some estimation – costs, time or quality. Since low costs are not the first priority of the agile movement [VersionOne 2016], it leaves us with two variables. To estimate timeframe, we need to know the amount of work in the product/iteration backlog, which is defined with requirements and quality is best monitored through cumulative flow diagram.

For a typical Agile project, once a list of stories is made for iteration, the typical next steps are prioritisation and estimation. The overtime goal is to steady the number of story points delivered per iteration (work capacity), thus providing some fixed value for long-term scheduling [Zang 2008]. Story points per iteration, day, hour (any fixed unit of time) are often referenced as a metric named Takt Time. It is the rate at which product increments must happen in order to meet customer demand. Product Delivery Time, in means of time-limited increment, is then calculated as Takt Time × Number of Story Points. With a known number of story points and product deadlines, we can then calculate the necessary Takt time, to finish the product increment on time [Bakardzhiev 2014].

Forecasting software delivery dates is possible with different types of simulation such as the Monte Carlo Simulation. Monte Carlo Simulation is a problem-solving technique used to approximate the probability of certain outcomes by running multiple simulations using random variables. In our case, it provides for the probabilities of finishing all project activities at a given future date. In Figure 3, we can see the probability distribution of project completion, measured in the number of hours [Odenwelder 2015].

The simulation gives us a result – with a given Takt Time our team is able to deliver a completed product in around 500 work hours with a probability of 50%. However, this estimation is pretty risky.
A better estimation would be one where probability approaches 100% (that is around 530 work hours). The date of project completion is not a single value, but a set of possible results and it depends on us how much risk we are willing to take. There are multiple variations of Monte Carlo Simulation for agile estimation. We presented the basic one, where Takt Time is related to work hours, not taking task estimation errors or the stability of velocity into account. Regardless, from a simulation like this, we can get a better insight into how long a project can last, how many team members we need, what are the potential costs, what is the desired Takt Time, etc.

6. DISCUSSION

Our goal to propose the acronym AIM and endorse it by scientific papers is fully met. With it, we wanted to highlight the important influence of groups of metrics AIM for process improvement and ACE for the indirect assessment of the expected product. With this noted, we can answer the asked questions.

The RQ1 was aimed at discovering the metrics and their usage in agile project management within the company. We presented the most used and popular metrics, with which we could control the process of agile software development. The study showed that agile teams usually decide for velocity, burndown charts, planned vs actual, or work in progress. These metrics encourage the main objectives of agile – on time delivery, the ability to make changes quickly, a motivated team and, of course, a satisfied customer. Furthermore, metrics help us with decision making, optimising workflows, eliminating waste and making improvements over time. Our overview of metrics and measures can be used in a software development firm, to increase awareness among management, agile practitioners, support and within the team itself, to assist the company’s growth.

The second question, RQ2, was aimed at finding how to evaluate a contractor. While most papers try to answer RQ1, we also gave some attention to the customer side – how do they evaluate an agile contractor? Customers are typically not as well educated about agile practices. Therefore, they might not know what to monitor, measure, and evaluate. We presented key perspectives and practices for the customer side, to understand how well a contractor performs on a daily basis and overall. With the help of these practices, a customer can quickly notice difficulties in the project and warns the contractor about them – this way, both parties participate in reducing risk and improving their trust.

Our future plans are to test proposed metrics in a real life scenario in the form of a research method Case study or Experiment. We would also like to verify whether and to what extent the Monte Carlo method assists in agile estimation and conduct additional research on the evaluation of products.

REFERENCES

DOI:http://dx.doi.org/10.1109/AGILE.2006.17
Software Metrics for Agent Technologies and Possible Educational Usage

MIRJANA IVANOVIĆ, University of Novi Sad
MARcin PAPRZYCKI, Polish Academy of Sciences and Warsaw Management Academy
MARia GANZHA, Polish Academy of Sciences and Warsaw University of Technology
COSTIN BADICA and AMELIA BADICA, University of Craiova

In a long history of inventing different methods for assessing software quality huge number of software metrics has been proposed. In this paper we will first give a brief overview of different approaches to measure software and all processes connected to software development. Particular attention further will be paid on metrics in agent technology and multiagent systems (MAS) that are based on “classical metrics” but introduce novel specific techniques in line with agent paradigm. In spite the fact that there is a number of metrics developed for software agents and MASs it is a little bit strange that in higher ICT education we do not have appropriate topics devoted to these very specific metrics. Such drawback should be resolved in some acceptable way. In this paper we try to propose some adequate ways and possibilities of improving current situation in ICT higher-education in agent paradigm and suggest several slight changes in existing curricula.

Categories and Subject Descriptors: D.2 [SOFTWARE ENGINEERING]; D.2.8 [Metrics]; D.3 [PROGRAMMING LANGUAGES]; I.2.11 [Distributed Artificial Intelligence]: Multiagent systems; K.3 [COMPUTERS AND EDUCATION]

General Terms: Measurement, Metrics, Education

Additional Key Words and Phrases: Agent-oriented metrics, Application of agent-oriented metrics in education.

1. INTRODUCTION

Software metrics are based on old discipline of measurement invented by different scientists. Based on this, overtime, essential principles to measure software and adequate connected activities are being proposed. First software metrics have appeared in the sixties. Among them, the most well-known is the Lines of Code (LOC) metrics. This and a lot of other metrics had been proposed to measure program quality through programmer’s productivity. In long history of inventing different methods for assessing software quality in the literature, two important activities could be distinguished: the measurement and the software metrics.

Two fundamental definitions of the measurement were proposed by Norman Fenton [Alexandre 2002]. The first one states that: “Formally, we define measurement as a mapping from the empirical world to the formal, relational world. Consequently, a measure is the number or symbol assigned to an entity by this mapping in order to characterize an attribute”. The second definition includes a numerical aspect: “Measurement is the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to describe them according to clearly defined rules.”

This work is partially supported by the Multilateral Agreement “Agent Technologies: Intelligent Agents and Smart Environments” between Serbia, Poland and Romania.

Author’s address: Mirjana Ivanović, University of Novi Sad, Faculty of Sciences, Department of Mathematics and Informatics, Novi Sad, Serbia; e-mail: mira@dmi.uns.ac.rs; Marcin Paprzycki, Systems Research Institute Polish Academy of Sciences, Warsaw, Poland and Warsaw Management Academy, Warsaw, Poland; e-mail: paprzyck@ibspan.waw.pl; Maria Ganzha, Systems Research Institute Polish Academy of Sciences, Warsaw, Poland and Warsaw University of Technology, Warsaw, Poland; e-mail: maria.ganzha@ibspan.waw.pl; Costin Badica, Department of Computer Science, University of Craiova, Craiova, Romania; e-mail: cbadica@software.ucv.ro; Amelia Badica, Faculty of Economics and Business Administration, University of Craiova, Craiova, Romania; e-mail: ameliabd@yahoo.com.

Copyright © by the paper’s authors. Copying permitted only for private and academic purposes.

The same author proposed the first definition of software metrics [Alexandre 2002]: “...software metrics is a collective term used to describe the very wide range of activities concerned with measurement in software engineering. These activities range from producing numbers that characterize properties of software code (these are the classic software 'metrics') through to models that help predict software resource requirement and software quality. The subject also includes the quantitative aspects of quality control and assurance - and this covers activities like recording and monitoring defects during development and testing.”

Paul Goodman [Goodman 1993] proposed another definition of software metrics: “The continuous application of measurement-based techniques to the software development process and its products to supply meaningful and timely management information, together with the use of those techniques to improve that process and its products”.

Separately, measurements related to software efficiency are the core of, so-called, scientific computing. Here, the main question that is being answered can be stated as: “how efficient is given implementation of a given algorithm when being executed on a given computer” [Paprzycki and Stpiczynski 2006]. While it can be claimed, that this aspect of software metrics is already captured by the above definitions, what should be acknowledged is the fact that, during last 40+ years, scientific computing has developed its own metrics, well grounded in engineering practices.

During long period of developing and application of different software metrics, several domains of their general use are recognized:
- Metrics are essential to obtain objective reproducible measurements that can support software quality assurance, debugging, performance, estimating costs and management.
- Metrics are good instrument for discovering and predicting defects in code, but also for predicting project risks and success.
- Metrics are also oriented to quantifying several goals in project management: quality of software, cost and schedule of a software project, size/complexity of development.
- Metrics allow to select the most efficient approach to implementation of algorithms on complex large-scale computer systems.

Constant development, evaluation and application of different innovative software metrics are definitely necessary in order to obtain higher quality of software development and final product. Nevertheless the fact that majority of classical software metrics could be applied for software produced for specific areas, development of new metrics is still very dynamic area. As software is constantly getting more and more complex and adjusted to different very specific environments (distributed software, cloud computing, grid computing, web applications, ...), it is of essential importance to propose more adequate specific metrics. Majority of classical metrics also could be applied in agent technologies and multiagent systems, especially for measuring quality of source code, but specific behavior and functionalities of MAS need very subtle and particular metrics.

In this paper, we will present some existing approaches and propositions for software metrics in the area of agent technologies and multiagent systems. After that we will briefly discuss possibilities and need to include this topic in specific courses devoted to agent technologies and/or software testing in ICT study programs. Our main intention in this paper is just to open this important question as by our knowledge there are no studies that explore possibilities to include agent metrics in study programs.

As it is very delicate area and not we will avoid to precisely suggest topics, parts of courses, methodology of teaching and organization of exams but we will give general directions, suggestions and possible solutions.

The rest of paper is organized as follows. Section 2 gives an overview of classical software metrics that present bases for all other specific software types and their measurements and metrics. Section 3 brings different approaches and propositions of several authors to measure agents and multiagent systems. Section 4 considers several possibilities to introduce agent-oriented metrics in ICT study programs. Last section concludes the paper.
2. “CLASSICAL” SOFTWARE METRICS

Software metrics could be described as process of measuring of specific properties of a segment of software or its specifications. As measurement and metrics are closely related in software engineering they are usually classified in: process measurement through process metrics, product measurement through product metrics, and project measurement through project metrics [Thakur 2016].

2.1 Process Metrics – Process metrics are connected to the quality and effectiveness of software process. They can determine different aspects like: effort required in the process, maturity of the process, effectiveness of defect removal during development, and so on. Some specific process metrics are: Number of errors discovered before the software delivered to users; Defect detected and reported during use of the software after delivery; Time spent to fix discovered errors; Human effort used for development of software; Conformity to schedule; Number of contract modifications; Relation between estimated and actual cost.

2.2 Product Metrics – Product metrics are oriented to the particular software products produced during the phases of software development. They can indicate if a product is developed strictly following the user requirements and involve measuring the properties of the software like the reliability, usability, functionality, efficacy, performance, reusability, portability, cost, size, complexity, and style metrics. These metrics are also known as quality metrics as they measure the complexity of the software design, size or documentation created. Product metrics additionally assess the internal product attributes like: (i) analysis, design, and code model; (ii) effectiveness of test cases; (iii) overall quality of the software under development. Some of important product metrics in the development process are mentioned below [Thakur 2016].

- **Metrics for analysis model**: are connected to various aspects of the analysis model like system functionality, size.
- **Metrics for design model**: are connected to the quality of design like architectural design metrics, component-level design metrics.
- **Metrics for source code**: are connected to source code complexity, maintainability.
- **Metrics for software testing**: are connected to the effectiveness of testing and are devoted to design of efficient and effective test cases.
- **Metrics for maintenance**: are connected to assessment of stability of the software product.

2.3 Project Metrics – These metrics are connected to the project characteristics and its completion, and they could be of great help to project managers to assess their current projects. They enable: to track potential risks, adjust workflow, identify problem areas, and evaluate the project team's abilities. Some characteristic examples of these metrics are: number of software developers, productivity, cost and schedule.

Project measurements that include both product and process metrics are useful for project managers as they help in judging the status of projects so that the teams can react accordingly. Two groups of measures are recognized: Direct and Indirect [Jayanthi and Lilly Florence 2013].

- **Direct Measures**
  - Measured directly in terms of the observed attribute - length of source-code, duration of process, number of defects discovered.
  - Direct Measures (internal attributes) - cost, effort, LOC, speed, memory.

- **Indirect Measures**
  - Calculated from other direct and indirect measures.
  - Indirect Measures (external attributes) - functionality, quality, complexity, efficiency, reliability, maintainability.

Some authors also distinguish several other types of metrics mentioned below.

2.4 Requirements metrics [Costelo and Liu 1995] – are oriented towards different aspects of requirements and usually include: Size of requirements, Traceability, Volatility and Completeness.

2.5 Software package metrics [Kaur and Sharma 2015] – are connected to a variety of aspects of software packages:
- Number of classes and interfaces
- Afferent couplings – number of packages depending on the evaluated package
- Efferent couplings – number of outside packages the package under evaluation depends on
- Abstractness – number of abstract classes in the package
- Instability – represented as a ratio: efferent couplings/total couplings, in range 0-1.
- Distance from main sequence – represents how the package balances between stability and abstractness.
- Package dependency cycles – packages in the package hierarchy that depend on each other.

2.6 Software Maintenance Metrics [Lee and Chang 2013] – Some of very important metrics for the maintenance phase are: Fix backlog and backlog management index; Fix response time and fix responsiveness; Percent delinquent fixes and so on.

2.7 Resources metrics are also well known type of metrics, combined into several groups [Dumke et al. 2000]: personnel metrics that include skill metrics, communication metrics and productivity metrics; software metrics that include paradigm metrics, performance metrics and replacement metrics; hardware metrics that include reliability metrics, performance metrics and availability metrics.

2.8 Metrics related to the efficiency of implementation: as the area of large-scale computing has evolved “outside of mainstream software engineering” it is necessary to introduce and use these specific metrics as well. Metrics related to the efficiency of implementation among others include: raw speed, parallel speedup, parallel efficiency and so on [Paprzycki and Zalewski 1997], [Zalewski and Paprzycki 1997].

2.9 There are some metrics that are connected to customers like Customer Problems Metric and Customer Satisfaction Metrics [Lee and Chang 2013] and they include: Percent of satisfied and completely satisfied customers; Percent of dissatisfied and completely dissatisfied customers; Percent of neutral customers.

In this section we tried to present very briefly essences of an important area of software engineering and software production. This area is in fact a new discipline oriented to software assessment, measurement, and constant development and application of wide range of software metrics. As different kinds of software systems and products are facilitate and support almost all human activities and everyday life, producing appropriate software is one of the most dynamic areas. As a consequence software has to be more and more reliable, safe and secure. Accordingly, for software engineering community, it is important and necessary to continue to develop more accurate ways of measuring very specific software products.

3. METRICS AND THEIR USAGE IN AGENT TECHNOLOGIES AND MULTIAGENT SYSTEMS

Agent technologies and Multiagent System (MAS) are specific software paradigm oriented towards building extensive distributed systems. This paradigm is based on several specific characteristics necessary for modeling, in natural way, very complex systems. Additionally these systems are deployed on variety of computing devices (within mobile ad hoc networks and satellite links) that are based on non-traditional communications ways. These systems are very specific and their development present new challenges for analyzing and describing system performance [Lass et al. 2009], [García-Magarriño et al. 2010].

Regardless of popularity of this paradigm within the research community, majority of developed MASs are still a result of entirely theoretical research and there are very few examples of large scale systems applied in real environments, for more details, see [Ganzha and Jain 2013]. In spite this fact different approaches to applying measurement and more adjusted metrics in the area of agents and MAS have been developing recently. An older approach and description of essential characteristics of agents and multiagent systems and summary of metrics for agent-based systems is given in [Dumke et al. 2000]. In this paper we will be focusing to newest approaches in this area. Some of recent, particular approaches will be presented in the rest of the section.
3.1 METRICS IN AGENT TECHNOLOGIES I.

Some authors claim that measuring the complexity represent a good indicator to control the development of agent-based software and estimate the required effort to maintain it [Marir et al. 2014].

MAS are very complex software systems which abstract representation consists of agents. The agent is supported by an agent framework. Below the framework is the platform that executes on a host i.e. adequate computing device. All together they are incorporated, i.e. distributed in, and interact with the physical environment. According to that measurement can be performed at four layers in the model: agent, framework, platform, and environment/host layers [Lass et al. 2009].

In [Lass et al. 2009], authors recognize a variety of metrics that can be applied at each of the mentioned layers. These metrics may be divided into two groups, according to (i) their effectiveness (or performance), and (ii) types of data they represent.

**Measures of Effectiveness** – are connected to the ability of the system to complete assigned task in an environment.

**Measures of Performance** – describe not the quality of software solution but the quality of obtaining the solution. They represent quantitative measures of secondary performances like resource consumption (power consumed, communications range/time to successfully perform a certain task).

**Data Classification** – here metrics are divided into four categories depending on empirical determinations, mathematical transformations and statistical operations: nominal, ordinal, interval, and ratio. Below we will very briefly explain them, for more details see [Lass et al. 2009].

- **Nominal** measurements are labels assigned to data: equality of objects, number of cases, contingency correlation. This type of measure can determine equality of objects.
- **Ordinal** measurements are rankings or orderings of data described by “less than” or “greater than” property between data.
- **Interval** measurements are similar to ordinal measurements, but the differences between any two measures have meaning. In addition to the ordinal statistical operations, the mean, standard deviation, rank-order correlation and product-moment correlation are acceptable.
- **Ratio** measurements are similar to interval measurements except that there is an absolute zero point. Any statistical operation and multiplying by a constant are allowed.

3.2 METRICS IN AGENT TECHNOLOGIES II.

There are some approach based on the computational complexity of MAS [Dekhtyar et al. 2002], [Dziubiński et al. 2007] while others studied the complexity of MAS code i.e. software complexity [Dospisil 2003], [Klügl 2008].

Specific software complexity metrics are very important to evaluate developed MAS [Marir et al. 2014] and in this subsection we will concentrate on some of them.

The complexity of a software system is an important factor in measuring required effort to understand, verify and maintain the software. In MAS, the number of agents and structure of their interaction are the key factors that influence its complexity. Similarly the complexity of agents has a direct influence on the complexity of MAS.

At the agent-level, complexity can be analyzed according to two orthogonal criteria: the complexity of the agent’s structure and of its behaviors. On the other hand, complexity at the MAS level can be analyzed based on agents’ behavior and social interaction between agents in an environment. Some important metrics for measuring these components are proposed in [Marir et al. 2014] and will be very briefly present in the rest of this subsection. For more details see [Marir et al. 2014].

1. **The structural complexity metrics** - characteristic metrics in this group are enlisted below.

   **The Size of the Agent's Structure** - Different attributes and their combinations are used to specify the state of the agent. This metric is used to calculate the complexity of the agent’s structure.
The Agent’s Structure Granularity - agent’s structure is composed of attributes and some of them could be very complex. Accordingly it is necessary to assess the complex agent structure.

The Dynamicity of the Agent’s Structure - apart from composed attributes, the ones of the agent can be of a container nature (for example attributes allow adding and removing variables like the list). Furthermore, the extensive dynamicity of the agent’s structure causes instability and such dynamicity can be measured by identifying the structure update in two different moments.

2. The behavioral complexity metrics - using different metrics, also the behavioral complexity of the agent can be measured:
   - The Behavioral Size of the Agent - represents the indicator of the degree of agent’s specialization and gives the number of behaviors ensured by the agent.
   - The Average Complexity of Behaviors - generally speaking, an agent with only one complex behavior can be more complex that an agent with several simple behaviors. So, it is possible to calculate the complexity of a behavior using well known cyclomatic metrics [McCabe 1976].
   - The Average Number of Scheduled Behaviors - an agent can launch several behaviors. So in the complex systems, the management of the scheduling different behaviors is not an easy task. In [Marir et al. 2014], authors proposed an indicator of the concurrency intra-agent. They calculate it as average number of behaviors scheduled in each moment.

3. The social structure complexity metrics - a set of agents that exist in an environment represent a MAS so the complexity of the social structure of MAS can be measured in different ways.
   - The Heterogeneity of Agents - this metric indicates the number of classes of agents, MAS composed of heterogeneity agents is more difficult than a homogeny MAS.
   - The Heterogeneity of the Environment’s Objects - the existence of heterogeneous objects in the environment is in accordance to the complexity of the MAS.
   - The Size of the Agent’s Population - in the MAS, the agents can be created and destroyed dynamically. The complexity of the MAS is increasing by increasing the number of agents.

4. The interactional complexity metrics - the interaction between agents is essential and it also can be measured by different metrics.
   - The Rate of Interaction’s Code - this metric presents the rate of source code devoted to ensure the communication between agents. It gives only partial information about the collective behavior of agents as sent messages could be repeated.
   - The Average Number of Messages Exchanged per Agent - as previously mentioned metrics has drawbacks this metrics estimates the real number of exchanged messages. This metric is of dynamic nature and calculates required effort to understand the collective behavior of the MAS.
   - The Rate of the Environment’s Accessibility – the agents can be interacting indirectly by manipulating the environment’s objects. So, this static metrics shows the complexity of the MAS because of the existence of public attributes increases the agents coupling.

Proposed metrics can be also combined with, above mentioned, classical metrics (like the size in the Lines of Code metric), in order to provide more information about the complexity of agent-based software. Also it is possible to associate to each metric a weight, which reflects its importance. The appropriate weights are left to the appreciation of the users of the proposed metrics.

3.3 METRICS IN AGENT TECHNOLOGIES III.

In [Homayoun Far and Wanyama 2003], authors proposed several metrics based on the MAS complexity approached in objective and subjective way.

Subjective metrics - Subjective complexity is oriented to a human user who evaluates the complexity of the agent system. One proposition is to use a modified version of the Function Point method [Albrecht and Gaffney 1983] that accounts for algorithmic complexity. Important parameters involved in the model for each agent are: external inputs and outputs, external inquiries, external interfaces, internal data structures, algorithmic complexity and knowledge complexity factor. The overall complexity of the MAS is the mean of the adjusted MAS function point of its constituent agents.
Objective metrics - Objective complexity is based on complexity, seen as an internal property of the agent system. In this case, the cyclomatic complexity metrics can be used if the MAS system is nearly-decomposable. Complexity of the MAS is the sum of cyclomatic complexity of its constituent agents. For nearly-decomposability, the communicative cohesion metrics can be examined. The communicative cohesion metrics for an agent is defined in terms of the ratio of internal relationships (inter-actions) to the total number of relationships (i.e., sum of inter-actions and intra-actions).

3.4 METRICS IN AGENT TECHNOLOGIES IV.
In [Klügl 2008], the author distinguishes between overall system-level metrics that are relevant for the complete model, but also for the agent level metrics. Usually, agent metrics is oriented to measuring the population and the environmental complexity. However, interactions of agents are extremely interesting and are worth of being measured. Numerous metrics for system-level complexity are mentioned below [Klügl 2008].

- **Number of agent types** - is a measure for heterogeneity of the model i.e. it equals the number of agents (based on the number of classes).
- **Number of resource types** - is similar to the previous metrics, but for passive entities in the environment.
- **Maximum number of agents** - is metrics that represent the maximum number of agents concurrently present during a simulation run. However, there is a conceptual problem when the maximum number is only adopted at the beginning of the simulation.
- **Minimum number of resources** - is measure similar to the maximum number of agents and only actually used resources should be counted.
- **Maximum delta of agent population** - determines the variability of population numbers over a given interval of time. In fact it forms the rate of population change.
- **Maximum delta of resource population** - is the analogue to the previous metrics.
- **Agent-resource relation** - is the number of agents divided by the number of resources.
- **Number of agent shapes** - This is a measure for spatial complexity and represents different geometries that agents may possess.
- **Number of resource shapes** - is the similar to the previous metrics and represents number of different geometries that occur in the set of resources.
- **Maximum resource status** - Size Resources may be of differently complexity. This metrics counts the maximum number of status variables that resource may possess.
- **Maximum resource parameter** - This metrics computes the maximum number of parameters that influence the values of the status variables.

Abovementioned metrics are oriented to measuring population of agents and environmental complexity. Following group of metrics is oriented to measuring characteristics of individual agents.

- **Architectural complexity** - This is a measure of the agent architecture but indicators for it are not obvious. Authors of [Klügl 2008] proposed to rank the architectures into three categories based on their complexity and use it as a metrics: Behavior-describing architectures, Behavior-configuring architectures and Behavior-generating architectures.
- **Action plasticity metric** - Plasticity denotes the potential adaptivity of the behavior in reaction to the environmental influences. This metrics achieves full power when it is combined with additional measures concerning the behavioral plasticity and variability.
- **Size of procedural knowledge** - This metrics is also related to behavior plasticity. It is oriented to the size of the procedural knowledge that is available for the agent.
- **Number of cognitive rules** - This metrics is based on cognitive rules i.e. concept of sharing actions that affect the internal status or beliefs of an agent.
Above mentioned measures can be computed based on static model code. As interactions between agents are usually very dynamic, the values of metrics in the following group can be determined for agent-based simulations [Klügl 2008] during a simulation run.

- **Sum of public information items** - This measure is about the size of external interfaces and it represents the number of concurrently publicly accessible variables i.e. information items.

- **Number of external accesses** - This metrics is basically an abstraction from some message counting metrics. Here it is interesting to take care of how often external data is accessed by the agent in its behavior definition as addition to the number of available information units.

- **Number of agent references (NAR)** - This metrics represents the mean number of agents contained in one agents’ internal model i.e. it addresses the coherence of the agent system. As this value may be varying over time, they can take value between NAR-mean and NAR-stdev but also minimum and maximum number of references as well as the time-related delta of these values.

- **Number resource references** - This metrics represent the number of references of an agent that it holds towards addressing resources.

- **Number of mobility actions** - This metrics represents the number of move actions per agent per time unit. This metric is only useful when there is an actual map where the agents may change their local position.

### 3.5 METRICS IN AGENT TECHNOLOGIES V.

Efficiency-oriented metrics are presented in [Chmiel et al. 2004], [Chmiel et al. 2005]. In these studies MASs were approached from the efficiency of implementation perspective. Here, among others, the following metrics have been experimentally evaluated – efficiency of: message exchanges, agent mobility, database access, agent creation and destruction, and combinations of these.

These metrics have been selected as they represent key characteristics of MAS that are actually being executed in a distributed environment.

The above mentioned metrics, in agent technologies cover a variety of relevant aspects. Although the authors proposed a wide range of appropriate metrics this area is still in constant development.

### 4. AGENT METRICS - POSSIBLE EDUCATIONAL USAGE

Empirical software engineering is one of the important directions in the software engineering approached as a discipline of knowledge. In this discipline, empirical methods are used to evaluate and develop wide range of tools and techniques. Recently, authors of [Birna van Riemsdijk 2012], proposed the use of empirical methods to advance the area of agent programming. Introducing systematic methodologies and qualitative measuring can establish multi-agent systems as a mature and recognized software engineering paradigm. Such methods could help in clear identification of advantages and application domains.

Furthermore, an essential question could be raised: is it necessary or could it be useful to introduce such specific topics in appropriate educational settings and ICT study programs? We can propose three possible ways of inclusion in educational processes specific empirical methods, measurements and metrics for agents and multi-agent systems.

- **Possibility 1** – Introduce brief, specific subtopic on quality measures and different metrics for MAS in regular, classical Software engineering courses. As it is usually very complex course that encompasses a lot of topics and practical work the idea would be just to give theoretical introduction of metrics for MAS and not to ask students for additional practical exercises.

- **Possibility 2** - One of interesting possibilities to apply quality measures and different metrics for MAS could address agent-oriented PhD theses. Usually PhD theses in area of agent technologies include implementation of a prototype or even a real world application. Some of them produce huge amount of code and high-quality software implementations. So application of different metrics in agent technologies and multiagent systems could be used for assessment of quality (or, at least, important characteristics) of these agent-oriented systems.
Some comparison between classical and MAS metrics would brought additional quality of theses.

**Possibility 3** : Another interesting possibility to use quality measures and different metrics for MAS could be incorporation of several specific topics in some of existing courses within ICT study programs.

Usually, within software engineering and/or ICT master study programs, there is a course devoted to software testing techniques. There are also some study programs that include agent technology course as an elective, or as a seminar course [Badica et al. 2014]. The one of motivations for this proposition comes also from positive experiences our colleagues from Poland have had in a "Software agents, agent systems and applications" course, offered for upper level undergraduate and first year MS students at the Warsaw University of Technology. They used specific experiments [Chmiel et al. 2004], [Chmiel et al. 2005] in order to design innovative homework exercises. During the course, students work in groups on semester-long projects. Their earlier experience indicated that students do not pay enough attention to the material delivered in-class, so they augmented it with homework exercises. Designing these activities, lecturers took into account the fact that, on the one hand, the key aspects of agent systems are messaging and mobility, on the other, students have only a limited amount of time, especially those who are close to graduating and work on their final projects. Therefore, they have designed two homework assignments. The first of them was similar to the "spamming test", while the other followed the "relay race" experiments. For each of them students had to implement a demostrator agent system (using JADE agent platform), perform series of experiments on multiple computers, and write a report on their findings. Lecturers came to quite interesting results. First, as expected, students have found that there is a direct relationship between the "quality of the code" and the efficiency of the developed system. For some of the students this was a real eye-opener, as execution time is rarely something that much attention is paid to. Second, they have found the JADE agent platform is quite robust. However, they have reported some issues when running it using wireless connections. Fourth, writing reports is an issue that is not paid enough attention to, during CS education and the proposed activities attempt at overcoming this shortcoming (see, also, [Paprzycki and Zalewski 1995]). Finally, it is definitely a valuable pedagogical addition to the agent systems course and leaves a room for introduction of some elements of agent systems measurements and application of some specific metrics.

To conclude, for specific agent oriented courses (nevertheless mandatory or elective), it is essential to devote important part of the course to agent and MAS measurements and metrics, to compare and emphasize similarities and differences between them and classical metrics. Also in such courses is necessary to organize appropriate practical tasks and exercises. One possibility could be to give students a source code of MAS and ask them to perform different metrics and make comprehensive analysis and comparison of obtained results.

Recently agent technology becomes more and more important in realization of distributed, real-life very complex systems. So, students have to be familiar with such important technology and paradigm which they will probably use in their future jobs. So it is necessary to give them some at least basic insights in measuring software quality and application of appropriate metrics in agent systems. Courses devoted to agent technology or software testing represent good opportunity to introduce students with such specific and important topics.

### 5. CONCLUSION

Assessment and measurement of all phases, aspects, and activities of software development and final products is an old but still dynamic discipline and area of research. Nowadays there are a lot of developed and proposed metrics. Different types of software products with their specific
characteristics require development and application of more appropriate metrics. It is evidently that there is even a number of metrics devoted to software agents and MASs.

Having in mind discussion presented in this papers it is a little bit strange that in study programs and in higher education generally we do not have appropriate topics of this, neither in Software engineering, Software testing, nor Agent-oriented courses. This is surely a mistake that should be considered and resolved and in the paper we try to propose some ways and possibilities of doing this.

ACKNOWLEDGMENT

This paper is a part of the Serbia-Romania-Poland collaboration within agreement on “Agent systems and applications.”

REFERENCES


DOI: http://dx.doi.org/10.1145/1774088.1774278


DOI: http://dx.doi.org/10.14257/ijhit.2015.8.12.15


Why Information Systems Modelling Is Difficult

HANNU JAAKKOLA, Tampere University of Technology, Finland
JAAK HENNO, Tallinn University of Technology, Estonia
TATJANA WELZER DRUŽOVEC, University of Maribor, Slovenia
BERNHARD THALHEIM, Christian Albrechts University Kiel, Germany
JUKKA MÄKELÄ, University of Lapland, Finland

The purpose of Information Systems (IS) modelling is to support the development process through all phases. On the one hand, models represent the real-world phenomena – processes and structures – in the Information System world and, on the other hand, they transfer design knowledge between team members and between development phases. According to several studies there are reasons for failed software projects in very early phases, mostly in bad quality software requirements’ acquisition and analyze, as well as in lacking design. The costs of errors are also growing fast along the software life cycle. Errors made in software requirements analyze are increasing costs by the multiplying factor 3 in each phase. This means that the effort needed to correct them in the design phase is 3 times, in the implementation phase 9 times and in system tests 27 times more expensive than if they would be corrected at the error source; that means in the software requirements analyze. This also points out the importance of inspections and tests. Because the reasons for errors in the requirements phase are in lacking requirements (acquisition, analyze) which are the basis of IS modelling, our aim in this paper is to open the discussion on the question “Why is Information Systems modelling difficult?” The paper is based on teachers’ experiences in Software Engineering (SE) classes. The paper focuses on the modelling problems on the general level. The aim is to provide means for the reader to take these into account in the teaching of IS modelling.

Categories and Subject Descriptors: D [Software]: D.2 [Software Engineering]: D.2.1 [Requirements / Specifications]; D 2.9 [Management]; H [Information Systems]: H.1 [Models and Principles]; H.1.0 [General]

General Terms: Software Engineering; Teaching Software Engineering, Information Systems, Modelling

Additional Key Words and Phrases: Software, Program, Software development

1. INTRODUCTION

The purpose of Information Systems (IS) modelling is to establish a joint view of the system under development; this should cover the needs of all relevant interest groups and all evolution steps of the system. The modelling covers two aspects related to the system under development – static and dynamic. A conceptual model is the first step in static modelling; it is completed by the operations describing the functionality of the system. These are, along the development life cycles, cultivated further to represent the view needed to describe the decisions made in every evolution step from recognizing the business needs until the final system tests and deployment. The conceptual model represents the relevant concepts and their dependences in the terms of the real-world. Further, these concepts are transferred to IS concepts on different levels.

The paper first focuses in the basic principles related to IS modelling. The topics selected are based on our findings in teaching IS modelling. The list of topics covers the aspects that we have seen as difficult to understand by the students. The following aspects are covered: Variety of roles and communication (Section 2), big picture of Information Systems development (Section 3), role of abstractions and views (Section 4), characteristics of the development steps and processes (Section 5), varying concept of concept (Section 6) and need for restructuring and refactoring after IS deployment (Section 7). Section 8 concludes the paper.

These different points of view give – at least partial – answers to our research problems: Why Information Systems modelling is difficult to teach? Why this topic is important to handle? In our

Authors’ addresses: Hannu Jaakkola, Tampere University of Technology, Pori, Finland, hannu.jaakkola@tut.fi; Jaak Henno, Tallinn University of Technology, Tallinn, Estonia, jaak.henno@ttu.ee; Tatjana Welzer-Družovec, University of Maribor, Maribor, Slovenia, tatjana.welzer@um.si; Jukka Mäkelä, University of Lapland, Rovaniemi, Finland, jukka.makela@ulapland.fi; Bernhard Thalheim, Christian Albrechts University Kiel, Germany, thalheim@is.informatik.uni-kiel.de.

Copyright © by the paper’s authors. Copying permitted only for private and academic purposes.

work we recognized problems in learning the principles of Information Systems modelling. If these problems are not understood, the software engineers’ skills are not at the appropriate level in industry. The paper could also be understood as a short version of the main lessons in Software Engineering (SE).

2. UNDERSTANDING THE ROLES AND COMMUNICATION

Software development is based on communication intensive collaboration. The communication covers a variety of aspects: Communication between development team members in the same development phase, communication between development teams in the transfer from one development phase to the next one, and communication between a (wide) variety of interest groups. The authors have handled the problems related to collaboration in their paper [Jaakkola et al. 2015]. Figure 1 is adopted from this paper.

![Fig. 1. Degrees of collaboration complexity [Jaakkola et al. 2015].](image)

The elements in Fig. 1 cover different collaboration parties (individual, team, collaborative teams (in the cloud), collaboration between collaborative teams (cloud of clouds) and unknown collaboration party (question mark cloud). The collaboration situations are marked with bidirectional arrows. Without going into the details (of the earlier paper) the main message of the figure is the fast growing complexity in collaboration situations \((1^*1; 1^*n'; nk^n'k^*m')\). In increasing amounts there are also unknown parties (question mark cloud; e.g. in IS development for global web use), which increases the complexity. The explicit or implicit (expected needs of unknown parties) communication is based on messages transferred between parties. Interpretation of the message is context-sensitive (i.e., in different contexts the interpretation may vary). The message itself is a construction of concepts. The conceptual model represents the structure of concepts from an individual collaborator’s point of view. An important source of misunderstanding and problems in collaboration is an inability to interact with conceptual models.

In this paper we concentrate on two important roles – the Systems Analysts and the customer (variety of roles). The starting point is that the Systems Analysts are educated in ICT Curricula and they should have a deep understanding of the opportunities provided by ICT in business processes. The customer should present the deep understanding of the application area instead, and they are not expected to be ICT experts. What about the Systems Analyst – should he/she also be expert in ICT
applications? We will leave the exact answer to this question open. Our opinion is that, first and foremost, the Systems Analyst should be a model builder who is filtering the customer’s needs and, based on abstractions, finally establishes a baseline as a joint view – from the point of view of all interest groups - to the system under development. The joint view is based on communication between different parties. The Standish Group has reported communication problems between Systems Analysts and users - lack of user involvement – to be one of the important sources of IS project failures (Chaos report [Standish Group 2016]).

3. UNDERSTANDING THE BIG PICTURE OF MODELLING

Information System development is based on two different views, the static one and the dynamic one, having a parallel evolution path. All this must be recognized as a whole already at the beginning, including the evolution of requirements through the development life cycle. Figure 2 illustrates flow in the “big picture” of modelling. In the upper level of IS development the approach always follows the principles of a “plan driven” approach, even in the cases where the final work is based on Agile or lean development.

In this paper we do not focus on the discussion of the current trends in software development models. The traditional plan-driven (waterfall model based) approach is used. It is an illustrative way to concretize the basic principles of the constructive approach in software development. The same principles fit in all approaches, from plan-driven (waterfall based) to agile, lean, component based, software reuse based etc. approaches. According to Figure 2 the Information System development has its roots in business processes (understanding and modelling). Business processes represent the dynamic approach to the system development, but also provide the means for the preliminary concept recognition and the operations needed to handle them. The conceptual model is a static structure describing the essential concepts and their relationships. The Information System development continues further by the specification of the system properties (to define the system borders in the form of external dependencies) and transfers the real-world concepts first into the requirement level, and further to the architecture and implementation level concepts. Separation of the structure and behavior is not always easy; people are used to describing behavior by static terms (concepts) and static state by dynamic terms (concepts).

The role of “work product repository” is not always recognized. The development flow produces necessary work products, which are used by other parts of the development flow. Conformity between work products must be guaranteed, but is not always understood clearly. Conformity problems, both
in the horizontal (evolution path of work products) and vertical (dynamic vs. static properties) direction are typical.

4. UNDERSTANDING THE ROLE OF ABSTRACTIONS AND VIEWS

The IS development is based on abstractions – finding the essence of the system under development. Figure 3 illustrates the role of abstractions in Information Systems modelling.

![Diagram showing the role of abstractions](image)

Fig. 3. The role of abstractions [Koskimies 2000; modified by the authors].

The Information System is the representative of the real-world (business) processes in the “system world”. The model (set) of Information System describes the real-world from different points of view (viewpoint) and a single model (in the terms of UML: Class diagram, state diagram, sequence diagram, ...) provides a single view to certain system properties. Information System is an abstraction of the real-world covering such structure and functionality that fills the requirements set to the Information System. Such real-world properties that are not included in the Information System are represented by the external connections of it or excluded from the system implementation (based on abstraction). As seen in Figure 3, the starting point of the model is in the real-world processes, which are partially modelled (abstraction) according to the selected modelling principles; both the static and dynamic parts are covered. The individual models are overlapping, as well as the properties in the real-world (processes). This establishes a need for checking the conformity between individual models; this is not easy to recognize. An additional problem related to abstractions is to find the answer to the question “What should be modelled?” and “How to fill the gaps not included in the models?”. No clear answer can be given. However, usually the problems in Information Systems relate more to the features that are not modelled than to those that are included in the models. Models make things visible, even in the case that they include some lacks and errors (which are also becoming visible this way).

The Information System development covers a variety of viewpoints to the system under development. Structuring the viewpoints helps to manage all the details of the Information System related data as well as the dependences between these. In this context, we satisfy by referring to the widely used 4+1 View model introduced originally by Kruchten [Kruchten 1995], because it is referred to widely and was also adopted by the Rational Unified Process specification.
The aim of the 4+1 view model (Figure 4) is to simplify the complexity related to the different views needed to cover all the aspects in Information Systems’ development; the relations between different views are not always clear. Views serve different needs: A logical view provides necessary information for a variety of interest groups, a development view for the software developers, a physical view for the system engineers transferring the software to the platforms used in implementation, and the process view to the variety of roles responsible for the final software implementation. Managing the conformity between the variety of views (models) is challenging. Again, to concretize the role of views in Information Systems modelling, we will bind them to UML (static path related) specifications: Logical view – the main artefact is a class diagram; development view – the main artefact is a component diagram; physical view – the main artefact is a deployment diagram; process view - the artefacts cover a variety of communication and timing diagrams. Dynamic path decisions are specified by a variety of specifications, like state charts, activity diagrams, sequence diagrams and timing descriptions.

One detail not discussed above is the role of non-functional (quality) properties, assumptions and limitations. Without going to the details, we state that they are changing along the development work to functionality, system architecture, a part of the development process, or stay as they are to be verified and validated in qualitative manner.

5. UNDERSTANDING THE CHARACTERISTICS OF THE DEVELOPMENT PATH AND PROCESSES

The purpose of the Information Systems development life cycle models is to make the development flow visible and to provide rational steps to the developer to follow in systems development. There exists a wide variety of life cycle models – from the waterfall model (from the 1960s) as the original one to the different variants of it (iterative – e.g. Boehm’s spiral model), incremental, V-model and, further, to the approaches following different development philosophies (e.g. Agile, Lean); see e.g. [Sommerville 2016]. As already noted above, our aim is not to go in detailed discussion of development models. All of them represent in their own way a model of constructive problem solving, having a more or less similar kernel with different application principles.

We selected the V-model to illustrate the development path for two reasons. The origin of the V-model is in the middle of 1980s. In the same issue, both Rook [Rook 1986] and Wingrove [Wingrove1986] published its first version, which has since been adopted by the software industry as the main process model for traditional (plan-driven) software development. Firstly, it separates clearly the decomposition part (top-down design) and composition part (bottom-up design) in the system evolution, and, secondly, it shows dependences between the early (design) and late (test) steps. An additional feature, discussed in the next Section, relates to the evolution of the concept of concept along the development path.
The development activity starts (Figure 5; see also Figure 2) from business use cases (processes) that are further cultivated towards user requirements (functionality) and the corresponding static structure. In the top down direction (left side) the system structure evolution starts from conceptual modelling in the terms of the real-world. These are transferred further to the structures representing the requirements set to the Information System (in terms of the requirements specification). Architecture design modifies this structure to fill the requirements of the selected architecture (in terms of the architecture) having focus especially on the external interfaces of the system. The detailed design reflects the implementation principles, including interfaces between system components and their internal responsibilities. Implementation ends the top-down design part of the system development and starts the bottom-up design. The goal of the bottom-up design is to collect the individual system elements and transfer them to the higher level abstractions, first to components (collection of closely related individual elements – in terms of the UML classes) and further to the nodes, which are deployable sub-systems executed by the networked devices. The bottom-up modelling includes the sketching and finalizing phases. An additional degree of difficulty in this “from top-down to bottom-up” elaboration is its iterative character; the progress is not straightforward, but iterative, and includes both directions in turn.

6. UNDERSTANDING THE VARYING CONCEPT OF CONCEPT

Along the development path the abstraction level of the system is changing. This reflects also in the used terminology. This is illustrated in Figure 5’s middle part – concept evolution. In the beginning of the development work the modelling is based on the real-world concepts (conceptual model); this terminology is also used in communication between the Systems Analyst and different interest groups. As a part of requirements specification these concepts are transferred to fill the needs of system requirement specification. The terminology (concepts used) represents the requirements level concepts, which do not have (necessarily) 1-1 relation. In architecture design the concepts related to architecture decisions become dominant – i.e. the role of design patterns and architecture style become important. This may also mean that, instead of single concept elements, the communication is based on compound concepts. In practice this may mean that, instead of single elementary concepts (class diagram elements), it becomes more relevant to communicate in the terms of design patterns (observer-triangle, proxy triangle, mediator pair, factory pair, etc.) or in the terms of architecture style (MVC solution, layers, client-server solution, data repository solution). The implementation phase...
brings the need for programing level concepts (idioms, reusable assets, etc.). To summarize the discussion, the communication is based on different concepts in different parts of the development life cycle – we call it the evolution of concepts.

7. PROACTIVE MODELLING - STRUCTURAL AND CONCEPTUAL REFACTOERING

Programs model real-life systems and are designed for real, currently existing computer hardware. But our real-life – our customs, habits, business practices and hardware are changing rapidly and our computerized systems should reflect these changes in order to perform their tasks better. Thus, software development is never finished – software should be modified and improved constantly and, therefore, should be designed in order to allow changes in the future. Because of that the design should take into account the need for future changes in a proactive manner; otherwise the changes become expensive and difficult to implement and cause quality problems. Proactive modelling is based on the use of interfaces instead of fixed structures, modifiable patterns in design, generalized concepts and inheritance instead of fixed concepts, the use of loose dependencies instead of strong ones, extra complexity in concept to concept relations, etc.

The most common are changes in program structure - structural refactoring, applying a series of (generally small) transformations, which all preserve a program's functionality, but improve the program's design structure and make it easier to read and understand. Programers’ folklore has many names and indices for program sub-structures (design smells), which should be reorganized or removed: Object abusers (incomplete or incorrect application of object-oriented programing principles), bloaters (overspecification of code with features which nobody uses, e.g. Microsoft code has often been called 'bloatware' or 'crapware'), code knots (code which depends on many other places of code elsewhere, so that if something should be changed in one place in your code you have to make many changes in other places too, so that program maintenance becomes much more complicated and expensive). Structural refactoring generally does not change programs’ conceptual meaning, thus, in principle, it may be done (half)-automatically and many methods and tools have been developed for structural refactoring [Fowler 1999; Kerievsky 2004; Martin 2008].

Cases of conceptual refactoring are much more complicated. Our habits and behavior patterns change constantly; we are using new technology that was not used commonly at the time of program design, i.e. when the conceptual model was created; increased competition is forcing new business practices; etc. All these changes should also be reflected in already introduced programs and, generally, they also require re-conceptualization of the programs or some parts of them. We will clarify this in the following examples below.

Microsoft, who have often been accused of coupling useful programs (e.g. the Windows OS) with bloatware and crapware, introduced in 2012 a special new service "Signature Upgrade" for "cleaning" up a new PC – you bring your Windows PC to a Microsoft retail store and for $99 Microsoft technicians remove the junk – a new twist in the Microsoft business model.

An even bigger change in the conceptual model of Microsoft's business practices occurred when Microsoft introduced Windows 10. With all the previous versions of the Windows OS Microsoft has been very keen on trying to maximize the income from sales of the program, thus the OS included the subsystem "Genuine Windows" which has to check that the OS is not a pirated copy but a genuine Microsoft product (but quite often also raised the alert "This is not a Genuine Windows! " in absolutely genuine installations). With Windows 10 Microsoft changed by 180° the conceptual model of monetizing – it became possible to download and install Windows 10 free of charge! Even more, Microsoft started to foist Windows 10 intensely onto all users of Windows PC and, for this, even changed the commonly accepted functionality of some screen elements: in all applications clicking the small X in windows upper right corner closes the window and its application but, contrary to decades of practice in windowed User Interfaces (UIs) and normal user expectations, Microsoft equated closing the window with approving the scheduled upgrade – this click started the (irreversible) installation of Windows 10. This forced change in the conceptual meaning of a common screen element proved to be wrong and disastrous to Microsoft. A forced Windows 10 upgrade rendered the computer of a
H. Jaakkola, J. Henno, T. Welzer Družovec, J. Mäkelä, B. Thalheim

Californian PC user unusable. When the user could not get help from Microsoft’s Customer Support, she took the company to court, won the case and received a $10,000 settlement from Microsoft; Microsoft even dropped its appeal [Betanews 2016]. The change in the company’s conceptual business policies has created a lot of criticism for Microsoft [Infoword 2016].

Many changes in conceptual models of software are caused by changes in the habits and common practices of clients which, in turn, are caused by the improved technology they use. Once functioning of many public services was based on a living queue – the customer/client arrived, established their place in the queue and waited for his/her turn to be served. In [Robinson 2010] a case of conceptual modelling is described for designing a new hospital; a key question was: “How many consultation rooms are required”? The designer’s approach was based on data from current practice: “Patient arrivals were based on the busiest period of the week – a Monday morning. All patients scheduled to arrive for each clinic, on a typical Monday, arrived into the model at the start of the simulation run, that is, 9.00am. For this model (Fig. 6a) we were not concerned with waiting time, so it was not necessary to model when exactly a patient arrived, only the number that arrived”.

This approach of conceptual modelling of a hospital’s practice ignores totally the communication possibilities of patients. In most European countries, computers and mobile phones are widespread and used in communication between service providers and service customers, and this communication environment should also be included in the conceptual model of servicing. Nowadays, hospitals and other offices servicing many customers mostly all have on-line reservation systems, which allow customers to reserve a time for visit and not to rush with the requirement to reserve a time for the visit on Monday morning or staying in the living queue. A new attribute, Reservation, has been added to the customer object. The current reservation system is illustrated in Fig. 6b.

![Fig. 6. The conceptual model of mass service (Robinson 2010): (a) In 2010 (Robinson 2010), (b) Nowadays.](image)

Cultural/age differences can cause different variations of the conceptual model of reservation systems. For instance, in Tallinn with its large part of older technically not proficient population (sometimes also non-Estonian, i.e. have language problems) for some other public services (e.g. obtaining/prolonging passports, obtaining of all kind of permissions/licenses) the practice of reserving time has not yet become common. In the Tallinn Passport Office (https://www.politsei.ee/en/) everyone can make a reservation for a suitable time [Reservation System (2016)], but many older persons still appear without one. In the office customers with reservations are served without delay, but those who do not have a reservation are served in order of appearance, which sometimes means hours of waiting. Seeing how quickly customers with reservations are served is a strong lesson for them – here the conceptually (new for them) system of reservations does not only change the practice of office, but also
teaches them new practices, i.e. here, innovation in technology (the Reservation System) also changes the conceptual practices of customers.

Practical use of a reservations system sometimes also forces changes to the system itself. For instance, most of the doctors in Estonia, Finland and Slovenia work with reserved times. However, sometimes it happens that a customer who has a reserved time is not able to come. Medical offices require cancellation (some even practice a small fine if cancellation is not done in-time). In order to find a replacement, the office should be able to contact potential customers (who have a reservation for some future time). Thus, two more fields were introduced to the object model of the customer: Mobile phone number, Minimal time required to appear at the service. A new functionality was also added to the reservation system: if somebody cancels, the reservation system compiles a list of potential 'replacement' customers, i.e. customers, who have a future reservation and are able to appear at the service provider in time and the office starts calling them in order to agree a new reservation.

8. CONCLUSION

There is a lot of evidence that the most serious mistakes are made in the early phases of software projects. Savolainen [Savolainen 2011] reports in her Thesis and studies (based on the analyze of tens of failed software project data) that, in almost all the studied cases, it was possible to indicate the failure already before the first steps of the software project (pre-phases, in which the base for the project was built in collaboration between the software company and customer organization). The errors made in early phases are tend to accumulate in later phases and cause a lot of rework. Because of that, the early phase IS models have high importance to guarantee the success of IS projects. The Standish Group Chaos Reports cover a wide (annual) analyze of problems related to software projects. The article of Hastie & Wojewoda [Hastie & Wojewoda 2015] analyzes the figures of the Chaos Report from the year 2015 (Figure 7). The Chaos Report classifies the success of software projects in three categories: Successful, challenged and failed. The share of failed projects (new definition of success factors covers the elements on time, on budget with a satisfactory result) has been stable on the level a bit below 20% (Figure 7, left side). The suitability of the Agile process approach seems also to be one indication for success in all project categories – even in small size projects. The Report has also analyzed the reasons on the background of the success (100 points divided): Executive sponsorship (15), emotional maturity (25), user involvement (15), optimization (15), skilled resources (10), standard architecture (8), agile process (7), modest execution (6), project management expertise (5) and clear business objectives (4).

<table>
<thead>
<tr>
<th>Agile vs. Waterfall</th>
<th>Size</th>
<th>Method</th>
<th>Successful</th>
<th>Challenged</th>
<th>Failed</th>
</tr>
</thead>
<tbody>
<tr>
<td>All projects</td>
<td>Agile</td>
<td>39 %</td>
<td>52 %</td>
<td>9 %</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Waterfall</td>
<td>11 %</td>
<td>60 %</td>
<td>29 %</td>
<td></td>
</tr>
<tr>
<td>Large size projects</td>
<td>Agile</td>
<td>18 %</td>
<td>59 %</td>
<td>23 %</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Waterfall</td>
<td>3 %</td>
<td>55 %</td>
<td>42 %</td>
<td></td>
</tr>
<tr>
<td>Medium size projects</td>
<td>Agile</td>
<td>27 %</td>
<td>62 %</td>
<td>11 %</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Waterfall</td>
<td>7 %</td>
<td>68 %</td>
<td>25 %</td>
<td></td>
</tr>
<tr>
<td>Small size projects</td>
<td>Agile</td>
<td>58 %</td>
<td>38 %</td>
<td>4 %</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Waterfall</td>
<td>44 %</td>
<td>45 %</td>
<td>11 %</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 7. The success of software projects in 2011-2015 [based on Hastie & Wojewoda 2015]).

The Agile vs. Waterfall results are opposite to the ICSE Conference presentation of Barry Boehm (2006; 2006a) related to the software engineering paradigms. According to him, the Agile approach is best suited to small projects that are non-critical and include high dynamics in requirement changes, implemented by skilled people in organizations used to chaos.
The purpose of our paper has been to point out important aspects related to IS modelling. The following aspects are discussed:

- The variety of roles and their responsibilities in IS development;
- Understanding the big picture of the modelling is difficult;
- Understanding the abstractions is difficult;
- Understanding the role of the development path phases and their interrelations is difficult;
- Concepts are varying along the development life cycles;
- Understanding the views and the development flow is difficult.

In teaching IS modelling all this must be taken into account. The experience based realizations of these main problem categories cover at least the following aspects:

- Inadequate skills in using modelling languages – used in an incorrect way (e.g. including dynamic features in static diagrams – functionality in class diagrams; problems to understand what is a class and what is the connection between classes);
- Low motivation to make modelling – preference given to implementation and coding without modelling;
- In a teaching context, we never have the opportunity to solve real modelling problems; small sub-problems instead;
- Difficulties in understanding what is a dynamic and what is a static concept;
- Models are not complete descriptions of the system – what to leave out and what to include; what are essential concepts and what are not; how to fill the gaps;
- Business rules are difficult to understand, because students do not know the real application environment;
- Missing motivation to learn new approaches – “I already know” – syndrome;
- Expectations of the existing skills – in reality reset and relearn is needed because of the “antipattern” type of behavior;
- Models are overlapping and it is difficult to get the different views to conform;
- Models belonging to different abstraction levels are using different concepts – mismatch of conceptual thinking.

Our results are in favor with the analyze and studies discussed above. It is important to benchmark the existing studies and to transfer the “lessons learned” into study modules. The problem also is the fact that the IS modelling is a by-product as a part of other teaching topics and, finally, practicing it remains on the artificial level instead of focusing on the modelling of real (large) information systems. Our original aim was also to handle the topic “How to teach IS modelling?”, but this will be left for further studies that apply the findings of this paper in curriculum and course implementation design in different educational environments. Nevertheless, the authors, as well as other readers, have an opportunity that, on the basis of the research done introduces some innovative steps and solutions out of the box in their own teaching process what will be, together with the reached experiences, an added value to the present paper for the further studies. We also expect a valuable contribution from the discussion at the conference, while changing of the curricula and its implementation is always a demanding step.

REFERENCES


Why Information System Modelling is Difficult


Pharmaceutical Software Quality Assurance System

BOJANA KOTESKA and ANASTAS MISHEV, University SS. Cyril and Methodius, Faculty of Computer Science and Engineering, Skopje
LJUPCO PEJOV, University SS. Cyril and Methodius, Faculty of Natural Science and Mathematics, Skopje

The risk-based nature of the pharmaceutical computer software puts it in a critical software category which imposes a must quality assurance and careful testing. This paper presents the architecture and data model of a quality assurance system for computer software solutions in pharmaceutical industry. Its main goal is to provide an online cloud solution with increased storage capacity and full time authorized access for quality checking of the developed software functionalities. The system corresponds to the requirements of the existing standards and protocols for pharmaceutical software quality. This system aims to ease the process of pharmaceutical software quality assurance process and automate the document generation required for quality document evidence.

Categories and Subject Descriptors: D.2.4 [Software Engineering]: Software/Program Verification — validation; D.2.11 [Software Architectures] Domain-specific Architecture

General Terms: System Architecture
Additional Key Words and Phrases: Quality assurance system, pharmacy, verification.

1. INTRODUCTION

Life science companies are obligated to follow strict procedures when developing computer software. For example, computer software designed for food or drug manufacturing process must fulfill strict quality requirements during the software development life cycle. In order to prove that the developed software meets the quality criteria, companies must deliver documented evidence which confirms that computer software is developed according to the defined requirement specification. The document evidence is a part of the validation process, which also includes software testing. According to the Guideline on General Principles of Process Validation defined by Center for Drugs and Biologics and Center for Devices and Radiological Health Food and Drug Administration [Food et al. 1985], process validation is defined as "Establishing documented evidence which provides a high degree of assurance that a specific process will consistently produce a product meeting its predetermined specifications and quality characteristics". The Food and Drug Administration (FDA) agency applies this to all processes that fall under its regulation, including computer systems [FDA 2011].

In the present paper, we propose a quality assurance system for computer software solutions in the pharmaceutical industry. We describe both the system architecture and data model in details and the benefits from the implementation of such a system. The idea of putting this system in a cloud is to provide centralized solution with continuously monitoring of the software development progress and its quality. Additionally, the cloud provides increased storage capacity and full time authorized access for quality checking of the developed software functionalities. Our system corresponds to the existing guidelines, protocols, GxP (generalization of quality guidelines in pharmaceutical and food industries)
rules and good manufacturing practice (GMP) methods specified for drug software system design and quality. The goal of the system is to provide a structured data environment and to automate the process of documents generation required for quality document evidence. It is mainly based, but not limited on the quality requirements specified in Good Automated Manufacturing Practice (GAMP) 5 risk based-approach to Compliant GxP Computerized Systems [GAMP 5]. The main idea is to ensure that pharmaceutical software is developed according to already accepted standards for developing software in pharmaceutical industry, in this case, GAMP 5.

The paper is organized as follows: in the next Section we give an overview of the existing computer software quality validation methods in the pharmaceutical industry. In Section 3, we describe the system architecture in details. Section 4 provides the data model of our system. The benefits and drawbacks of the systems are provided in Section 5 and conclusive remarks are given in the last Section.

2. RELATED WORK

GAMP 5 [GAMP 5] is a cost effective framework of good practice which ensures that computerized systems are ready to be used and compliant with applicable regulations. Its aim is to ensure patient safety, product quality, and data integrity. This Guide can be used by regulated companies, suppliers, and regulators for software, hardware, equipment, system integration services, and IT support services.

In addition to GAMP 5, there are several more validation guiding specifications that are commonly used in validating automation systems in pharmaceutical industry.

The "Guidance for Industry: General Principles of Software Validation" [US Food and Drug Administration and others 2002] describes the general validation principles that FDA proposes for the validation of software used to design, develop, or manufacture medical devices. This guideline covers the integration of software life cycle management and the risk management activities.

The "CFR(Code of Federal Regulations) Title 21 - part 11" [US Food and Drug Administration and others 2012] provides rules for the food and drug administration. It emphasizes the validation of systems in order to ensure accuracy, reliability, consistent intended performance, and the ability to discern invalid or altered electronic records.

The "CFR(Code of Federal Regulations) Title 21 - part 820" [Food and Drug Administration and others 1996] sets the current good manufacturing practice (CGMP). The requirements in this part are oriented to the the design, manufacture, packaging, labeling, storage, installation, and servicing of all finished devices intended for human use. These requirements ensure that finished devices will be safe and effective.

The "PDA Technical Report 18, (TR 18) Validation of Computer-Related Systems" [PDA Committee on Validation of Computer-Related Systems 1995] elaborates the steps to be taken in selecting, installing, and validating computer systems used in pharmaceutical GMP (Good Manufacturing Practice) functions. It provides information about practical documentation that can be used to validate the proper performance of the computer systems.

According to the "1012-2004 - IEEE Standard for Software Verification and Validation" [148 2005] the term software also includes firmware, microcode, and documentation. This standard specifies the software verification and validation life cycle process requirements. It includes software-based systems, computer software, hardware, and interfaces and it can be applied for software being developed, reused or maintained.

In [Wingate 2016], the authors provide practical advices and guidance on how to achieve quality when developing pharmaceutical software. Various processes utilized to automate QA (quality assurance) within CRM systems within the pharmaceutical and biotech industry and to define current QA requirements are presented in [Simmons et al. 2014].

Compared to the researches that have been made so far, no system for automatic quality assurance based on GAMP 5 has been proposed yet in the pharmaceutical industry. The system we propose should provide cloud based solution for pharmaceutical software management and automatic document generation based on GAMP 5.

3. SYSTEM ARCHITECTURE

Fig. 1 shows the architecture of our pharmaceutical software quality assurance system. Pharmacists from different pharmaceutical laboratories access the quality assurance system solution hosted in the Cloud by using a web browser.
Each user in the pharmaceutical laboratory has login credentials which allow him to log in to the system and to manage data for the computer software being tested. A user from the pharmaceutical laboratory 1 has permissions only to manage data for software solutions developed in his laboratory. Also, a pharmacist with provided credentials has a possibility to access the cloud solution outside the laboratory by using any electronic device that is connected to Internet and supports web browsing. The primary method for the authentication will be weblogin. Users are authorized to access only the data for the projects they participate in.

After the successful login, the user has a possibility to chose a software project from the list of the project being developed in his laboratory. According to example forms proposed by GAMP 5 [GAMP 5], the system must provide generation of the following documents:

—Risk assessment form;
—Source code review form;
—Forms to assist with testing;
—Forms to assist with the process of managing a change;
—Forms to assist with document management;
—Format for a traceability matrix;
—Forms to support validation reporting;
—Forms to support backup and restore;
—Forms to support performance monitoring.
The main idea of our quality assurance system is the automatic generation of the required document forms. The system should provide a preview of the missing documents by checking the inserted data for a selected software solution.

The manual document filling is replaced by importing the data from the database. User is only responsible for inserting the data for the software solution by using the web interface. For example, if a user inserted the names of the software functions once, they will be used for the generation of all documents that contains records for the software functions names. When a change for an inserted function is required or a new test should be added, the user only selects the function from the lists of provided functions and change the required data. There is also an option for autofill of certain fields provided in the web interface such as: today’s date, user name, project name, function auto increase number, test status, etc.

The details about the required data for successful generation of all quality documents (listed above) is given in the data model, described in the next Section. The benefits of using cloud solution for our quality assurance system is described in the Section 5.

4. SYSTEM DATA MODEL

The data model of our pharmaceutical software quality assurance system is shown in Fig. 2. Each user has an opportunity to access multiple projects that is authorized for. Projects must have at least one user. A project is composed of many functions which are divided into subfunctions. The entity "Document" is intended for storing the each generated document specified in GAMP 5, as listed in Section 3.

The risk assessment form can be generated by using the "RiskAssessment" entity where each row represents one row from the risk assessment document. Each row of the "RiskAssessment" entity is aimed for a specific system subfunction. A given subfunction can have multiple risk assessment row records.

The entity "SourceCodeReview" is designed for the creation of the Source Code Review Report document. Similarly, each row from this entity represents a row in the Source Code Review Report and it is dedicated to a specific system subfunction.

Test Results Sheet is created for a system subfunction by using the entity "Test" and for each test a new document is generated. A Test Incident Sheet must be connected to the specific test. There might be more test incidents for a given test.

Change Request is a document for proposing project changes. Each change request can have multiple change notes as shown in our data model. The "ChangeRequest" and "ChangeNote" entities are used for this purpose.

The system backup is documented in the Data Backup document. In our data model this entity is named "DataBackup". A new document is created for each performed system backup. Data Restoration Form is aimed to be generated from data stored in one row in the "DataRestoration" entity table.

Monitoring Plan Form is consisted of records for different Monitored Parameters. These data are stored in the "MonitoredParameter" entity table. Each row of this table represents a data row in the document.

The Test Progress Sheet, Change Request Index, all forms to assist with document management (Document Circulation Register, Document History, Master Document Index, Review Report, Review Summary), Traceability Matrix Form, forms to support validation reporting are summary documents and they are generated with querying the data model by joining the required entity tables.

5. PROS AND CONS OF THE SYSTEM IMPLEMENTATION

The proposed quality assurance system has both advantages and disadvantages. It can be beneficial in terms of:

—Centralized solution accessible from everywhere;
—Automatic generation of quality assurance documents;
—Records for functions and subfunctions are used in multiple document creation;
—Summary reports are generated from the existing data, no need to insert additional data;
Fig. 2  Pharmaceutical Software Quality Assurance System Data Model
—Name unification of system components (functions, subfunctions);
—Easy accessible interface;
—Allowing parallel data insertion;
—Reduced number of errors in documents;
—Cloud provides elasticity, scalability and multi-tenancy;
—Only pay for the options you want in the Cloud;
—Cloud provides easy backup of the data at regular intervals, minimizing the data loss;
—No need of an investment in hardware and infrastructure.

As a main possible disadvantage is the Internet connection factor. We cannot rely on an Internet connection to access the data if Internet goes down on user side or on the cloud provider’s side. Also, the users do not have physical control over the servers.

6. CONCLUSION

In this paper we propose an architecture and data model for pharmaceutical software quality assurance system hosted on the Cloud. We made a brief review of the existing guidelines and standards for developing quality software in pharmaceutical industry. The proposed architecture shows the easy system accessibility from any electronic device connected to Internet. We also provide data model showing the organization and structure of the data used in the system. There are many advantages for developing such a system and we describe each of them. In the future, we plan to implement and use this system in practice and to found any inconsistencies that can be improved in the next system versions.

Acknowledgement

This work is supported by the project Advanced Scientific Computing Infrastructure and Implementations, financed by the Faculty of computer science and engineering, UKIM.

REFERENCES


Assessing the Impact of Untraceable Bugs on the Quality of Software Defect Prediction Datasets

GORAN MAUŠA and TIHANA GALINAC GRBAC, University of Rijeka, Faculty of Engineering

The results of empirical case studies in Software Defect Prediction are dependent on data obtained by mining and linking separate software repositories. These data often suffer from low quality. In order to overcome this problem, we have already investigated all the issues that influence the data collection process, proposed a systematic data collection procedure and evaluated it. The proposed collection procedure is implemented in the Bug-Code Analyzer tool and used on several projects from the Eclipse open source community. In this paper, we perform additional analysis of the collected data quality. We investigate the impact of untraceable bugs on non-fault-prone category of files, which is, to the best of our knowledge, an issue that has never been addressed. Our results reveal this issue should not be an underestimated one and should be reported along with bugs’ linking rate as a measure of dataset quality.

Categories and Subject Descriptors: D.2.5 [SOFTWARE ENGINEERING]: Testing and Debugging—Tracing; D.2.9 [SOFTWARE ENGINEERING]: Management—Software quality assurance (SQA); H.3.3 [INFORMATION STORAGE AND RETRIEVAL] Information Search and Retrieval

Additional Key Words and Phrases: Data quality, untraceable bugs, fault-proneness

1. INTRODUCTION

Software Defect Prediction (SDP) is a widely investigated area in the software engineering research community. Its goal is to find effective prediction models that are able to predict risky software parts, in terms of fault proneness, early enough in the software development process and accordingly enable better focusing of verification efforts. The analyses performed in the environment of large scale industrial software with high focus on reliability show that the faults are distributed within the system according to the Pareto principle [Fenton and Ohlsson 2000; Galinac Grbac et al. 2013]. Focusing verification efforts on software modules affected by faults could bring significant costs savings. Hence, SDP is becoming an increasingly interesting approach, even more so with the rise of software complexity.

Empirical case studies are the most important research method in software engineering because they analyse phenomena in their natural surrounding [Runeson and Höst 2009]. The collection of data is the most important step in an empirical case study. Data collection needs to be planned according to the research goals and it has to be done according to a verifiable, repeatable and precise procedure [Basili and Weiss 1984]. The collection of data for SDP requires linking of software development repositories.
that do not share a formal link [D'Ambros et al. 2012]. This is not an easy task, so the majority of researchers tend to use the publicly available datasets. In such cases, researchers rely on the integrity of data collection procedure that yielded the datasets and focus mainly on prediction algorithms. Many machine learning algorithms are demanding and hence they divert the attention of researchers from the data upon which their research and results are based [Shepperd et al. 2013]. However, the datasets and their collection procedures often suffer from various quality issues [Rodriguez et al. 2012; Hall et al. 2012].

Our past research was focused on the development of systematic data collection procedure for the SDP research. The following actions had been carried out:

— We analyzed all the data collection parameters that were addressed in contemporary related work, investigated whether there are unaddressed issues in practice and evaluated their impact on the final dataset [Mauša et al. 2015a];

— We had evaluated the weaknesses of existing techniques for linking the issue tracking repository with the source code management repository and developed a linking technique that is based on regular expressions to overcome others' limitations [Mauša et al. 2014];

— We determined all the parameters that define the systematic data collection procedure and performed an extensive comparative study that confirmed its importance for the research community [Mauša et al. 2015b];

— We developed the Bug-Code Analyzer (BuCo) tool for automated execution of data collection process that implements our systematic data collection procedure [Mauša et al. 2014].

So far, data quality was observed mainly in terms of bias that undefined or incorrectly defined data collection parameters could impose to the final dataset. Certain data characteristics affect the quality characteristics. For example, empty commit messages may lead to duplicated bug reports [Bachmann and Bernstein 2010]. That is why software engineers and project managers should care about the quality of the development process. The data collection process cannot influence these issues but it may analyse to what extent do they influence the quality of the final datasets. For example, empty commit messages may also be the reason why some bug reports remain unlinked. Missing links between bugs and commit messages lead to untraceable bugs. This problem is common in the open source community [Bachmann et al. 2010].

In this paper, we address the issue of data quality with respect to the structure of the final datasets and the problem of untraceable bugs. This paper defines untraceable bugs as the defects that caused a loss of functionality, that are now fixed, and for which we cannot find the bug-fixing commit, i.e. their location in the source code. Our research questions tend to quantify the impact of untraceable bugs on SDP datasets. Giving answer to this question may improve the assessment of SDP datasets' quality and it is the contribution of this paper. Hence, we propose several metrics to estimate the impact of untraceable bugs on the fault-free category of software modules and perform a case study on 35 datasets that represent subsequent releases of 3 major Eclipse projects. The results revealed that the untraceable bugs may impact a significant amount of software modules that are otherwise unlinked to bugs. This confirms our doubts that the traditional approach, which pronounces the files that are unlinked to bugs as fault-free, may be lead to incorrect data.

2. BACKGROUND

Software modules are pronounced as Fault-Prone (FP) if the number of bugs is above a certain threshold. Typically, this threshold is set to zero. The software units that remained unlinked to bugs are typically declared as Non-Fault-Prone (NFP). However, this may not be entirely correct if there exists
a certain amount of untraceable bugs. This is especially the case in projects of lower maturity level. No matter which linking technique is used in the process of data collection from open source projects, all the bugs from the issue tracking repository are never linked. Therefore, it is important to report the linking rate, i.e. the proportion of successfully linked bugs, to reveal the quality of the dataset. Linking rate is usually improved with the maturity of the project, but it never reaches 100%. Instead, we can expect to link between 20% and 40% bugs in the earlier releases and up to 80% - 90% of bugs in the "more mature", later releases [Mauša et al. 2015a; Mizuno et al. 2007; Gyimothy et al. 2005; Denaro and Pezze 2002]. Moreover, an Apache developer identified that a certain amount of bugs might even be left out from the issue tracking system [Bachmann et al. 2010].

Both of these data issues reveal that there is often a number of untraceable bugs present in the open source projects, i.e. a serious data quality issue. So far, the problem of untraceable bugs was considered only in studies that were developing linking techniques. For example, the ReLink tool was designed with the goal to find the missing links between bugs and commits [Wu et al. 2011]. However, our simpler linking technique based on regular expressions performed equally good or better than the ReLink tool and it did not yield false links [Mauša et al. 2014; Mauša et al. 2015b]. The bugs that remained unlinked could actually be present in the software units that remained unliked and, thus, disrupt the correctness of the dataset. Thus, it may be incorrect to declare all the software units not linked to bugs as NFP. To the best of our knowledge, this issue remained unattended so far. Nonetheless, there are indications that lead us to believe that the correctness of SDP datasets that is deteriorated by untraceable bugs can be improved. Khoshgoftaar et al. collected the data for SDP from a very large legacy telecommunications system and found that more than 99% of the modules that were unchanged from the prior release had no faults [Khoshgoftaar et al. 2002; Khoshgoftaar and Seliya 2004].

3. CASE STUDY METHODOLOGY

We use the GQM (Goal-Question-Metrics) approach to state the precise goals of our case study. Our goal is to obtain high quality of data for SDP research. To achieve this goal, we have already analysed open software development repositories, investigated existing data collection approaches, revealed issues that could introduce bias if left open to interpretation and defined a systematic data collection procedure. The data collection procedure was proven to be of high quality [Mauša et al. 2015b]. However, a certain amount of untraceable bugs is always present. If such a bug actually belongs to a software module that is otherwise unlinked to the remaining bugs, than it would be incorrect to pronounce such a software module as fault-free.

3.1 Research questions

Research questions that drive this paper are related to the issue of untraceable bugs and their impact on the quality of data for SDP research. To accomplish the aforementioned goal, we need to answer the following research questions (RQ):

(1) How many fixed bugs remain unlinked to commits?
(2) How many software modules might be affected by the untraceable bugs?
(3) How important it is to distinguish the unchanged software modules from other modules that remain unlinked to bugs?

The bug-commit linking is done using the Regex Search linking technique, implemented in the BuCo tool. This technique proved to be better than other existing techniques, like the ReLink tool [Mauša et al. 2014], and the collection procedure within the BuCo tool has shown to be more precise than other existing procedures, like the popular SZZ approach [Mauša et al. 2015a]. Using this technique
we minimize the amount of bugs that are untraceable. Furthermore, BuCo tool uses the file level of granularity and software modules are regarded as files in the remainder of the paper.

3.2 Metrics

We propose several metrics to answer our research questions. The metric for the RQ1 is the linking rate (LR), i.e. the ratio between the number of successfully linked bugs and the total number of relevant bugs from the issue tracking repository. The metrics for the RQ2 and RQ3 are defined using the following categories of software modules:

— **FP** – files linked with at least one bug
— **Unlinked** – files not linked to bugs
— **Changed** – files for which at least one of 50 product metrics is changed between two consecutive releases $n+1$ and $n$
— **Removed** – files that are present in release $n$, and do not exist in release $n+1$
— **FP Candidates** – Unlinked files that are Changed or Removed
— **NFP** – Unlinked files that are not Changed nor Removed

The relationship between these categories of files are presented in Figure 1. No previously published related research investigated the category of Non-Fault-Prone (NFP) files. It is reasonable to assume they categorized the Unlinked category as NFP. However, the linking rate that is below 100% reveals that there is a certain amount of untraceable bugs and we know that a file might be changed due to an enhancement requirement and/or a bug. Hence, we conclude that some of the Unlinked files that are Changed or Removed might be linked to these untraceable bugs, and categorize them as **FP Candidates**. The Unlinked files that are not Changed are the ones for which we are more certain that they are indeed Non-Fault-Prone. Thus, we categorize only these files as NFP. This approach is motivated by Khoshgoftaar et al. [Khoshgoftaar et al. 2002; Khoshgoftaar and Seliya 2004] as explained in section 2.

Using the previously defined categories of files, we define the following metrics:

\[
C_U = \frac{FP\_Candidates}{Unlinked} \tag{1}
\]

The FP\_Candidates in Unlinked ($C_U$) metric reveals the structure of Unlinked files, i.e. what percentage of Unlinked files is potentially affected by untraceable bugs. This metric enables us give an estimation for our RQ2.

\[
FpB = \frac{FP}{Linked\_bugs} \tag{2}
\]
Files per Bug (FpB) metric reveals the average number of different files that are affected by one bug. It should be noted that the bug-file cardinality is many-to-many, meaning that one bug may be linked to more than one file and one file may be linked to more than one bug. Hence, the untraceable bugs could be linked to files that are already FP, but we want to know how many of the Unlinked files they might affect. Therefore, we divide the total number of FP files (neglecting the number of established links per file) with the total number of linked bugs.

\[ Ub_U = \frac{FpB \times Untraceable\_bugs}{Unlinked} \] (3)

The Untraceable bugs in Unlinked (Ub,U) metric estimates the proportion of Unlinked files that are likely to be linked to untraceable bugs, assuming that all the bugs behave according to the FpB metric. This metric enables us give another estimation for our RQ2. It estimates how wrong would it be to pronounce all the Unlinked files as NFP. The greater is the value of metric Ub,U, the more wrong is that traditional approach. We must point out that there are also bugs that are not even entered into the BT repository. However, the influence of this category of untraceable bugs cannot be estimated, but it could only increase the value of Ub,U.

\[ Ub,C = \frac{FpB \times Untraceable\_bugs}{FP\_Candidates} \] (4)

The Untraceable bugs in FP_Candidates (Ub,C) metric estimates the percentage of FP_Candidates that are likely to be linked to untraceable bugs (Ub,U/C,U), assuming that all the bugs behave according to the FpB measure. This metric enables us give an estimation for our RQ3. It estimates how wrong it would be to pronounce all the FP_Candidates as NFP. The closer is the value of this metric to 1, the more precisely would it be not to pronounce the FP_Candidates as NFP. In other words, the Ub,C metric calculates the percentage of files that are likely to be FP among the FP_Candidates (Ub,U/C,U).

3.3 Data

The source of data are three major and long lasting open source projects from the Eclipse community: JDT, PDE and BIRT. The bugs that satisfy following criteria are collected from the Bugzilla repository: status - closed, resolution - fixed, severity - minor or above. The whole source code management repositories are collected from the GIT system. Bugs are linked to commits using the BuCo Regex linking technique and afterwards the commits are to files that were changed. The cardinality of the link between bugs and commits is many-to-many, and the duplicated links between bugs and files are counted only once. The file level of granularity is used, test and example files are excluded from final datasets, and the main public class is analyzed in each file. A list of 50 software product metrics is calculated for each file using the LOC Metrics\(^1\) and JHawk\(^2\) tools.

4. RESULTS

Table I shows the total number of releases and files we collected; FP, NFP, Changed and Removed files we identified; the total number of relevant bugs from the issue tracking repository; the linking rate obtained by BuCo Regex linking technique; and the total number of commits in the source code management repository. The results of our linking technique are analysed for each project release and presented in Table II. The LR exhibits a rising trend in each following release and reaches stable and high values (80% - 90%) in the "middle" releases. A slight drop in LR is possible in the latest releases.

---

\(^1\)http://www.locmetrics.com/
\(^2\)http://www.virtualmachinery.com/
<table>
<thead>
<tr>
<th></th>
<th>JDT</th>
<th>PDE</th>
<th>BIRT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Releases analyzed</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>Files analyzed</td>
<td>52,033</td>
<td>23,088</td>
<td>31,110</td>
</tr>
<tr>
<td>Files</td>
<td>FP</td>
<td>4,891</td>
<td>5,307</td>
</tr>
<tr>
<td></td>
<td>NFP</td>
<td>27,891</td>
<td>12,059</td>
</tr>
<tr>
<td>Changed</td>
<td>13,443</td>
<td>4,208</td>
<td>10,803</td>
</tr>
<tr>
<td>Removed</td>
<td>917</td>
<td>1,930</td>
<td>67</td>
</tr>
<tr>
<td>Bugs</td>
<td>collected</td>
<td>18,404</td>
<td>6,698</td>
</tr>
<tr>
<td>Bugs</td>
<td>linked</td>
<td>13,193</td>
<td>4,189</td>
</tr>
<tr>
<td>Commits</td>
<td>151,408</td>
<td>23,427</td>
<td>75,216</td>
</tr>
</tbody>
</table>

Table I. Raw Data Analysis

However, observing the absolute value of bugs in those releases, we notice the difference is less severe. As these releases are still under development, new bugs are being fixed and new commits still arrive so this rates are expected to change. These results show that a considerable amount of bugs is untraceable and indicate that their influence may not be insignificant.

The distributions of four file categories are computed for each release of every project and presented in stacked column representations in Figures 2, 3 and 4. We confirm that the problem of class imbalance between FP and Unlinked files (Changed, Removed and NFP) is present in all the releases. The percentage of FP files is usually below 20%, on rare occasions it rises up to 40% and in worst case scenarios it drops even below 5%. The trend of FP files is dropping as the project becomes more mature in the later releases. The NFP files are rare in the earlier releases of the projects showing that the projects are evidently rather unstable then. Their percentage is rising with almost every subsequent release and it rises to rates comparable to the FP category in the "middle" releases. The Removed files are a rather insignificant category of files. The Changed files are present in every release and they exhibit a more stable rate than the other categories.

Tables III, IV and V present the evaluation metrics which we proposed in section 3.2. Metric C,U reveals the relative amount of files that are not linked to bugs, but have been changed in the following release. Because of the untraceable bugs, we cannot be certain about their fault proneness. We
Assessing the Impact of Untraceable Bugs on the Quality of Software Defect Prediction Datasets

Fig. 2. Distribution of files in JDT releases

Table III. Evaluation Metrics for JDT

Fig. 3. Distribution of files in PDE releases

Table IV. Evaluation Metrics for PDE

Fig. 4. Distribution of files in BIRT releases

Table V. Evaluation Metrics for BIRT
notice a significant amount of such data. Metric FpB reveals the average number of distinct files that are changed per bug. The metric is based upon the number of bugs that were successfully linked to commits. Considering that multiple bugs may affect the same files, it is not unusual that one bug on average affects less than 1 distinct file. Later releases have less bugs in total, there is less chance that they affect the same files and there is a slight increase in the value of FpB. The FpB metric is used to estimate the amount of files prone to bugs that were untraceable from the bug tracking repository, expressed in the metric Ub_U. The Ub_U metric varies between releases, from very significant in earlier releases to rather insignificant in the later releases. The Ub_C metric reveals how important would it be to distinguish Changed and Removed files from the NFP files. With its values close to 0%, we expect little bias in the category of NFP files. However, with its greater value, the bias is expected to rise and the necessity to make such a distinction is becoming greater. In several cases, its value exceeds 100%. This only shows that the impact of untraceable bugs is assessed to be even greater than affecting just the FP_Candidates. In the case of JDT 3.8 its value is extremely high because this release contains almost none FP_Candidates. This metric was developed on our own so we cannot define a significance threshold. Nevertheless, we notice this value to be more emphasized in earlier releases that we described as immature and in later releases that are still under development.

4.1 Discussion

Linking rate (LR) enables us to answer our RQ1. We noticed that the LR is very low in the earliest releases of analyzed projects (below 50%). After a couple of releases, the LR can be expected to be between 80% and 90%. We also observe that the distribution of FP files exhibits a decreasing trend as the product evolves through releases. That is why we believe that developer are maturing along with the project and, with time, they become less prone to faults and more consistent in reporting the Bug IDs in the commit titles when fixing bugs. The latest releases are still under development and exhibit extreme levels of data imbalance, with below 1% of FP files. Therefore, these datasets might not be the proper choice for training the predictive models in SDP.

Our results enable us to give an estimate for the RQ2. The Unlinked files contain a rather significant ratio of files that are FP candidates, spanning from 10% up to 50% for the JDT and BIRT projects and above 50% in several releases of the PDE project. Among the FP candidates, we expect to have a more significant amount of files that are FP due to the untraceable bugs in earlier releases because of low LR. According to the Ub_C metric, we may expect that the majority of FP_Candidates actually belong to the FP category in the earliest releases. According to the Ub_U metric, the untraceable bugs affect a rather insignificant percentage of all the Unlinked files after a couple of releases.

The metrics we proposed in this paper enable us to answer our RQ3. The difference between the Ub_U and Ub_C values confirm the importance of classifying the Unlinked files into Changed, Removed and NFP. In the case of high Ub_C values (above 80%) it may be prudent to categorize FP_Candidates as FP and in the case where Ub_C is between 20% and 80% it may be prudent to be cautious and not to use the FP_Candidates at all. In the case of high difference between the Ub_U and Ub_C metrics, we may expect to have enough of NFP files in the whole dataset even if we discard the FP_Candidates. This is confirmed in the distribution of NFP files which displays an increasing trend which becomes dominant and rather stable in the "middle" releases.

The process of data collection and analysis is fully repeatable and verifiable but there are some threats to validity of our exploratory case study. The construction validity is threatened because the data do not come from industry and the external validity is threatened because only one projects come from only one community. However, the chosen projects are large and long lasting ones and provide a good approximation of the projects from the industrial setting and they are widely analyzed in related
research. Internal validity is threatened by assumptions that all the bugs affect the same quantity of different files and that Unchanged files are surely NFP.

5. CONCLUSION

The importance of having accurate data is the initial and essential step in any research. This paper is yet another step in achieving that goal in the software engineering area of SDP. We noticed that untraceable bugs are inevitable in data collection from open source projects and that this issue remained unattended by the researchers so far. This exploratory case study revealed that it may be possible to evaluate the impact of untraceable bugs on the files that are unlinked to bugs. The results show that the earliest and the latest releases might not be a good source of data for building predictive models. The earliest releases are more prone to faults (containing higher number of reported bugs), radical changes (containing almost no unchanged files) and suffer from low quality of data (lower linking rate). On the other hand, the latest releases suffer from no previously mentioned issues, but are evidently still under development and the data are not stable.

The future work plans to investigate the impact of the explored issues and the proposed solutions to the problem of untraceable bugs on the performance of predictive models. Moreover, we plan to expand this study to other communities using our BuCo Analyzer tool.

REFERENCES


XML Schema Quality Index in the Multimedia Content Publishing Domain

MAJA PUŠNIK, MARJAN HERIČKO AND BOŠTJAN ŠUMAK, University of Maribor
GORDANA RAKIĆ, University of Novi Sad

The structure and content of XML schemas impacts significantly the quality of data respectively documents, defined by XML schemas. Attempts to evaluate the quality of XML schemas have been made, dividing them into six quality aspects: structure, transparency and documentation, optimality, minimalism, reuse and integrability. XML schema quality index was used to combine all the quality aspects and provide a general evaluation of XML schema quality in a specific domain, comparable with the quality of XML schemas from other domains. A quality estimation of an XML schema based on the quality index leads to a higher efficiency of its usage, simplification, more efficient maintenance and higher quality of data and processes. This paper addresses challenges in measuring the level of XML schema quality within the publishing domain, which deals with challenges of multimedia content presentation and transformation. Results of several XML schema evaluations from the publishing domain are presented, compared to general XML schema quality results of an experiment, that included 200 schemas from 20 different domains. The conducted experiment is explained and the state of data quality in the publishing domain is presented, providing guidelines for necessary improvements in a domain, dealing with multimedia content.

Categories and Subject Descriptors: H.0. [Information Systems]: General; D.2.8 [Software Engineering]: Metrics — Complexity measures; Product metrics; D.2.9. [Software Engineering]: Management — Software quality assurance (SQA)

General Terms: Software quality assurance

Additional Key Words and Phrases: software metrics, quality metrics, XML Schema, multimedia, publishing

INTRODUCTION

This paper is focused on the publishing domain, documents in the publishing domain and the quality level of documents’ structure, defined by XML schemas. XML schemas are a widely used technology for structure definition of XML documents and can be on very different quality levels, measured by predefined XML schema metrics, in our case with a quality index, explained in this paper. The activities in this paper include (1) collecting available XML schemas from the publishing field, (2) measuring the characteristics of XML schemas based on quality metrics (the quality index) and (3) critically evaluating their quality and setbacks as well as (4) comparing the results with XML schemas from other (previously evaluated) domains.

The publishing process is performed in both printed and electronic form, however there is an increasing number of eBooks (Shaffer, 2012). In recent times eBooks have started to become more interactive to the extent of providing rich multimedia content; hyperlinks to resources on the web; allowing the reader to highlight text, add notes and bookmark pages, videos, interactive games and other. Newer interactive features include different multimedia content such as embedded audio, video, slide shows and image galleries (Fenwick Jr, J.B., Phillips, R., Kurtz, B.L., Weidner, A., Meznar, 2013) and there are many publishing aspects that need to be addressed based on new

Authors’ addresses: Maja Pušnik, Marjan Heričko, Boštjan Šumak, Faculty of Electrical Engineering and Computer Science, University of Maribor, Smetanova 17, 2000 Maribor, Slovenia, email: maja.pusnik@uni-mb.si, marjan.hericko@um.si, bostjan.sumak@um.si
Gordana Rakic, University of Novi Sad, Faculty of Sciences, Department of Mathematics and Informatics, Trg Dositeja Obradovica 4, 21000 Novi Sad, Serbia; email:gordana.rakic@DMI.UNS.ac.rs

Copyright © by the paper’s authors. Copying permitted only for private and academic purposes.

demands. In this paper, we are evaluating the quality of XML schema support, included in different multimedia content types for the publishing purposes, and comparing the results with other domains.

The paper is organized in the following manner: related work and research questions are included in the first introduction section. The quality index with quality aspects is presented in the second section. Section three includes results of applying quality index on XML schemas from the publishing field and a comparison with other domains is discussed. Limitations and threats to validity are explained in section 4 and conclusion and future work is presented in section 5, followed by listed references.

1.1 Related work

Publishing domain, its multimedia content and XML technologies have little history compared to other domains, however several papers were found, addressing its connection and influence on quality issues. Only for the last 5 years, more than 100 search results were identified. 16 papers were extracted, focusing on topics regarding multimedia content including XML documents and the publishing of different document types. Based on existing research, several publishing fields with extensive multimedia content were identified: geographical documentation, medical documentation and general software documentation.

XML technologies are greatly involved in defining and presenting geographical documentation (Laurini, 2014) and land administration (Lemmen, van Oosterom, & Bennett, 2015) as well as other industries; (Chen, 2016) discusses XML schema benefits in the process of integration, filtering and formatting of graphical information across the globe (Patel, Nordin, & Al-Haiqi, 2014). The presentation of data with XML support is also documented in medical literature (Wagner et al., 2016) and is helpful at evaluation of student’s literature understanding (Aparicio, De Buenaga, Rubio, & Hernando, 2012) as well as in correlating fields such as ecology (Chamanara & König-Ries, 2014). Context-awareness and behavior adaptation of different multimedia content based on XML technologies is addressed in (Chihani, Bertin, & Crespi, 2014) and knowledge domain is emphasized in (Ait-Ameur & Méry, 2015). Publishing of learning material is greatly supported by XML; where authors define visual perception improvement through XML annotations (Hiippala, 2012), mobile learning products have even greater need for XML technologies (Sarrab, Elbasir, & Alnaeli, 2016), and proper (educational) literature for dangerous life situations such as earthquakes (Gaeta et al., 2014) also relies on XML based structures. Educational aspect of appropriately designed context can provide support in the system (Chik-Parnas, Dragomiroiu, & Parnas, 2010) and big data within large amounts of documents is addressed as well in (Priyadarshini, Tamilselvan, Khuthbudin, Saravanan, & Satish, 2015). The research of listed papers exposes several aspects of benefits and potential problems in the publishing field in general and creates a basis for the following research questions, which will be addressed in this paper:

(1) Does the publishing domain use XML documents and what standard XML schemas are being used?

(2) What is the quality level of XML schemas in the publishing domain?

(3) How are they compared to XML schemas in other domains such as computer science and other?

(4) How can the level of quality be improved?

Literature review and an experiment based on existing XML schema metrics were methods, used to answer listed research questions in addition to critical comparison to existing results of other domains. A set of metrics for assessing the quality of XML schemas is presented as well, united in a general quality evaluation - XML Schema Quality Index, used to bring all results to a common ground, making them comparable. The quality index is presented in the next section.
QUALITY INDEX

The quality of XML schemas is a general term and includes several aspects (structure, transparency, optimality, minimalism, reuse and integrability), addressed in (Pušnik, Boštjan, Hericko, & Budimac, 2013). The aspects were defined based on a preliminary research of general software metrics, presenting most used latent variables which cannot be always measured objectively. Therefore, the 6 aspects include measurable parameters of XML schema building blocks and their relations, encapsulating the final quality index: (1) structural quality aspect, (2) transparency and documentation of the XML schema quality aspect, (3) XML schema optimality quality aspect, (4) XML schema minimalism quality aspect, (5) XML schema reuse quality aspect and (6) XML schema integrability quality aspect. The exact calculation are presented in (Pusnik, Hericko, Budimac, & Sumak, 2014).

The goal of this paper is to measure the quality of XML schemas, which are an important part of several domains, where data is exchanged in form of XML documents. The quality of XML schemas indirectly impacts the information system quality and further on different companies’ business processes (publishing companies as only one example). Companies however often use XML schemas, who meet the minimum criteria of syntactical correctness and content description (Pusnik, Hericko, Budimac, & Sumak, 2014). Quality evaluation was conducted on 200 schemas where results indicated that 30% of identified XML schemas (within 20 different domains) have a very low quality index and are built inappropriately (regarding the structure and documentation), influencing the quality of the information solution. The purpose of this paper was to evaluate and compare the publishing field to the general situation, since the publishing field does include structured data, being transferred on a daily basis. The domains, to which we compare the publishing domain, are presented in the next section.

1.2 XML schemas in the publishing domain

Analysis of 200 schemas from 20 domains was conducted in (Pušnik, 2014). The criteria which domain was included into the analysis was the number of used XML schemas: only top 20 domains with most XML document transactions and XML schema definitions were included. Domains, which provided a general state of used XML schemas, are presented in table (Table I)

<table>
<thead>
<tr>
<th>D1 - Mathematics and Physics</th>
<th>D11 –Decision Science</th>
</tr>
</thead>
<tbody>
<tr>
<td>D2 - Materials Science</td>
<td>D12 –Medicine</td>
</tr>
<tr>
<td>D3 - Telecommunications</td>
<td>D13 - Economics and finance</td>
</tr>
<tr>
<td>D4 - Manufacturing</td>
<td>D14 - Law</td>
</tr>
<tr>
<td>D5 - Energy and Electronics</td>
<td>D15 - Social science</td>
</tr>
<tr>
<td>D6 - Engineering</td>
<td>D16 –Health and sport</td>
</tr>
<tr>
<td>D7 - IT architecture and design</td>
<td>D17 –Construction</td>
</tr>
<tr>
<td>D8 - Traffic</td>
<td>D18 - Librarianship (Library)</td>
</tr>
<tr>
<td>D9 - Communications</td>
<td>D19 - Landscape and geography</td>
</tr>
<tr>
<td>D10 –Computer Science</td>
<td>D20 –Media, journalism, newspapers</td>
</tr>
</tbody>
</table>

The publishing domain was not specifically investigated within the primary set of domains, however was included in domain D20 - Media, journalism, newspapers. However, due to the expanding use of XML and related technologies in the publishing field, we conducted a similar research of XML schema quality in this specific field and compared it to the average values, received when analysing most often used XML schema. The 10 identified XML schemas, that were valid,
publically available and supported by all included and imported XML schemas were evaluated based on the six quality aspects, presented in the next section.

1.3 The six quality aspects
The aspects of XML schemas are evaluated and presented in (Pušnik et al., 2014) and include equations, combining the listed parameters (Table II). They are presented in more detail in the following sections.

1.3.1 Structural quality aspect (QA1). The structure aspect evaluates the number and relationship among building blocks of XML schemas. It includes several measured parameters and focuses on the level of complexity. Metrics include relationships between simple and complex data types, relationship between annotations and the number of elements, average number of restrictions on the declaration of a simple type, percentage of the derived type declarations of total number of declaration complex types and diversification of the elements or 'fanning', which is influenced by the complexity of XML schemas, suggesting inconsistencies that unnecessarily increase the complexity.

1.3.2 Transparency and documentation of the XML Schema (QA2). The importance of well documented and easy-to-read as well as understandable XML schema is derived from the following relationship: number of annotation per number of elements and attributes, illustrating the documentation of XML schemas, supposing that more information about the building blocks increases the quality.

1.3.3 XML schema optimality quality aspect (QA3). Metric evaluates whether the in-lining pattern has been used, the least preferable one in XML schema building. In doing so, we focus on the following relationships: the relationship between local and all elements, the relationship between local attributes and all attributes, the relationship between global and complex elements of all complex elements, the relationship between global and simple elements of all simple elements. Ratio between XML schema building blocks should be minimized, indicating minimization of local elements and attributes and maximization of global simple and complex types. The number of global elements however should be as low as possible, due to the problem of several roots (such flexibility is not always appreciated).

1.3.4 XML schema minimalism quality aspect (QA4). Metric of minimalism is defined as the level, when there is no other full set of less building blocks. Number of annotations, elements and attributes should be according to the size of XML schema (LOC respectively).

1.3.5 XML schema reuse quality aspect (QA5). Metric is focused on reuse of the existing software and includes parameters that allow the reuse and are inherently global. References are mostly calculated and number of references to elements (per defined elements) is measured as well as the number of references to attribute (per defined attributes), number of references to groups (per defined groups) and the number of imported or included XML schemas.

1.3.6 XML schema integrability quality aspect (QA6). Metrics measure capability of XML schema components to be integrated, including number of elements and references on elements (per defined elements), number of attributes and references on attributes (per defined attributes), number of groups and references on groups as well as number of imported XML schemas and annotations.

1.3.7 XML schema Quality Index (QI). Equally combines all six metrics and provides the average value. The values have been scaled between the 0 and 1 (Pušnik, 2014) for the results to be comparable.

The measurement and evaluation process based on XML schema parameters and metrics is described in (Pušnik et al., 2013) and summarized in Table II, addressing basic characteristics of XML
xml schemas. Their values are gathered into the six metrics, composing the holistic quality index in the following section.

Table II Names and abbreviations of all used parameters and metrics

<table>
<thead>
<tr>
<th>Simple metrics (parameters)</th>
<th>Composite metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td>File size [KB] (P1)</td>
<td>XML schema type (M1)</td>
</tr>
<tr>
<td>Number of imports (P2)</td>
<td>Ratio of simple to complex types of elements (M2)</td>
</tr>
<tr>
<td>Element related parameters: number of all elements (P3)</td>
<td>Percentage of annotations over total number of elements and attributes (M3)</td>
</tr>
<tr>
<td>number of global elements (P3.1), number of simple elements</td>
<td>Average number of restrictions per number of elements and attributes (M4)</td>
</tr>
<tr>
<td>(P3.2), number of complex elements (P3.4), number of global</td>
<td>Number of all data types (M5)</td>
</tr>
<tr>
<td>complex elements (P3.1.1), number of global simple elements</td>
<td>Percentage of derived data types over all complex types (M6)</td>
</tr>
<tr>
<td>(P3.1.2)</td>
<td>Average use of minimal and maximal occurs per defined elements (M7)</td>
</tr>
<tr>
<td>Attribute related parameters: number of all attributes (P4)</td>
<td>Average number of attributes per complex types (M8)</td>
</tr>
<tr>
<td>number of local attributes (P4.1), number of global attributes</td>
<td>Number of unbounded elements (M9)</td>
</tr>
<tr>
<td>(P4.2)</td>
<td>Element fanning (M10)</td>
</tr>
<tr>
<td>Lines of code (P5)</td>
<td>Quality index (QI)</td>
</tr>
<tr>
<td>Group related parameters: number of element groups (P6.1),</td>
<td></td>
</tr>
<tr>
<td>number of attribute groups (P6.2)</td>
<td></td>
</tr>
<tr>
<td>Reference related parameters: number of element references</td>
<td></td>
</tr>
<tr>
<td>(P7.1), number of references on simple elements (P7.1.1),</td>
<td></td>
</tr>
<tr>
<td>number of references on complex elements (P7.1.2), number of</td>
<td></td>
</tr>
<tr>
<td>references on attributes (P7.2), number of references on</td>
<td></td>
</tr>
<tr>
<td>element groups (P7.3), number of references on attribute groups</td>
<td></td>
</tr>
<tr>
<td>(P7.4)</td>
<td></td>
</tr>
<tr>
<td>Number of annotations (P8)</td>
<td></td>
</tr>
<tr>
<td>Number of restrictions (P9)</td>
<td></td>
</tr>
<tr>
<td>Number of derived (extended) types (P10)</td>
<td></td>
</tr>
</tbody>
</table>

RESULTS

XML schemas, defined within different companies and organizations for the needs of publishing process were analyzed. Based on the analysis of 200 XML schemas from 20 domains, the 10 XML schemas from the publishing domain were compared. The quality aspects of the publishing domain to average results is presented in Fig. 1.
Based on the calculated index quality, a composite of all six quality aspects (equally distributed), the research questions were addressed:

(1) RQ1 – Several XML schemas were found, connected to the publishing field (respectively publishing process) through active research. No standard forms were found, 10 were extracted.

(2) RQ2 – The quality level was measured through metrics, defined in (Pusnik et al., 2014). The average quality of XML schema in the publishing field is 39%.

(3) RQ3 – The quality index of 39% is significantly higher than by the average quality index (of all 20 domains, where XML schemas are most common) which is 29% based on an experiment from 2014.

(4) RQ4 – Comparing to average XML schemas, the publishing field had lower results only at transparency and documentation quality aspect, all other quality aspects were above average.

The results provided better quality index for the publishing domain for most publishing XML schemas compared to average ones (Fig. 1) from the set of 200 XML schemas. The T-test for finding significant difference among the groups was used. The p-value resulted in 0.21 which makes the difference non-significant among parameters. The results comparison is presented in Table III. The metric M1 is left empty, since average value cannot be determined.

**Table III. Comparing publishing domain with average results of all domains**

<table>
<thead>
<tr>
<th>PARAMETERS</th>
<th>Publishing domain - average</th>
<th>All domains - average</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>604,000</td>
<td>133,932</td>
</tr>
<tr>
<td>P2</td>
<td>1,400</td>
<td>0,789</td>
</tr>
<tr>
<td>P3</td>
<td>86,600</td>
<td>77,727</td>
</tr>
<tr>
<td>P3.1</td>
<td>48,000</td>
<td>26,755</td>
</tr>
<tr>
<td>P3.2</td>
<td>38,600</td>
<td>50,973</td>
</tr>
<tr>
<td>P3.3</td>
<td>31,800</td>
<td>27,691</td>
</tr>
<tr>
<td>P3.4</td>
<td>54,800</td>
<td>50,036</td>
</tr>
<tr>
<td>P3.1.1</td>
<td>35,700</td>
<td>19,250</td>
</tr>
<tr>
<td>P3.1.2</td>
<td>12,700</td>
<td>7,514</td>
</tr>
<tr>
<td>P4</td>
<td>26,600</td>
<td>47,655</td>
</tr>
<tr>
<td>P4.1</td>
<td>26,600</td>
<td>47,091</td>
</tr>
<tr>
<td>P4.2</td>
<td>0,000</td>
<td>0,564</td>
</tr>
<tr>
<td>P5</td>
<td>14,969,200</td>
<td>31,886,818</td>
</tr>
<tr>
<td>P6.1</td>
<td>1,000</td>
<td>4,364</td>
</tr>
<tr>
<td>P7.1</td>
<td>65,400</td>
<td>69,118</td>
</tr>
<tr>
<td>P7.1.1</td>
<td>9,000</td>
<td>3,177</td>
</tr>
<tr>
<td>P7.1.2</td>
<td>56,400</td>
<td>65,941</td>
</tr>
<tr>
<td>P7.2</td>
<td>0,000</td>
<td>1,927</td>
</tr>
<tr>
<td>P7.3</td>
<td>0,200</td>
<td>7,114</td>
</tr>
<tr>
<td>P7.4</td>
<td>3,700</td>
<td>11,664</td>
</tr>
<tr>
<td>P8</td>
<td>0,900</td>
<td>0,977</td>
</tr>
<tr>
<td>P9</td>
<td>30,100</td>
<td>75,118</td>
</tr>
<tr>
<td>P10</td>
<td>41,800</td>
<td>35,309</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>METRICS</th>
<th>Publishing domain - average</th>
<th>All domains - average</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M2</td>
<td>5,150</td>
<td>2,266</td>
</tr>
<tr>
<td>M3</td>
<td>0,749</td>
<td>0,513</td>
</tr>
<tr>
<td>M4</td>
<td>0,929</td>
<td>1,322</td>
</tr>
<tr>
<td>M5</td>
<td>31,900</td>
<td>34,755</td>
</tr>
<tr>
<td>M6</td>
<td>2,116</td>
<td>2,807</td>
</tr>
<tr>
<td>M7</td>
<td>38,458</td>
<td>0,768</td>
</tr>
<tr>
<td>M8</td>
<td>0,655</td>
<td>1,048</td>
</tr>
<tr>
<td>M9</td>
<td>73,000</td>
<td>59,473</td>
</tr>
<tr>
<td>M10</td>
<td>0,358</td>
<td>0,285</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>QUALITY APPECTS</th>
<th>Publishing domain - average</th>
<th>All domains - average</th>
</tr>
</thead>
<tbody>
<tr>
<td>QA 1</td>
<td>0,387</td>
<td>0,374</td>
</tr>
<tr>
<td>QA 2</td>
<td>0,005</td>
<td>0,022</td>
</tr>
<tr>
<td>QA 3</td>
<td>0,280</td>
<td>0,214</td>
</tr>
<tr>
<td>QA 4</td>
<td>0,071</td>
<td>0,072</td>
</tr>
<tr>
<td>QA 5</td>
<td>0,667</td>
<td>0,117</td>
</tr>
<tr>
<td>QA 6</td>
<td>0,908</td>
<td>0,959</td>
</tr>
</tbody>
</table>

**LIMITATIONS AND THREATS TO VALIDITY**

This research has limitations that have to be identified and discussed. Only one database was used (science direct) and only 10 XML schemas were evaluated due to comparability with other domains.
(for each domain 10 XML schemas were used). There is also a possibility of a human error and shortcomings of the used measurement tool are possible (Pušnik et al., 2013). The limitations are also connected to XML schema patterns respectively (un)reachability of all included or imported XML schemas. To confirm or disregard validity of results, the research should be repeated, based on other empirical research methods.

This paper does not include DocBook domain, although adopted by many publishing companies and common in the publishing field, as an extended use of DocBook is proposed in (Şah & Wade, 2012). The OASIS DocBook Schema for Publishers is an official variant of DocBook v5.x, specifically designed to support the publishing industry (Walsh., 2011) and is subjected to investigation of how well it supports interactive and multimedia content (Project, C-level, Spring, Supervisor, & Examiner, 2015). Authors pointed out the challenge of no standard nor best practices for writing documents on any subject and the publishers take different approaches with very different solutions. Therefore, in this research an investigation of the existing publishing field was launched, trying to include random and average XML schemas in publishing and DocBook was not included.

CONCLUSION AND FUTURE WORK

The paper addresses publishing domain issues through XML schemas, its characteristics, influence and contribution to organization, focused on assessing the quality of used XML schemas in the publishing field. The literature review reviled the importance of XML schemas within the publishing field and an experiment provided data to compare quality with other XML schemas and identify shortcomings that need to be addressed.

We have reused existing metrics for quality evaluation by comparing the 10 publishing XML schemas to 200 XML schemas of different origin. 6 aspects of quality were defined, combined into one quality metric, the quality index. The quality aspects include (1) structure, (2) transparency, (3) optimality, (4) minimalism, (5) reuse and (6) integrability. We have discovered that XML schemas from the publishing field are above average, providing an answer to the research question: the publishing domain does use XML schemas, the quality of them is above average however they still need to be improved mostly in the quality aspect of transparency and documentation. More detailed impact of XML schema quality is yet to be empirically confirmed and was included as an assumption in the paper.

Future work will extend the domains, where XML schemas will be evaluated. 20 domains have already been investigated, the publishing domain being the 21st. Additional domains will be further explored and compared as well additional XML schemas will be included in the experiment set for specific domains. Versions and the quality movement when changing an existing XML schema will also be explored.

ACKNOWLEDGMENTS

This joint work is enabled by bilateral project “Multidimensional quality control for e-business applications” between Serbia and Slovenia (2016-2017). Furthermore, the last author was partially supported by the Ministry of Education, Science, and Technological development, Republic of Serbia, through project no. OI 174023.

REFERENCES


Aparicio, F., De Buenaga, M., Rubio, M., & Hernando, A. (2012). An intelligent information access system assisting a case based learning methodology evaluated in higher education with medical


Combining Agile and Traditional Methodologies in Medical Information Systems Development Process

PETAR RAJKOVIC, University of Nis, Faculty of Electronic Engineering
IVAN PETKOVIC, University of Nis, Faculty of Electronic Engineering
ALEKSANDAR MILENKOVIĆ, University of Nis, Faculty of Electronic Engineering
DRAGAN JANKOVIC, University of Nis, Faculty of Electronic Engineering

Having experience with different traditional and agile methodologies in medical information system development and upgrade projects, we support the statement that no single methodology can be used with its full potential in all cases. Instead, mentioned methodologies have their own place in complex software projects and within certain category of sub-projects give the best results. In this paper we presented the guideline on choosing adequate methodology in different development and upgrade projects. All examined projects are defined within the scope of larger medical information system called Medis.NET developed by our research group, and deployed in 30 healthcare institutions. The bottom-line is that during requirement collection we rely on model and feature driven development supported by automatic code generation tools, which lead to faster prototype generation. Next, dividing the set of functionalities to sub-projects we group them in three categories – core projects, user-oriented, and loosely coupled application. For core projects we find traditional methodologies suitable since their outcome is a stable process and rich documentation. User-oriented sub-projects are developed under standard SCRUM, while for loosely coupled applications we chose Lean. We measured the effects of chosen approaches comparing time spent against estimated time for development, and by analyzing code metrics. Code metrics were automatically generated by our development environment, Microsoft Visual Studio, and we take into account few major values such as maintainability index, lines of codes, class coupling and cyclomatic complexity. In addition, we propose an adaptation for Lean model which gave us promising results in software upgrade projects. The proposed approach put additional responsibility to lead architects, since have to properly identifies all pros and cons for different methodologies and, knowing this well, wisely choose the right methodology for the right sub-project.

Categories and Subject Descriptors: H.4.0 [Information Systems]: General; D.2.8 [Software Engineering]: Metrics; D.2.2 [Software Engineering]: Design Tools and Techniques

General Terms: Design, Management

Additional Key Words and Phrases: Traditional software development, Agile software development, Lean, SCRUM

1. INTRODUCTION AND MOTIVATION

Since 2002 our research group was involved in many development and upgrade projects targeting medical information systems (MIS) [Rajkovic et al. 2005, 2013]. Initially, we relied on standardized traditional methodologies such are Waterfall, Spiral [Boehm 1988] and Rational Unified Process (RUP) [Kruchten 2004]. The main advantage of traditional methodologies was, and still is, a concise set of guidelines and significant volume of produced documentation [Leau 2012]. The main problem were long phases resulting with late response to potential changes requested by the end user. By introducing agile methodologies [Beck et al. 2001], primarily SCRUM [Schwaber et al. 2002], that require more interaction with the end users, we managed to reduce the number of new, surprising requests coming in the later stages of the project [Black et al. 2009]. On the other side, we faced with the set of new and unexpected problems – problems with lower volume of generated documentation as...
well as problems with social relations [Leau 2012]. For some users we become annoying and tedious, and some of our team members feels uncomfortable due to the frequent interaction with the clients.

With standard methodologies we realize that the stress level increases when project ends the development and has to enter to testing phase. With agile, this problem was also present, but stress-full intervals are scattered during the whole process and last shorter. After SCRUM, we looked for Lean methodology [Poppendieck 2003]. The main advantage was the task-based organization of development process. Having larger number of smaller tasks, allowed us to group them in logical packages and move these packages through Kanban table with different speed. This resulted in continuous system integration and our users were not so flooded with demo and test sessions.

When our first research and development project was about to finish in 2007, we had several pieces of software already finished and had the experience with few different software development methodologies (SDM). Based on that experience and lessons learned, we actually stop looking for a single methodology that fits all the cases of development and upgrade, but define the set of guidelines that would help us in choosing the adequate methodology for each sub-project. In 2008, we started with the preparation for new development project which primary output was a MIS dedicated for Serbian primary and ambulatory care. Defined set of the guidelines for choosing appropriate methodology we applied in mentioned development [Rajkovic et al. 2009]. Complete project consists of 33 development and upgrade subprojects run under different SDMs.

In related work section we described the most important works that were a great support to our idea. Next section this paper presents our guideline for choosing proper methodology for proper sub-project. Since we were encouraged with initial results, we decided to propose a modification of Lean approach by incorporating some SCRUM concepts, which is described in separate section. In result section, we evaluated our development and upgrade projects that was carried under Medis.NET software project. The evaluation is based on the code metrics provided by Microsoft Visual Studio and project timeline analysis.

2. RELATED WORK

The work presented in [Leau et al. 2012] clearly describes all the advantages and disadvantages for traditional and agile SDMs, claiming that adoption of proper methodology is crucial for governing software life cycle and later system acceptance. We faced with all of situations described in [Leau et al. 2012] in our projects developed in period 2002-2007, and start going to the idea that should not look to one almighty methodology, but to identify different types of projects and the methodologies which would lead to the most successful scenario. Due to frequent changes in many areas of stakeholder requests, as well as in legislative, in period of almost 15 years, we had enough opportunities to check different methodologies.

Using metrics to evaluate software projects maintainability is not a new idea. Since well-known work of [Coleman et al. 1994], many different studies were conducted. In that branch, we found [Doherty 2012] as important for our research. This paper presents analysis on the most important success factors for traditional and agile methodologies. This analysis shows that not methodologies themselves, but related factors such are user interaction level, team competence, or quality of collected requirements have bigger effect on development process organization and later system acceptance. In that light, it is easier to connect SDM with mentioned factors, and based on them chose a proper methodology.

Introducing hybrid development methodologies is also a topic discussed in literature. For example, [Runeson 2005] discusses introduction of agile methodologies to traditional stage gate model. According to this work, the role of gate model is in strategical planning, while agile approach suits better for “microplanning, day-to-day work control, and progress reporting”. Finding the suitable place for agile methodologies within large scale projects is described in [Lindvall et al. 2004]. Authors based the research on the data gathered from large companies (Daimler Chrysler, ABB, Nokia) and analyzed their experience. The general conclusion is that agile methodologies can match the needs for some specific types of the projects, while the major challenge lies in “efficient integration the agile project
into overall environment”. During the last decade agile methodologies developed rapidly and latest researches conducted on overall use of SDMs show significant advantage of SCRUM versus Waterfall [Shane et al. 2015]. Unfortunately this research shows only overall statistics, but not drilled down to specific domains of developed projects.

Since we decided to use spiral model, SCRUM and Lean as base development methodologies, we rely on Model Driven Development (MDD) in requirement collection, and on Feature Driven Development (FDD) in prototype building phase. Works presented in [Torchiano et al. 2013] and [Loniewski et al. 2010] gave us well argumentation for such decision. Beside we tend to use MDD also in later stages, mentioned three methodologies are the ones that we followed in our projects. Another two important aspects for optimal methodology utilizations are planning [Virani et al. 2014] and customer communication [Korkala et al. 2010]. These two “environmental” variables should also be considered when it comes to choosing the methodology for specific project. In general, many sources advocating usage of mixed and/or blended methodologies can be found. For example, we rely on [Okada et al. 2014] and [Hoda et al. 2013] when defining the updated lean model for software upgrade projects.

3. GUIDELINES FOR CHOOSING A METHODOLOGY FOR DEVELOPMENT PROJECT

Basically, every SDM we used in our first development project (Waterfall/Spiral/RUP, SCRUM and Lean) has its own advantages and disadvantages [Black 2009]. For this reason, we defined a set of guidelines that should help us in defining cases when some SDM should be applied. The summary of the guidelines contains following statements:

- During requirement collection phase, it is necessary to start building domain models and clearly identify main features
- Use component generation tools in order to reduce development time
- Identify extension points in created model as earlier as possible
- Categorize sub-projects as core projects, user-oriented (frontend) and loosely coupled components
- For core projects use traditional or agile methodology familiar to all assigned team members
- Frontend project develop under SCRUM to ensure proper communication with end users
- For loosely coupled components use Lean
- When it comes to team members - take care not only on technical, but also social skills, and assign people accordingly
- If needed, experiment with new development approach only in low to medium scale fronted and loosely coupled
- Do not experiment with new development approach in mission critical or core components

We started applying it in year 2009 when development of Medis.NET began. Thanks to the earlier projects, we had enough domain and process knowledge to define basic entities for future system. Also, we had developed domain models as well as model based code generation tools that should help us in overall improve our development process. Mentioned MDD tools are intensively used during requirement specification and prototype building phase [Rajkovic et al. 2015]. Once, when requirement collection phase is finished and customer signs request document, we already have initial advantage, since, until that moment, our domain models are on the way to be finished, system architecture is defined and the most of sub-projects are identified and implementation technology is chosen.

Next step is to define sub-projects and to divide them in three major groups – front-end projects, back-end projects and loosely coupled components. As front end projects we categorize these that are rich with GUI elements and forms. The focus of future’s everyday use, from the customer point of view, will be on them. For this reason, it is important to maintain constant contact with prospective users. These sub-projects are developed under SCRUM methodology. Not directly exposed to the customers are back-end related functionalities containing business logic and data processing functionalities. The projects belonging to this category make backbone of the system. Due to the importance, the most detailed documentation is required for this category of the projects. For this
reason, we usually choose some of traditional methodology based on a strong set of rules. Another advantage that traditional SDMs brought to this category of the projects are long testing and verification phases which are necessary for the most critical parts of the complete system [Black et al. 2009].

For partially independent and loosely coupled sub-systems, we choose Lean as a compromise between SCRUM and traditional methodologies. In MIS environment, to this category belong projects that are developed for some specific department usually requiring higher level of interaction with third party software and/or medical equipment. Next, common request is that these applications can work disconnected from the central MIS. The best examples for this type of the projects are applications for radiology and biochemical laboratory. Lean methodology looked as the good balance between SCRUM and traditional in this particular case. Equipment integration would be usually divided in sub-projects defined per communication protocol, which in reality means per device, or with larger institutions per devices that belongs in the same generation of certain manufacturer. In the scope of lean methodology all the tasks related to same communication protocol are grouped in the same work packages. Developed functionalities are then tested package by package, and the interaction with the users during the development process is reduced comparing to SCRUM.

4. LEAN MODEL ADAPTATION FOR SOFTWARE UPGRADE PROJECTS

The statuses that one task passes through according to the Lean model are: definition and moving to backlog, approving and moving to to-do list, designing, developing, testing and moving to the list of finished tasks. Initially, we fully relied on original lean approach. The good side of this approach was a possibility to group tasks in logical packages, and to change their statuses according to Kanban table. Grouping tasks to logical packages allowed us incremental development and partial delivery to the customer. On the other side, the main problems came in integration testing when discovered bugs can trigger more work needed than expected causing partial or even overall deadline extensions. Another weak spot, from our point of view, is moving tasks to to-do list. In the moment when task reaches to-do list, only general requests are known, and, often, followed with even more general assumptions. Due to this fact, design phase would then last longer, since design phase for upgrade projects need to include not only new functionalities, but also integration to existing system. This left a lot of space for misinterpretation and false assumptions. Even worse, this can happen also during development and testing phases. In upgrade projects, this kind of problems is common and even expected. This, it is important to identify them as earlier as possible and the react according the case.

For mentioned reasons we proposed the update of Lean approach for upgrade projects (Fig. 1). The proposed improvement is based on changing the role of design phase. Design phase will be split and partly incorporated into to-do list creation and partly to development phase. Looking at the design phase itself, we identified two major parts – functionality design and integration design. Designing integral parts for the new components is proportional to their integration level. Thus, we considered this part of the designing phase is one of the most critical. In order to reduce the possibility for errors in later stages, this part of design phase is moved to to-do list creation.

The cost of this decision is slower forwarding tasks through Kanban table in initial phases, but the benefit is visible later when missing links are rarely identified. Another benefit we notice is an effective introduction of new team members to the functionality of the existing system. Working together with experience colleagues they are able to get the insight to the initial system and to better understand the role of the update they will work on. Additional set of tasks moved to this phase is the improvement of software’s domain model (if needed) and the initial generation of the components that could be automatically generated. Now, when task is moved to to-do list, it has attached the document and code snippets needed for later development and the integration. Design phase now remains only with high-level design of new functionality followed with risk assessment. Detailed design is moved to development phase. Development phase is enriched also with initial testing that should bring verified component to the next stage of the process. Enriching the development phase by detailed planning on the beginning and by functional testing at the end, results in sub-projects consisting of several tasks
that can be treated as small software projects. These small software projects are then developed using SCRUM paradigm. We tend to keep these sub-projects as SCRUM guided projects with just few sprints that are responsible for development of integration ready components.

The next stage is testing. Since the functionality testing is done in development phase, the main part of testing phase is now integration testing. Since, in many cases functionality testing includes some segments of integration testing, this part is often used for internal acceptance test. On this way, the majority of bugs is identified earlier in development stage, so in ideal case, this step is more a formal verification. After this part is successfully finished, product is moved to the hands of prospective users for their acceptance testing. The main benefit of proposed update is lower number of moves back in the Kanban table, which results with faster development. Also, created code is with higher maintainability index (MI) easing future maintenance and further upgrades. In next chapter the results we get for projects developed under updated Lean are compared with these developed with standard Lean.

5. RESULTS AND DISCUSSION

With first MIS systems we developed, in period 2002-2006, we got the opportunity to test several SDM both for development and upgrade projects. As it has been explained in previous section, we decided to rely on MDD and FDD in early stages of development, but use traditional (Waterfall/Spiral), SCRUM and Lean for software development itself. [Atalag et al. 2014]. To calculate the effects of different methodologies we used standard code metrics offered by Microsoft Visual Studio development environment to check the code maintainability. To check development efficiency, we used ration between estimated and time needed for development. For calculating time wise efficiency, we used our projects logs and for code metrics calculation we used embedded code analysis tools that came as the standard part of Microsoft Visual Studio [Microsoft Library 2015]. As the leading parameter for the discussion, MI was used. Along with it, class coupling, number of lines of codes, cyclomatic complexity (CC), used methodology and project type are presented. We used our SDM assignment approach for total 33 sub-projects within our MIS (Table 1). CC is a measure of structural code complexity. Since it represents number of paths in the program, lower values are better. Higher CC results in harder code understanding and larger number of tests needed. CC should not be evaluated separately of the number of the lines of code. Higher number of lines of code will usually result in higher complexity number, so the relevant could be the ratio between complexity and lines of codes. As this value is lower, the code should be easier to maintain. At the end, the parameter named MI is a value in 0-100 range that should describe the relative easiness of code maintenance. Higher value is better. Microsoft states that minimal value claiming good maintainability is 20. Based on our experience, all the projects having this index lower than 50 are not quite easy maintainable especially for new team members. The formula for MI (taken from Microsoft Development Network) used in Visual Studio code analysis tool looks like:

\[ \text{Maintainability Index} = \text{MAX}(0, (171 - 5.2 \times \ln(\text{Halstead Volume}) - 0.23 \times \text{Cyclomatic Complexity}) - 16.2 \times \ln(\text{Lines of Code})) \times 100 / 171 \]

Both best ratio between time used versus estimated (UvE) and the best overall MI we got with projects developed under Lean methodology. Between SCRUM and traditional methodologies,
traditional methodologies gave us slightly better values for development, while SCRUM was significantly better with upgrade projects. Looking the difference between development and upgrade project Lean and SCRUM gave better results with upgrade than development projects. On the other side, usage of standard methodologies in upgrade projects resulted with significantly lower MI.

Table 1 Medis.NET MIS – Comparison of the code quality for different projects (Columns: MI – maintainability index, LoC – lines of code, CC – cyclomatic complexity, RCC – relative cyclomatic complexity (CC/LoC), UM – used methodology, PT – project type, UvE – time used vs time estimated displayed as percentage)

<table>
<thead>
<tr>
<th>Project Name</th>
<th>MI</th>
<th>LoC</th>
<th>CC</th>
<th>RCC</th>
<th>UM</th>
<th>PT</th>
<th>UvE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MedisControls</td>
<td>81</td>
<td>18131</td>
<td>4513</td>
<td>0.25</td>
<td>SCRUM</td>
<td>Dev</td>
<td>1.02</td>
</tr>
<tr>
<td>MedisData</td>
<td>86</td>
<td>19772</td>
<td>11115</td>
<td>0.56</td>
<td>Traditional</td>
<td>Dev</td>
<td>0.96</td>
</tr>
<tr>
<td>MedisDataAcquisitionLab</td>
<td>76</td>
<td>489</td>
<td>175</td>
<td>0.36</td>
<td>Lean</td>
<td>Upgrade</td>
<td>1.95</td>
</tr>
<tr>
<td>MedisDental</td>
<td>64</td>
<td>2293</td>
<td>582</td>
<td>0.25</td>
<td>SCRUM</td>
<td>Upgrade</td>
<td>0.88</td>
</tr>
<tr>
<td>MedisEHRCore</td>
<td>54</td>
<td>21571</td>
<td>2499</td>
<td>0.12</td>
<td>Traditional</td>
<td>Dev</td>
<td>0.91</td>
</tr>
<tr>
<td>MedisEHREditDialogs</td>
<td>61</td>
<td>25693</td>
<td>4422</td>
<td>0.17</td>
<td>SCRUM</td>
<td>Dev</td>
<td>1.12</td>
</tr>
<tr>
<td>MedisEHREditDialogsGin</td>
<td>57</td>
<td>1295</td>
<td>153</td>
<td>0.12</td>
<td>SCRUM</td>
<td>Dev</td>
<td>1.45</td>
</tr>
<tr>
<td>MedisEHREditDialogsIZIS</td>
<td>71</td>
<td>255</td>
<td>45</td>
<td>0.18</td>
<td>Modified Lean</td>
<td>Upgrade</td>
<td>0.75</td>
</tr>
<tr>
<td>MedisEHREditDialogsVisiting</td>
<td>44</td>
<td>10284</td>
<td>521</td>
<td>0.05</td>
<td>SCRUM</td>
<td>Upgrade</td>
<td>1.15</td>
</tr>
<tr>
<td>MedisEHREditDialogsSportMed</td>
<td>29</td>
<td>1266</td>
<td>30</td>
<td>0.02</td>
<td>Traditional</td>
<td>Upgrade</td>
<td>4.33</td>
</tr>
<tr>
<td>MedisEHREditDialogsStandards</td>
<td>62</td>
<td>1412</td>
<td>111</td>
<td>0.08</td>
<td>SCRUM</td>
<td>Upgrade</td>
<td>0.85</td>
</tr>
<tr>
<td>MedisEHREditDialogsDocuments</td>
<td>55</td>
<td>3706</td>
<td>435</td>
<td>0.12</td>
<td>Lean</td>
<td>Upgrade</td>
<td>1.04</td>
</tr>
<tr>
<td>MedisEHRFactory</td>
<td>66</td>
<td>280</td>
<td>67</td>
<td>0.23</td>
<td>Traditional</td>
<td>Dev</td>
<td>0.55</td>
</tr>
<tr>
<td>MedisFinanceReporting</td>
<td>77</td>
<td>10709</td>
<td>1967</td>
<td>0.18</td>
<td>Lean</td>
<td>Dev</td>
<td>0.92</td>
</tr>
<tr>
<td>MedisDoctorAssignment</td>
<td>77</td>
<td>1136</td>
<td>188</td>
<td>0.17</td>
<td>Lean</td>
<td>Dev</td>
<td>0.61</td>
</tr>
<tr>
<td>MedisIZISCore</td>
<td>89</td>
<td>1163</td>
<td>425</td>
<td>0.37</td>
<td>Modified Lean</td>
<td>Upgrade</td>
<td>0.89</td>
</tr>
<tr>
<td>MedisIZISExchange</td>
<td>83</td>
<td>658</td>
<td>276</td>
<td>0.42</td>
<td>Modified Lean</td>
<td>Upgrade</td>
<td>0.76</td>
</tr>
<tr>
<td>MedisCommunicationService</td>
<td>58</td>
<td>3989</td>
<td>641</td>
<td>0.16</td>
<td>SCRUM</td>
<td>Dev</td>
<td>1.01</td>
</tr>
<tr>
<td>MedisConfiguration</td>
<td>61</td>
<td>6454</td>
<td>838</td>
<td>0.13</td>
<td>Traditional</td>
<td>Dev</td>
<td>1.02</td>
</tr>
<tr>
<td>MedisLabIS</td>
<td>61</td>
<td>7506</td>
<td>1291</td>
<td>0.17</td>
<td>Lean</td>
<td>Dev</td>
<td>1.09</td>
</tr>
<tr>
<td>MedisOccupational</td>
<td>48</td>
<td>6042</td>
<td>324</td>
<td>0.05</td>
<td>Traditional</td>
<td>Upgrade</td>
<td>2.19</td>
</tr>
<tr>
<td>MedisTimeSchedule</td>
<td>67</td>
<td>3720</td>
<td>531</td>
<td>0.14</td>
<td>SCRUM</td>
<td>Upgrade</td>
<td>0.79</td>
</tr>
<tr>
<td>MedisDocumentsExchange</td>
<td>78</td>
<td>562</td>
<td>139</td>
<td>0.25</td>
<td>SCRUM</td>
<td>Upgrade</td>
<td>0.84</td>
</tr>
<tr>
<td>MedisFinance</td>
<td>42</td>
<td>1005</td>
<td>80</td>
<td>0.08</td>
<td>SCRUM</td>
<td>Dev</td>
<td>0.99</td>
</tr>
<tr>
<td>MedisReception</td>
<td>63</td>
<td>5600</td>
<td>951</td>
<td>0.17</td>
<td>SCRUM</td>
<td>Dev</td>
<td>1.06</td>
</tr>
<tr>
<td>MedisAccessControl</td>
<td>78</td>
<td>2693</td>
<td>941</td>
<td>0.35</td>
<td>Traditional</td>
<td>Dev</td>
<td>1.12</td>
</tr>
<tr>
<td>MedisRadiology</td>
<td>60</td>
<td>2328</td>
<td>298</td>
<td>0.13</td>
<td>Lean</td>
<td>Dev</td>
<td>0.97</td>
</tr>
<tr>
<td>MedisWorkOrganization</td>
<td>74</td>
<td>4853</td>
<td>956</td>
<td>0.20</td>
<td>Modified Lean</td>
<td>Upgrade</td>
<td>0.98</td>
</tr>
<tr>
<td>MedisReports</td>
<td>85</td>
<td>164</td>
<td>100</td>
<td>0.61</td>
<td>SCRUM</td>
<td>Dev</td>
<td>1.67</td>
</tr>
<tr>
<td>MedisSearchControls</td>
<td>66</td>
<td>4581</td>
<td>703</td>
<td>0.15</td>
<td>SCRUM</td>
<td>Upgrade</td>
<td>1.33</td>
</tr>
<tr>
<td>MedisInsuranceExchange</td>
<td>78</td>
<td>1312</td>
<td>467</td>
<td>0.36</td>
<td>Lean</td>
<td>Dev</td>
<td>2.01</td>
</tr>
<tr>
<td>MedisPatientScheduling</td>
<td>67</td>
<td>2538</td>
<td>528</td>
<td>0.21</td>
<td>SCRUM</td>
<td>Upgrade</td>
<td>0.89</td>
</tr>
<tr>
<td>MedisPatientDocumentation</td>
<td>62</td>
<td>4080</td>
<td>715</td>
<td>0.18</td>
<td>Lean</td>
<td>Dev</td>
<td>0.95</td>
</tr>
</tbody>
</table>

As it has been mentioned, we chose standard methodology twice for upgrade project. Once for sub-project for a department where (we believed) we had strong stakeholder request (MedisOccupational), while the another was the extension of EHR and corresponding GUI library for sports medicine department (MedisEHREditDialogsSportMed). With both cases we run into well-known problems with standard methodologies – we received too many update requests to late in project scope. Chasing the deadline, we ended up with on-site updates in fastest possible manner. Eventually, we get the projects with significantly lower MI. Luckily, project MedisEHREditDialogsSportMed having mentioned index of 29, has relatively small number of lines, so later bug fixes could be conducted without many problems. Problems with significant delays we experienced in projects where scope change and project extensions happened too often (such are reporting service and integration with
insurance fund). In this cases, chosen methodologies helped in a way that resulting code had acceptable MI, but the influence of selected methodology on overall delay was minor.

<table>
<thead>
<tr>
<th>Method</th>
<th>Number of projects</th>
<th>Maintainability index</th>
<th>Lines of code</th>
<th>Time needed vs time estimated</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Ovl</td>
<td>Dev</td>
<td>Upg</td>
<td>Ovl</td>
</tr>
<tr>
<td>Lean</td>
<td>12</td>
<td>6</td>
<td>6</td>
<td>71.08</td>
</tr>
<tr>
<td>Standard</td>
<td>8</td>
<td>6</td>
<td>2</td>
<td>68.25</td>
</tr>
<tr>
<td>Modified</td>
<td>4</td>
<td>0</td>
<td>4</td>
<td>79.25</td>
</tr>
<tr>
<td>SCRUM</td>
<td>14</td>
<td>7</td>
<td>7</td>
<td>63.93</td>
</tr>
<tr>
<td>Traditional</td>
<td>7</td>
<td>2</td>
<td>5</td>
<td>60.29</td>
</tr>
</tbody>
</table>

Development projects that we run under traditional methodologies were core projects which specifications were designed even before our Medis.NET started. The design was the result of lessons learned from our initial set of projects, and during development only few inputs were added, updated or deleted. This helped us in calm development of core system resulting with well-documented and highly maintainable code. Nevertheless, the average volume of these projects is more than 10000 lines of code and CC is kept on satisfied level. SCRUM approach gave us similar average results both for development and upgrade projects. Beside the average volume of development project is more than twice than with upgrade, all parameters describing code maintainability is on similar rank. For SCRUM approach, we could just confirm our old finding that SCRUM works well when both sides (developer and users) are dedicated to get final goal.

We got the best result in projects where Lean approach was applied. Initially we tested modified Lean approach on the project which outcome should be time-scheduling component. Comparing with the results we got from the projects of similar volume, but developed with SCRUM and Lean, we get slightly higher maintainability, but project was finished much faster and with lower bug rate during acceptance phase. Updated Lean gave us very optimistic results, and with use on future projects having higher volume of code and functionalities, we expect to prove it as feasible. Currently, we are working on the project that should integrate different MIS systems used in Serbian public healthcare where updated Lean is applied.

### 6. CONCLUSION

This paper presents our heuristic approach, based on the experience of our research group, for choosing proper SDM for specific development and upgrade projects within MIS systems. After 15 years long experience with MIS development we tend to believe that there is (still) not a silver bullet when it comes to methodologies supporting software life cycle. Having experience with different traditional and agile SDMs we support the statement that all of the mentioned methodologies have their own place in complex software projects and within certain category of sub-projects give the best results. In this paper we presented our initial experience with traditional and agile models as well as the guideline how to choose adequate methodology in different MIS related development and upgrade projects. All examined projects are defined within the scope of larger MIS called Medis.NET developed by our research group. The bottom-line is that during requirement collection we rely on MDD and FDD approach supported by automatic code generation tools, which lead to faster prototype generation. Next, dividing the set of functionalities to sub-projects we group them in three categories – core projects, user-oriented, and loosely coupled application. For core projects we find that traditional SDM are suitable since their outcome should be stable process and rich documentation. User-oriented sub-projects are developed under standard SCRUM, while for loosely coupled applications we chose Lean.

We measured the effects of chosen approaches comparing time spent against estimated time for development, and by analyzing code metrics results. Code metrics were automatically generated by our development environment, and we take into account few major values such are MI, lines of codes, class coupling and CC. For development project we get satisfied results for each of three used
methodologies. Lower MI for projects developed under SCRUM is mostly the consequence of more frequent specification change due to more intense interaction with customers. For upgrade projects, SCRUM and Lean gave slightly better results, while the usage of standard methodologies in upgrade projects resulted with the significant overdue and extremely low MI in some cases. For upgrade projects we started specifying updated Lean approach based on combination of Lean and SCRUM. It is used in the latest upgrade projects and first results look promising. In all projects where it was used, the consequence was faster development and code with higher MI.

Next step in our research will be analysis of results from testing phase. In this paper, the focus was clearly on development, while the software verification, despite of its importance, was not covered in details. Extending presented results with testing, validation and verification analysis will help us in defining more detailed guidelines covering wider area of MIS life-cycle.

With this paper we wanted also to show that is important that software development process must be guided carefully, and that no methodology can be claimed as “the best for all purposes and in all cases”. It is necessary that software architect properly identifies all pros and cons for different methodologies and, knowing this well, wisely choose the right methodology for the right project. Also, it is important to state that there is the place to combine different methodologies and lead the process with blended or mixed methodology. With proper methodology choosing guidelines and the introduction of even new promising approaches, development and upgrade processes for large and complex software projects, similar to Medis.NET, should be more effective, less stressful and gave more benefits both to developers and end users.

REFERENCES


Ken Schwaber and Mike Beedle. 2002. Agile Software Development with Scrum


How is Effort Estimated in Agile Software Development Projects?

TINA SCHWEIGHOFER, University of Maribor
ANDREJ KLINE, msg life odateam d.o.o.
LUKA PAVLIČ, University of Maribor
MARJAN HERIČKO, University of Maribor

Effort estimation is an important part of every software development project. Regardless of whether the development disciplines are traditional or agile, effort estimation attempts to systematically attach to other development elements. It is important to estimate the work load at the very beginning, despite the initial drawback of there being very little known about the project. And if, in addition, the effort estimations are accurate, they can contribute a lot to the success of a project being developed.

There are many approaches and methods available for performing effort estimation, each one with their own features, as well as pros and cons. Some of them are more appropriate for traditional software development projects, while others are meant for agile software development projects. The latter is also the subject of the systematic literature review presented in the article. Based on the set research questions, we researched the area of effort estimation in an agile software development project. Which methods are available, how objective the estimation is, what influence the estimation and most importantly how accurate those methods and approaches are. The research questions were answered and the basis for future empirical work was set.

Categories and Subject Descriptors: D.2.8 [Software Engineering] Metrics—Performance measures; Process metrics; Product metrics

Additional Key Words and Phrases: Effort Estimation, Estimation Accuracy, Agile, Software Development, SLR

1. INTRODUCTION

Effort estimation is the first of many steps in the software development process that can lead to a successful project's completion. It is a complex task, that constitutes the basis for all subsequent steps related to planning and management.

Effort estimation is also a very important part in agile software development projects. In order to achieve the highest possible levels of accuracy, software development teams can make use of different techniques, methods and approaches, including advising group effort estimation [Molokken and Jorgensen 2003], subjective expert judgement [Trendowicz and Jeffery 2014] and planning poker [Cohn 2005]. In a variety of approaches and methods used, different questions arise: What are the pros and cons of approaches for effort estimation, how can different models be applied to a different development environment and specific development team, and most importantly, how accurate is effort estimation when using a specific method or approach. The average error in effort estimation is measured between

Authors’ addresses: T. Schweighofer, Faculty of Electrical Engineering and Computer Science, University of Maribor, Smetanova 17, 2000 Maribor, Slovenia; email: tina.schweighofer@um.si; A. Kline, msg life odateam d.o.o., Titova 8, 2000 Maribor, Slovenia; email: andrej.kline@msg-life.com; L. Pavlič, Faculty of Electrical Engineering and Computer Science, University of Maribor, Smetanova 17, 2000 Maribor, Slovenia; email: luka.pavlic@um.si; M. Heričko, Faculty of Electrical Engineering and Computer Science, University of Maribor, Smetanova 17, 2000 Maribor, Slovenia; email: marjan.hericko@um.si

Copyright © by the paper’s authors. Copying permitted only for private and academic purposes.
20 and 30 percent [Abrahamsson et al. 2011; Grapenthin et al. 2014; Jørgensen 2004; Kang et al. 2010; Haugen 2006]. Therefore, the attempt to lower the number of errors in the accuracy rate is welcome.

The research area of our work is effort estimation and its accuracy in agile software development projects. In this article, a systematic literature review is presented. The relevant literature is presented and the answers for set research questions are given. The literature review also represents the research basis for our future work – an empirical study looking into the accuracy of effort estimation in a real-life industrial environment.

The content of the work is organized as follows. First, the theoretical background about effort estimation and effort estimation in an agile software development project is presented. Next is the main research, systematic literature review, together with results and a discussion. In the end the conclusion and future work is presented.

2. EFFORT ESTIMATION PROCESS

Effort estimation process is a procedure in which effort is evaluated and estimation is given in the context of the amount and number of needed resources by which we are able to end project activity and deliver a service or a product that meets the given functional and non-functional requirements to a customer [Trendowicz and Jeffery 2014].

There are many reasons for performing effort estimation. As presented in [Trendowicz and Jeffery 2014] it is performed to manage and reduce project risks, for the purpose of process progress and learning within an organization, for finding basic guidelines and productivity measurement, for the negotiation of project resources and project scope, to manage project changes and to reduce the amount of ballast in project management.

When choosing the most appropriate approach for effort estimation we have to be aware that delivering accurate estimations contributes to proper project decisions. Project decisions are important for short-term observation (within the context of one project) and for long-term observation where it can encourage the progress and effectiveness of work done within a development team and organization as a whole [Grimstad et al. 2006]. According to literature on the subject, 5 percent of the total development time should be aimed at effort estimation [Trendowicz and Jeffery 2014].

When estimating effort in agile development projects we can come across different challenges. We have to make a decision about which strategy of effort estimation we need to choose, how to connect good practices of agile development with efficient effort estimation and which factors have the most influence on the accuracy of the estimated effort.

3. SYSTEMATIC LITERATURE REVIEW

For the purposes of finding appropriate answers to a given problem, a systematic literature review was chosen as a research method. We tried to identify, evaluate and interpret all available contributions relevant to our research area [Kitchenham and Charters 2007].

3.1 Research questions

Within the researched area, the following research questions were formed:

*RQ1*. Which agile effort estimation methods are addressed?
*RQ2*. How objective is effort estimation and how much of a subjective evaluation is present?
*RQ3*. Which factors most influence agile effort estimation?
*RQ4*. Which studies regarding agile effort estimations have been performed?
*RQ5*. How useful are the particular agile effort estimation methods in the agile planning process?
How is Effort Estimated in Agile Software Development Projects?

3.2 Search process

Based on the proposed research questions, search strings were formed and based on formed strings, the search for primary studies was carried out in selected digital libraries. The search string and selected digital libraries are presented in Table I. For the purposes of getting more exact results we used different search restrictions. In the ScienceDirect base we only searched in the abstracts, titles and keywords, in IEEE Xplore, Scopus and ACM Digital Library, in the abstracts of studies and in SpringerLink we restricted the discipline to Computer Science. The results obtained by the restricted search are presented in Table II.

3.3 Study selection

After the potentially relevant studies were identified in selected data sources with proposed search strings, two selection cycles were carried out. First, we reviewed the title, keywords and abstract of each study, according to inclusion factors (the study addressed ways for effort estimation in agile projects and accuracy of estimated effort according to the real effort spent) and exclusion criteria (the study is not in English or German and cannot be found in digital libraries). The studies that were identified as appropriate were reviewed as a whole and a final decision was made, whereby we selected the studies that provided answers to research questions.

After the first selection cycle, 40 primary studies were selected and after the second selection cycle 12 relevant primary studies were selected for further analysis. Among primary studies, 4 of them were journal articles and 8 of them are conference papers published in conference proceedings. A detailed distribution with associated data sources and references is presented in Table III.

4. RESULTS AND DISCUSSION

4.1 Methods for effort estimation in agile software development

Many effort estimation methods in agile software development can be found. Among the found methods and techniques, the majority used subjective expert effort estimation. This includes techniques such as planning poker, expert judgement and story points [Usman et al. 2014; Mahnić and Hovela 2012;
Nguyen-Cong and Tran-Cao 2013; Torrecilla-Salinas et al. 2015; Jørgensen 2004; Popli and Chauhan 2014b; Haugen 2006; Popli and Chauhan 2014a]. Additionally, planning poker as a estimation method should be used in a controlled environment, with no boundary conditions, in a known domain and within a team where anyone can and dares to express their opinion.

Estimation by analogy was also frequently used [Abrahamsson et al. 2011; Grapenthin et al. 2014; Torrecilla-Salinas et al. 2015; Jørgensen 2004; Kang et al. 2010; Popli and Chauhan 2014b; Haugen 2006], done with the help of knowledge base [Nguyen-Cong and Tran-Cao 2013; Torrecilla-Salinas et al. 2015; Popli and Chauhan 2014b]. In that context, the tool that supports the approach can be established and maintained. It can be used for data input about conducted estimation cases and also the retrospective of the conducted estimation. The database can be used by the broader community of evaluators over an extended time period, which allows for the comparison of present estimation cases to examples of estimations and time used in an actual project from the related domain.

On the other hand, other methods and techniques used for effort estimation that are not based on experimental judgement or a group's approach to assessing are not so frequently used in agile software development projects. Those techniques are, for example, COCOMO (Constructive cost model), SLIM and regression analysis [Usman et al. 2014; Nguyen-Cong and Tran-Cao 2013; Torrecilla-Salinas et al. 2015; Jørgensen 2004; Popli and Chauhan 2014b].

After the detailed analysis, we did not find an answer as to how successful and efficient the use of proposed methods and techniques is.

4.1.1 RQ1 - Which agile effort estimation methods are addressed?. Most commonly used are different point methods, like story and functional points, user stories and expert judgements. Also, good practices of agile development, like pair programming, planning games, documentation in the form of user stories and other things that significantly contribute to the accuracy and quality of effort estimation, need to be properly taken into consideration.

4.2 The objectiveness of effort estimation

In selected primary studies, the objectiveness of effort estimation is identified in different ways. One of the reasons for this is that studies are carried out in different environments, whereas only five of them present real-life industrial cases [Abrahamsson et al. 2011; Grapenthin et al. 2014; Jørgensen 2004; Kang et al. 2010; Haugen 2006]. Additionally, not a lot of knowledge can be found about the success of a project from the aspect of accuracy of effort estimation. Therefore, the objectiveness of effort estimation is measured mainly according to subjective opinions of participants.

Estimation represent a subjective expert judgement [Usman et al. 2014]. Regardless of the used method for effort estimation, the final decision is made by one or more participants. Thus, they need to be experienced in the area in which they are performing estimation. As claimed in [Jørgensen 2004], subjective expert judgement is usually more accurate as formalized estimation models. Also, in domains where the team is experienced, the effort estimates are more reliable [Haugen 2006].

Authors [Inayat et al. 2015] are looking into multi-functional teams where experiences and knowledge of domain is distributed. Customer-on-site for prompt and on-going evaluation of the work done, lead to higher estimation accuracy [Inayat et al. 2015]. If human factors, like team experience and ability to manage projects, are on high level, then quality of estimate creation is high [Popli and Chauhan 2014a].

The approach of estimating stories, based on empirical knowledge base and using key words [Abrahamsson et al. 2011] is objective as much as entries in knowledge base from where the evaluator get their data.
4.2.1 **RQ2 - How objective is effort estimation and how much of a subjective evaluation is present?**

Subjective expert judgement is the most widespread method for effort estimation [Usman et al. 2014]. It is hard to assess the accuracy of effort estimation without any statistical measurement errors [Nguyen-Cong and Tran-Cao 2013]. Otherwise, the Kalman filter algorithm that is used to track the project’s progress, systematically summarizes the current errors, but the result very much depends on the given function points [Kang et al. 2010]. In conclusion, the proposed algorithm does not contribute much to the agile effort estimation.

4.3 Effort estimation factors

When modelling productivity and estimating project effort within software development, we need to be aware that present success does not necessarily guarantee future project success if projects are placed within a new context [Trendowicz and Jeffery 2014].

The effort needed for software development depends on factors, divided between context and scale factors [Trendowicz and Jeffery 2014; Trendowicz et al. 2008]. Context factors include: the programming language, application domain of the software, type of development and life cycle (methodology) of development. Scale factors include: the size of software, complexity of system interfaces and integration, project effort, project duration, maturity of software development process, the size and structure of the development team and the project budget [Trendowicz and Jeffery 2014].

Based on an analysis of selected primary studies factors influencing effort estimation were extracted. Presented factors are classified into four groups: personnel-, process-, product- and project factors, as presented in [Trendowicz and Jeffery 2014; Trendowicz and Münch 2009].

**Personnel factors**

Personnel factors are the characteristics of people involved in the software development project. They usually take into consideration the experience and capabilities of project stakeholders as development team members, as well as of software users, customers, maintainers, subcontractors, etc. [Trendowicz and Münch 2009]. In the analysed primary studies, the following personnel factors can be found:

—[Usman et al. 2014]: The team’s previous experiences, task size, efficiency and risk level of testing, domain and task knowledge.

—[Mahnič and Hovelja 2012]: Experiences and motivation of the development team, ability to combine different-knowledge developers.

—[Torrecilla-Salinas et al. 2015]: Team size, duration of iteration, experiences gained within an iteration, the achieved value of the tasks.

—[Jørgensen 2004]: Experiences and the knowledge of experts in the development team, the ability to combine experts, willingness and ability to educate new members regarding effort estimation.

**Process factors**

Process factors are connected with the characteristics of software development and also with the methods, tools and technologies applied during development [Trendowicz and Münch 2009]. The following process factors are found in primary studies:

—[Abrahamsson et al. 2011]: User stories description quality.

—[Nguyen-Cong and Tran-Cao 2013]: Need for measurement of statistical errors in the form of MRE and MMRE.

—[Grapenthin et al. 2014]: Accuracy of requirement knowledge during the project.

**Project factors**

Project factors include various qualities of project management and organization, resource manage-
ment, working conditions and staff turnover [Trendowicz and Münch 2009]. Among primary studies, the following project factors are found:

—[Inayat et al. 2015]: The use of agile practices (for example, cooperation with the customer, testing approach, retrospective, project organization and others).

—[Kang et al. 2010]: A common understanding of what one measuring point is, changing requirements from customers, new requirements, changing priority of existing requirements.

**Product factors**

Product factors describe the characteristics of software product being developed through all development phases. The factors refer to products as software code, requirements, documentation and others, as well as their characteristics [Trendowicz and Münch 2009]. In the analysed primary studies none of the product factors were found.

4.3.1 **RQ3 - Which factors most influence agile effort estimation?**. An analysis of primary studies shows that personnel factors come before project factors in the agile effort estimation process. This means that the level of knowledge and the experiences of experts in teams and also the way in which development teams are constructed, are crucial for effort estimation. The communication can improve estimates and reduce task changes during the projects [Grapenthin et al. 2014]. It is important to use data from past tasks and control lists for evaluation. The feedback regarding estimation also needs to be given and presented to all team members [Jørgensen 2004]. Based on all the findings, we can conclude that personal factors really are more important than project factors, which is also confirmed in [Popli and Chauhan 2014a].

4.4 Conducted studies in the area of agile effort estimation

Among analysed primary studies, different reports about conducted studies can be found. The studies vary by their scope, by the methods used for effort estimation and in particular by the end results. Some studies [Abrahamsson et al. 2011; Grapenthin et al. 2014; Jørgensen 2004; Kang et al. 2010; Haugen 2006] present cases from an industrial environment, while other studies [Usman et al. 2014; Mahnič and Hovelja 2012; Nguyen-Cong and Tran-Cao 2013; Torrecilla-Salinas et al. 2015; Inayat et al. 2015; Popli and Chauhan 2014b; 2014a] present cases from an academic environment. It can be seen that studies repeat and that a little knowledge about the conducted project of agile development can be found.

4.4.1 **RQ4 - Which studies regarding agile effort estimations have been performed?**. Many studies are available, but many of them repeat. Very little empirical knowledge is available. The accuracy of effort estimation is not very good, which can be seen by the amount of work that needs to be done and on the release date. Both evaluations are often missed as claimed in [Popli and Chauhan 2014b].

4.5 Methods for effort estimation in agile project planning and development

In primary studies, the most commonly used methods for effort estimation are XP [Usman et al. 2014; Abrahamsson et al. 2011; Nguyen-Cong and Tran-Cao 2013; Inayat et al. 2015; Kang et al. 2010; Popli and Chauhan 2014b; Haugen 2006; Popli and Chauhan 2014a] and SCRUM [Usman et al. 2014; Mahnič and Hovelja 2012; Nguyen-Cong and Tran-Cao 2013; Grapenthin et al. 2014; Torrecilla-Salinas et al. 2015; Inayat et al. 2015; Popli and Chauhan 2014b], which can be related and explained by their popularity and understanding in an agile development society. Some other methods are also found, but they do not constitute a significant share. Those methods include, for example: RUP (Rational Unified Process) [Nguyen-Cong and Tran-Cao 2013], Lean [Jørgensen 2004; Nguyen-Cong and Tran-Cao 2013], Hybrid method [Nguyen-Cong and Tran-Cao 2013] and Crystal [Popli and Chauhan 2014b].
Important to note is that some of the primary studies go back to the year 2002 and beyond. This causes some methods of effort estimation, now used in agile effort estimation, did not yet get their nowadays names, since agile methods were then in Europe at the beginning of recognition and use.

4.5.1 **RQ5 - How useful are the particular agile effort estimation methods in the agile planning process?**. The planning poker works well when evaluating smaller tasks [Mahnič and Hovelja 2012], also user stories combined with the knowledge base give more accurate results for smaller tasks [Abrahamsson et al. 2011]. As mentioned in [Torrecilla-Salinas et al. 2015] function points and summary of time used is presented in the context of an agile web project together with an empirical report. It is important to adjust to the latest findings regarding effort estimation, which means continuous learning [Torrecilla-Salinas et al. 2015]. Using agile development practices can contribute to higher accuracy in effort estimation [Inayat et al. 2015].

5. CONCLUSION
The area of effort estimation in agile software development was researched based on the proposed research questions. Many studies can be found in different articles, but a lot of them repeat themselves. On the other hand, there are also quite a few articles that provide empirical knowledge about effort estimation. This is especially surprising since agile software development methods emerged as early as the year 2000 (in Europe).

As data extraction within a systematic literature review shows, the most widely used estimation methods are user card points, story points and functional points, user stories and experience-based experts’ estimation. The authors’ experience set personnel factors ahead of project factors, which means that for the estimation, the knowledge and skill level of the group experts is essential, as well as the ability to form proper working teams. Also, the studies prove that the usage of agile practices in the software development process, such as working in pairs (concept creation, testing, refactoring), planning games, user stories as well as documentation, all lead to a higher quality of effort estimation, and thus to more accurate estimations.

In the article [Nguyen-Cong and Tran-Cao 2013], many presented effort estimation models are not empirically proven and objectivity is measured from the viewpoint of the accuracy of effort estimation. Basically, this can be widespread to the others primary studies. The presented studies usually present a narrow business area, where many restrictions are pointed out, for example restrictions about the development team size, profile of evaluators, techniques used and, primarily, the duration of the presented studies. The duration is usually not long enough for an objective evaluation of effort estimation method usefulness.

The conducted systematic literature review did disclose a lot of options for future work. Among other reasons the review was conducted for the purpose of setting a theoretical background for an empirical study that will be done. The study will look into the accuracy of effort estimation in a real life industrial environment. It will track the estimation accuracy of user stories in an environment, where twenty five developers using extreme programming discipline, are tracked for a period of two years and measured by their software development effort estimation accuracy. Also, future work will be oriented towards an attempt to improve accuracy by creating a knowledge database of elapsed user stories and a retrospective of similar work done in the past. This was already mentioned by some authors in the primary studies that proposed the introduction of a knowledge base to be used in the effort estimation process. This was one of the triggers for carrying out an empirical study that will present and introduce such a knowledge base in a real life industrial environment.
REFERENCES


N. C. Haugen. 2006. An empirical study of using planning poker for user story estimation. In AGILE 2006 (AGILE'06). 9 pp.–34. DOI: http://dx.doi.org/10.1109/AGILE.2006.16


1. INTRODUCTION

Many deficiencies of a program can be invisible to the compiler. It optimizes the code and the resulting code could not reflect all the characteristics of the source code. The advantage of analyzing the machine code is, most certainly, the precision. But because we could greatly benefit from systems that could indicate the irregularities in the phase of writing the source code, static analysis has great significance. This is especially the case in very large systems, where the main difficulty would be finding the piece of code that is causing those irregularities. Static analysis helps us detect some commonly made mistakes very easily and therefore reduces time-to-market. However, we cannot completely rely on it. That means we cannot use it instead of testing or guarantee the quality of a program only by applying static analysis.

SSQSA framework gives us a great variety of possibilities for static analysis. Some of the planned functionalities were support for static timing analysis and Worst-Case Execution Time (WCET) estimation at code level [Wilhelm et al. 2008; Lokuciejewski and Marwedel 2009; Lokuciejewski and Marwedel 2011]. Worst-Case Execution Time is estimated as a part of timing analysis and provides the upper (worst) value of the execution time. It is important, since real-time systems must satisfy some time limitations. However, there are some problems regarding the estimation of the Worst-Case Execution Time. Some of them are making estimations on a program that does not have an ending (that contains infinite loops) or making predictions for input values. Estimating WCET would be much easier if the input values were known in advance or if we could say with most certainty which input values would cause the largest WCET results. By estimating WCET on various input values, it is possible to get very different results. That is because paths of a control flow graph are taken based on them and the difference in complexity between those paths could be very significant, which has direct influence on WCET value. Therefore, estimated WCET result is an estimation, which should be a very close approximation, but it must not be below the real WCET result. Despite these difficulties,
WCET estimation on source code level is also significant because it sometimes allows us to perform this estimation in the earliest phases of software development, accomplished e.g. by performing timing model identification using a set of programs and WCET estimate calculation using flow analysis and the derived timing model. That is important because if the final executable violates the timing bounds assumed in earlier stages, the result could be a highly costly system re-design [Gustafsson et al. 2009].

This work is mostly oriented towards language independence, which makes it uncommon when compared to other approaches in solving this problem. The expected outcome of this particular work was planned to be reflected in as successful estimation as possible, no matter the input language. This kind of analysis goes quite deep into detail and in order to perform the estimation in a thorough and proper manner, evolution of parts of the SSQSA framework is considered inevitable.

The rest of the work deals with the selection of approach and its implementation. Related work is provided in section 2. Section 3 deals with undertaken work for WCET estimation on control flow graphs. The obtained results are given in section 4. Limitations that were met are within section 5 and the conclusion is described in the last section.

2. RELATED WORK

Worst-Case Execution Time estimation is an ongoing problem and many different approaches have been taken in solving it. Here are some notable solutions.

aiT\(^1\) WCET Analyzers [Lokuciejewski and Marwedel 2011] offer a solution which determines WCET in tight bounds by using binary executable and reconstructs a control-flow graph. WCET determination is separated into several consecutive phases - value analysis, cache analysis, pipeline analysis, path analysis and analysis of loops and recursive procedures. The difference is in the approaches is using eCST over binary code before conversion to control-flow graph. Additionally, the analysis aiT performs on ranges of values, access to main memory and some other features are far more complex than what we have accomplished. aiT WCET analyzer is a commercial tool already used by some enterprises, mostly for embedded systems testing.

Bound-T\(^2\) is a tool that also uses machine code in its static analysis approach. Similarly, it uses control-flow graphs and call trees for WCET estimation. Therefore, it is also language-independent and does not require any special annotations in the source code. It also has an option to take into consideration user defined assertions, for example, upper loop count. Bound-T is cross-platform and can analyze code that targets multiple processors. It is a free, open-source tool. The main difference between Bound-T and SSQSA WCET analyzer prototype is that Bound-T uses compiled machine code for analysis, while SSQSA works with program source code.

SWEET\(^3\) (SWEdish Execution Time tool) is a research prototype tool with the flow analysis as the main function. The result obtained by flow analysis is information used further for WCET estimation. The analysis is performed on programs in ALF (ARTIST2 Language for WCET Flow Analysis) internal intermediate program language format. Different sources, such as C code or assembler code, could be transformed into ALF code, leading to generation of flow facts, further used in low level analysis or in WCET estimation. Using ALF was also an approach that was taken into consideration for implementation of WCET estimation in SSQSA, but was left because the estimation was already thought to be possible by converting the eCST structure.

Heptane\(^4\) static WCET estimation tool is a stable research prototype written in C++. However, it runs only on Linux and Mac OS X and supports MIPS and ARM v7\(^5\) architectures. SSQSA is written

\(^{1}\) [http://www.absint.com/ait/](http://www.absint.com/ait/)
\(^{2}\) [http://www.bound-t.com/](http://www.bound-t.com/)
\(^{3}\) [http://www.mrtc.mdh.se/projects/wcet/sweet/](http://www.mrtc.mdh.se/projects/wcet/sweet/)
\(^{4}\) [https://team.inria.fr/alf/software/heptane/](https://team.inria.fr/alf/software/heptane/)
\(^{5}\) MIPS is a processor architecture which is used for the purpose of digital home, home networking devices, mobile applications and communication products, while ARM (Advanced RICS Machine) is an entire family of architectures for computer
in Java and therefore it is possible to run its code anywhere. Heptane analyzes source code written in C and is not able to deal with pointers, malloc or recursion. SSQSA is language independent and is, in theory, able to deal with recursion. Heptane transforms the binary code into a control-flow graph in XML format. Afterwards, it is able to do low-level and high-level analysis, such as WCET.

Chronos is an open source software that deals with static analysis and WCET estimation. It includes both the data input and the hardware into WCET estimation. After gathering program path information (derived from source program or the compiled code) and instruction timing information, it performs WCET estimation. The difference from our approach is generating the control-flow graph from the binary code written only in C. On the contrary, in SSQSA control-flow graph is derived from eCST, created from the source code of the supported languages without compilation. The underlying hardware, however, is not considered in SSQSA WCET estimation.

3. WCET ESTIMATION IN SSQSA

For the needs of WCET estimation, two approaches were taken into consideration. The first one was using the existing universal intermediate structure, eCST (on which SSQSA framework is based) [Rakić 2015]. Another one was using ALF as a domain specific language [Gustafsson et al. 2009; Rakić and Budimac 2014]. ALF can be used as an intermediate language and several tools for converting popular programming languages code to ALF code already exist. ALF is specifically designed for flow analysis and WCET. It is also designed in such a way that it is very easy to use it for all kinds of code analysis.

Many languages (Java, C#, Modula-2, Delphi, ...) were already supported by SSQSA and they can be transformed into eCST structure. Since the WCET estimation without involving ALF appeared possible, the first approach was selected. Also, this meant the validation could be more precise by implementing WCET on eCST, since the sample of supported languages was bigger and there were only a few translators for ALF.

3.1 Conversion from eCST to eCFG

The work was done on a structure called eCST and its nodes. A node in eCST contains important pieces of information, stored in a token. It holds information on the node type, which is of high importance in conversion from eCST to eCFG (enriched Control Flow Graph), on which the WCET estimation would be later done. The approach was the following: using the Depth-First Search algorithm, eCST was traversed and its nodes were used for generating eCFG. It is important to mention that not all of the nodes are necessary for the conversion to eCFG. Only a subset of all the universal nodes have been selected as of importance to WCET estimation. Those included in the resulting eCFG are the following:

- COMPILATION_UNIT,
- FUNCTION_DECL,
- FUNCTION_CALL,
- VARDECL,
- STATEMENT,
- ASSIGNMENT_STATEMENT,
- LOOP_STATEMENT,
- BRANCH_STATEMENT,
- BRANCH,
- CONDITION.
An eCFG node stored the following information on the type of the node, optional additional information, whether a node is initial/final and its child nodes. Due to the need for later calculation of condition values in specific nodes (loop and branch), a simple symbol map was also implemented.

3.2 WCET analysis on eCFG

For the need of estimating the Worst-Case Execution Time, a wrapper around a simple node was made. It stored some additional information on the node: bestTime, worstTime and averageTime. Since the eCFG structure was simple for iteration, each of the nodes was easily analyzed. Usage of a symbol map made it possible to restore values necessary for the calculation of various conditions in loop and branch nodes.

The calculation is performed the following way: each of the eCFG nodes has a WCET value, which is added to the total value. However, WCET estimation on loop and branch statements is somewhat more complex. The value of nodes contained within the loop statement body is being summed and multiplied by the count of loop statement executions. Therefore, it is also necessary to perform evaluation of loop statement condition expression in order to estimate the number of executions. On the other hand, the branch statement’s WCET value is chosen by estimating separate WCET values of each branch. Upon separate estimations, the largest WCET value is considered the worst and assigned as branch statement’s WCET value.

For regular nodes (non-composite ones, all except branch and loop nodes), the worst and best times are equal to 1. This means that one line of code equals to one time unit. For more complicated (composite) nodes, the value depends on the related nodes. For branch nodes, the worst time is the worst time of the "worst" branch, while the best time is the best time of the "best" branch. For loop nodes, the worst case value is calculated by evaluating the conditional expression, which then becomes multiplied with the worst execution time of the inner-loop nodes. The best time of a loop node is always one, based on the fact that if the condition cannot be evaluated or if it is false, the loop body will not be executed. In all cases, the average time is calculated by dividing the sum of worst and best time by two. So, average time is the average value of best and worst case execution time.

3.3 Analyzing function declarations and function calls

Specification of function declarations and function calls which would enable establishing connections among numerous functions was necessary. The missing node that was a part of eCST, but not yet transformed and added into eCFG was the function call node. The problem with implementing this node was that many modern languages allow method (function) overriding, which means that the function name is not enough to identify which function is going to be called. To make this prototype work, a simple solution in form of a structure which holds all the function declarations and their respective parameter numbers was implemented. For this prototype, only the number of the arguments was considered and not their type, but with remark that this is needed for a more precise evaluation in the future. Afterwards, a function call hierarchy was produced and all the function calls were taken into consideration, which meant that the prototype was no longer bound to analyzing single methods in a compilation unit.

By adding function calls to eCFG, it was possible to create an interprocedural control flow graph (ICFG) [Lokuciejewski and Marwedel 2011]. What makes an ICFG different from a regular control flow graph is that it deals with control flow of an entire program, rather than a single function. A function call node is connected to the entry node of the function that is invoked and after reaching the function exit node, the ICFG returns to the node following the original function call node.

4. RESULTS

XML is the default form of exchanging information among different components of SSQSA. It is generated by performing a recursive walk through eCFG. The XML that is generated from eCFG slightly differs from the one generated as a result of WCET estimation. The source code, its control flow graph and resulting XML containing WCET-related data are shown on an example that performs
“divide and conquer” problem solving approach: a problem is divided into simpler sub-problems until a certain threshold is reached and the sub-problems are simple enough to be solved. On the left side of Figure 1, the Divide and Conquer source code can be seen. The graph on the right side represents nodes of the generated Control Flow Graph. This figure has been made in the way that it is simple to compare a line of code on the left hand side and, at the same level, the produced nodes of the Control Flow Graph for that specific line of code. The structure of this Control Flow Graph is actually the XML file given below.

In this XML file, lines with more complex structures, like BRANCH_STATEMENT and LOOP_STATEMENT, have been bolded in order to emphasize the way the inner nodes affect these statements’ worst case values.

In our example, the goal of the program is to set the value of a variable to zero, but only if its value is smaller than a certain threshold. If the value is greater than the threshold, the program will “divide” the variable by performing subtraction by one. The following eCFG shows only the most important, divideAndConquer method.

```xml
<?xml version='1.0' encoding='UTF-8' standalone='no'?>
<CFG WCET='359'>
  <COMPILATION_UNIT average='1.0' best='1' children='2' id='1' info='/' worst='1'/>
  <FUNCTION_DECL average='1.0' best='1' children='3' id='2' info='divideAndConquer' worst='1'/>
  <BLOCK_SCOPE average='0.0' best='0' children='4' id='3' info=''/>
  <VAR_DECL average='1.0' best='1' children='5' id='4' info='int, n, 100'/>
  <BRANCH_STATEMENT average='178.0' best='1' children='6,21' id='5' info='n,GREATER,13' worst='355'/>
  <BRANCH average='355.0' best='355' children='7' id='6' info='/' worst='355'/>
  <CONDITION average='1.0' best='1' children='8' id='7' info='n,GREATER,13' worst='1'/>
  <BLOCK_SCOPE average='0.0' best='0' children='9' id='8' info=''/>
  <LOOP_STATEMENT average='349.0' best='10' children='10' id='9' info='n,GREATER,13' worst='349'/>
  <CONDITION average='1.0' best='1' children='11' id='10' info='n,GREATER,13' worst='1'/>
  <BLOCK_SCOPE average='0.0' best='0' children='12' id='11' info=''/>
  <FUNCTION_CALL average='1.0' best='1' children='13' id='12' info='System.out.println' worst='1'/>
  <ASSIGNMENT_STATEMENT average='1.0' best='1' children='14' id='13' info='n, FUNCTION_CALL' worst='1'/>
</CFG>
```

Figure 1 – Source code and its Control Flow Graph representation
The developed prototype is able to give correct WCET estimations only for a certain subset of possible examples, which is explained further in the next section. Working on the prototype however was hard, because of limited amount of information which was provided to the tool because of the conversion of source code to eCST. It is clear that WCET analysis would be much more precise when the estimations are performed on machine code, like some of the already mentioned projects. Nevertheless, it is possible to analyze and estimate WCET even on the source code level which can be of great benefit.

5. LIMITATIONS

Although some estimations are successfully performed and some progress was made, this project is only a prototype that does not cover all the given test examples. For now, it only works on simple pieces of code. There are some major points for improving in continuing the research on this project:

- **Condition evaluation in WCET analyzer**
  Currently, while performing WCET estimations, only simple conditions are successfully evaluated, such as \( i < 5 \) or \( i < j \). There are certain complications regarding the complex conditions, for example \( i < j + k \) || \( i < \text{func}(a, b) \).

- **Evaluation of complex expressions when initializing or assigning value to a variable**
  This prototype currently works only with a simple statements when generating eCFG from assignment statement or declaration of variables, such as int \( i = 5 \). The conversion works also for an assignment statement of this kind: int \( j = i \), but not for more complicated, such as int \( k = j + i \). This assignment statement cannot be evaluated because evaluation of arithmetic statements is not implemented yet.

- **Determination of function call target should be more precise**
  As already mentioned, currently only the number of function parameters is taken into consideration instead of checking their types, names etc. This should be improved by involving the parameter types into deducing the paths of the control-flow graph more precisely which can be done by reuse of Static Call Graph generation implemented in eGDNGenerator component of SSQSA [Rakić 2015].

6. CONCLUSION

The undertaken work is only a path towards a language independent WCET estimation. However, it is only the first phase towards its more precise estimation. The second is introducing the platform variable to the estimation, which means domain specific universal nodes could also become a part of the eCST structure. Upon success in two mentioned phases, the result validation is to be done by making comparison to the result generated by the SWEET tool. Also, an interesting proposal is to see
if a model similar to Statecharts (enriched Statechart) could be included in SSQSA framework and used for WCET analysis. The Statechart implementation has already begun but it is in the early stage of development.

The motivation behind this work is involving the static timing analysis in SSQSA framework. Upon finishing this first phase of research and implementation, it is clear that working further in this direction could lead us to some meaningful and accurate results, in the sense of WCET estimation working successfully on complex programs. The problems that were met are mostly related to solving some implementation issues explained in section Limitations in more detail. Upon their resolving, a highly functional language independent WCET estimation on source code level could be performed. Therefore, the future work towards introducing the platform variable could be undertaken.

REFERENCES
N. Sukur, N. Milošević, S. Pešić, J. Kolek, G. Rakić and Z. Budimac
Towards the Code Clone Analysis in Heterogeneous Software Products

TIJANA VISLAVSKI, ZORAN BUDIMAC AND GORDANA RAKIĆ, University of Novi Sad

Code clones are parts of source code that were usually created by copy-paste activities, with some minor changes in terms of added and deleted lines, changes in variable names, types used etc. or no changes at all. Clones in code decrease overall quality of software product, since they directly decrease maintainability, increase fault-proneness and make changes harder. Numerous researches deal with clone analysis, propose categorizations and solutions, and many tools have been developed for source code clone detection. However, there are still open questions primarily regarding what are precise characteristics of code fragments that should be considered as clones. Furthermore, tools are primarily focused on clone detection for a specific language, or set of languages. In this paper, we propose a language-independent code clone analysis, introduced as part of SSQSA (Set of Software Quality Static Analyzers) platform, aimed to enable consistent static analysis of heterogeneous software products. We describe the first prototype of the clone detection tool and show that it successfully detects same algorithms implemented in different programming languages as clones, and thus brings us a step closer to the overall goals.

Categories and Subject Descriptors: D.2.7 - [Software engineering - Distribution, Maintenance, and Enhancement]: Restructuring, reverse engineering, and reengineering

Keywords: Software Quality, Software Maintainability, Static Analysis, Code Clone Detection

1. INTRODUCTION

Copy-paste activity is a common developer practice in everyday programming. However, this practice introduces code clones, parts of identical, or near-identical code fragments. It is estimated that between 5% and 23% of large-scale projects represents duplicated code [Roy et al. 2009] [Pulkkinen et al. 2015]. Such code is harder to maintain, increases potential errors and “bugs” and decreases overall quality of software [Dang and Wani 2015] [Sheneamer and Kalita 2016] [Roy et al. 2009] [Pulkkinen et al. 2015]. If original code has some error, by copy-paste activity this error is scattered on several places. Consequently, when this error is resolved later on, developer must pay attention to change the code in all of these places. Similarly, when some change or new functionality should be introduced in a part of code that is repeated in multiple places across the project, the same task is being repeated multiple times. Thus, we can conclude that code clones not only make the source code more complicated to maintain, but also increase the cost of maintenance.

“Identical or near identical source codes” [Sudhamani and Lalitha 2014], “segments of code that are similar according to some definition of similarity” [Rattan et al. 2013] and various other definitions across literature lack precision when defining code clones. Different authors define similarity in different ways and clones are described on different levels of granularity (ranging from sequences of source code to architectural clones). Common for all explanations is that clones come from copy-paste activity, with minor (or no) modifications. The point at which degree of modification becomes too big to consider two parts of code as clones is not clearly determined. Still, a generally adopted classification of code clones across literature exists [Dang and Wani 2015] [Sheneamer and...
Clones are classified in four groups, as follows:

- **Type-1**: two code fragments are type-1 clones if they are identical, with exclusion of whitespace, comments and layout of code.
- **Type-2**: two code fragments are type-2 clones if they are identical, with exclusion of identifiers, types, whitespace, comments and layout of code.
- **Type-3**: two code fragments are type-3 clones if they are identical, with exclusion of some lines added, deleted or altered, identifiers, types, whitespace, comments and layout of code.
- **Type-4**: two code fragments are type-4 clones if they have the same behavior, but are syntactically different.

In their work, [Roy et al. 2009] introduced an example which illustrates each clone type, based on a simple example - function that calculates a sum and a product (Figure 1).

---

**Fig. 1. Editing scenarios for each clone type [Roy et al. 2009]**
When dealing with code clone detection, several possible approaches have been proposed [Sheneamer and Kalita 2016] [Roy et al. 2009] [Rattan et al. 2013]:

- **Textual** - compare two source code fragments based on their text, line by line, using none or little code transformations (such as removal of whitespace, comments etc.). These methods can be language-independent, but they mostly deal with Type-1 clones [Sheneamer and Kalita 2016] [Roy et al. 2009].
- **Lexical** - apply tokenization of source code, i.e. they transform source code in sequences of tokens. These approaches are generally more space and time consuming in comparison with textual [Sheneamer and Kalita 2016], but are more robust regarding some minor code changes, which textual approaches can be very sensitive to [Roy et al. 2009].
- **Syntactical** - either parse the source code into abstract syntax trees and then implement algorithms that detect matches in subtrees (tree-based), or they calculate several different metrics on parts of source code and compare results of these metrics. Metrics used are number of declaration statements, control statements, return statements, function calls etc. [Sheneamer and Kalita 2016] [Sudhamani and Lalitha 2014]
- **Semantic** - divided to graph-based and hybrid approaches. They are focused on detecting code parts that perform similar computation even when they are syntactically different. Graph-based approaches create PDGs (Program Dependency Graphs) that are then used to analyze data and control flow of different code parts. Hybrid approaches combine several different methods in order to overcome flaws that specific methods encounter [Sheneamer and Kalita 2016].

In the next chapter, we present some related work in this area. Chapter 3 describes language-independent source code representation in form of eCST (enriched Concrete Syntax Tree) and our algorithm for clone detection on that representation. Chapters 4 and 5 contain results and conclusions of our research, respectively. Finally, we propose some ideas for future work in Chapter 6.

2. RELATED WORK

Detailed description of various available tools has been given in [Dang and Wani 2015], [Sheneamer and Kalita 2016], [Roy et al. 2009], [Oliviera et al. 2015]. We will mention two that use similar approaches as ours.

In [Sudhamani and Lalitha 2014] and [Sudhamani and Lalitha 2015] a tool is presented that identifies clones by examining structure of control statements in code. They introduce a distance matrix which is populated with number of different control statements in two code fragments and also takes into account nested control statements. Similarity function is then used to calculate a similarity between these fragments, based on values from the matrix. Our tool also uses some kind of a distance matrix and then calculates similarity based on a similarity function. However, instead of taking into account only control statements, we compare by wider scope of language elements, including control statements (which are represented by LOOP_STATEMENT and BRANCH_STATEMENT universal nodes).

[Baxter et al. 1998] presents a tool for clone detection based on tree matching. Code is parsed into ASTs (Abstract Syntax Trees) and then subtree matching algorithms are used to detect identical or near identical subtrees. This is similar with our intermediary representation, but we abstract over the concrete language constructs and make the trees language-independent. In order to deal with scalability, they use a hashing function, so subtrees are first hashed into buckets and then only subtrees from the same bucket are examined. This is something we have to consider in future versions in order to achieve more scalable solution.

The most important thing that makes our tool different is that we aim at clone detection that is independent of language, but in a way that it detects code clones across compilation units written in
different programming languages. Great majority of today’s large-scale software products are written in more than one programming language. Different components/modules are developed in different technologies, and thus there is a need for a tool which would overcome technology differences when detecting clones. There are tools that are language independent, especially some text-based tools, but they mostly deal with one language at a time. [Sheneamer and Kalita 2016] [Roy et al. 2009] Part of SSQSA called eCSTGenerator enables us to collect many files written possibly in different languages at the same time, and transform all of them into their respective eCSTs. Once eCSTs are generated, they can all be analyzed together, regardless of their original language.

3. DESCRIPTION

Our clone detection tool has been created as part of SSQSA (Set of Software Quality Static Analyzers) framework [Rakić 2015]. This project introduced its intermediary source code representation called eCST (enriched Concrete Syntax Tree) that consists of two types of nodes. Universal nodes represent language constructs on higher, more abstract level. Example of such nodes are COMPILATION_UNIT, FUNCTION_DECL, STATEMENT, etc. These nodes are language-independent and are internal nodes of the eCST. Analysis of universal nodes in eCSTs, generated from source codes in different languages, enables reasoning about the semantics of these source codes, even though they are written in different languages that can even use different programming paradigms. Leaves of eCSTs contain nodes that represent concrete syntactical constructs from the source code and are language-dependent. Examples of such nodes are “;”, “for”, “:=” etc.

In Figure 2 is presented a part of eCST generated for insertion sort algorithm. It is a subset of nodes generated for the whole compilation unit, with presented only characteristic universal nodes relevant for the analysis. The rest of the tree, especially leaf nodes are omitted in order to abstract over the concrete language of implementation. The highest level contains FUNCTION_DECL universal node which is used as a parent for all lower level nodes that capture the information about a sorting function. As we go down the tree, universal nodes begin to represent more specific language constructs, of finer granularity. For example, LOOP_STATEMENT is used to for capturing the information about any kind of loop and contains CONDITION node and sequence of nodes that represent statements, as its children nodes. These statements can be again some loops, branches, assignment statements, function calls etc. Mentioned insertion sort algorithm was developed in four different languages: Java, Modula-2, JavaScript and PHP. Figure 3 contains respective source codes.

Our clone detection algorithm implementation is based on working with universal nodes, and thus enables clone detection in functions that were written in different programming languages. We implemented a dynamic algorithm that compares two eCST subtrees representing functions by comparing their respective nodes. eCSTs are first scanned in order to extract only subtrees which represent function bodies. Each function body is then considered to be separate tree. These trees are transformed into sequences of nodes, using BFS (Breadth-First Search) strategy. Each two sequences are then compared by creating a matrix $m \times n$ ($m$ and $n$ being number of nodes in two sequences). Every pair of nodes $(i, j), 0 \leq i < m, 0 \leq j < n$ is compared and the result is inserted into the matrix following dynamic programming principle:

$$
matrix[i][j] = \text{compare}(\text{sequence1}[i], \text{sequence2}[j]) + \min(\text{matrix}[i-1][j-1], \text{matrix}[i][j-1], \text{matrix}[i-1][j])
$$

When matrix is full, algorithm searches for a “best match” between two trees, i.e. it searches a sequence of entries in the matrix going from $\text{matrix}[m-1][n-1]$ to $\text{matrix}[0][0]$ in which number of matched nodes is the greatest. Finally, similarity of two functions is measured with the formula
(B / A) * 100%. B represents the number of matches in the best match and A is the total number of nodes on the path from matrix[m−1][n−1] to matrix[0][0] on which the best match was found.

4. RESULTS

Algorithm was tested on three different sets of problems:
- scenario-based clones proposed in [Roy et al. 2009] and presented in Figure 1 in order to test our clone detection algorithm on one language and compare it with results of other tools
- implementation of few algorithms in Java, JavaScript, Modula-2 and PHP to check behavior of our tool on several different languages to test language-independence
- coding competition samples proposed in [Wagner et al. 2016] in order to test limitations of our algorithm and get some indications in which direction should we continue.

4.1 Algorithm correctness check

First we implemented scenario-based clones [Roy et al. 2009] in one language - Java. Since the nature of eCST is such that it ignores layout and whitespace (comments can be detected or ignored, it is configurable), Type-1 clones are easily detected with a 100% matching. All scenarios for Type-2 had a 100% matching, except of scenario S2d which had a 93% matching. This is because additional subtrees are generated for statements sum = sum + (i × i) and prod = prod × (i × i), compared to sum = sum + i and prod = prod * i (for parts of statements in parenthesis) that do not have matches in original. And since total number of nodes for these functions are small (40 for original), few extra nodes make a big difference in overall calculation of similarity. Results for Type-3 ranged from 81% to 100%, and for Type-4 where all above 90%.
As we can see from the table, the biggest problem for our tool represented clones $S3c$ and $S3e$ which have whole additional branches inserted. Reason for this is in the structure of eCSTs created accordingly, similarly to scenario $S2d$ explained above. New subtrees are generated for these branches that increase number of nodes in these two scenarios substantially (substantially in comparison with total amount of nodes that are created for the original function).

Since the paper which proposed clone scenarios [Roy et al. 2009] was comparing different tools based on their published characteristics, at this point we do not have empirical data to compare our detection of scenario-based clones with different tools. In [Roy et al. 2009] tools were categorized on the 7-level scale ranging from Very well to Cannot in terms of whether specific tool can detect certain scenario or not. Regarding the category of tree-based clone detection tools, to which our tool belongs to, authors determined that none of the tools from this category would be able to detect Type-4 clones, except CloneDr [Baxter et al. 1998] which would probably be able to identify scenario $S4a$. Following their scale range, we would place our first prototype in Medium level (3rd level), since we still do not have enough information about potential false positives, and we must for now presume that our tool may return substantial number of those.
Towards the Code Clone Analysis in Heterogeneous Software Products

Table I. Results for scenario-based clone example proposed by [Roy et al. 2009]

<table>
<thead>
<tr>
<th></th>
<th>Type-2 clones</th>
<th>Type-3 clones</th>
<th>Type-4 clones</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>S2a</td>
<td>S2b</td>
<td>S2c</td>
</tr>
<tr>
<td></td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
</tbody>
</table>

4.2 Cross-language consistency

In this phase two implementations of sorting algorithms, insertion sort and selection sort, as well as recursive function that calculates Fibonacci's numbers were considered. Implementations have been done in four different programming languages: Java, JavaScript, PHP and Modula-2. A part of eCST generated for insertion sort algorithm and respective source codes have already been given in Figures 2 and 3. These are semantically the same algorithms, only differences come from syntactic rules of their respective languages. Thus, slightly different trees are going to be generated. For example, Modula-2 function-level local variable declarations are located before the function block scope, so no VAR_DECL nodes are going to be presented in the BLOCK_SCOPE of Modula-2 function, in contrast to other languages.

4.3 Limitations

We used a sample of a dataset proposed in [Wagner et al. 2016], which represents various solutions to problems that were being solved at a coding competition. This set of problems was quite interesting since all implementations have a common goal - they solve the same problem. However, calculated similarities were quite small (not going over 30%), despite being written in the same language (Java). This corresponds to results published by [Wagner et al. 2016] where another class of clones is discussed - clones that were not created by copy-paste activity, but independently. These clones are called functionally similar clones (FSC). As in case of other tools [Wagner et al. 2016], ours was not able to identify this type of clones, and it is still an open issue to cope with.

5. CONCLUSION

With our clone detection algorithm we showed that even inter-language clones could be detected when operating on the level of universal nodes. Since most programming languages share the same concepts and similar language constructs, same algorithm implemented in two or more languages could produce the same eCST trees and thus their shared structure can be detected, which we showed on the few examples in Java, Modula-2, PHP and JavaScript. We also showed that our tool successfully identifies different copy-paste scenarios as highly similar code fragments. However, this is only the first prototype and has certain limitations and weaknesses. Our similarity calculation is very sensitive in respect of length of code. For example, when a substantial amount of code is added in between two parts of code that were result of a copy-paste activity, their similarity will decrease, perhaps even below some threshold we set up as a signal for clone pair, depending on the amount of code added.

6. FUTURE WORK

There is a lot of space for improvement in our tool, regarding current approaches and taking new ones. Our analysis is currently only dealing with function-level granularity. This should be extended in both ways - narrowing and widening it. Our similarity calculation is particularly sensitive to adding new parts of code or removing some parts (Type-3 clones), because it takes into account number of nodes which can change substantially with these changes. Our calculation should be
normalized in order not to fluctuate so drastically with these insertions and deletions. Also, since the algorithm compares all units of interest (currently function bodies) with each other, this is not a solution that would scale very good on large projects. A work-around should be carried out in order to deal with this problem, some grouping of similar units, either by using some sort of hash function, a metric value etc.

Regarding future directions, we could change our implementation to work not with eCSTs, but with eCFGs (enriched Control Flow Graphs) [Rakić 2015], which would allow us to concentrate more on semantics while detecting clone pairs and searching for architectural clones using eGDNs (enriched General Dependency Networks) [8], both representations already being part of SSQSA.

Output is currently only text-based, with calculated similarities for each two functions in some given scope, and optionally whole generated matrices. This kind of output could of course be improved, by introducing some graphical user interface which would, for example, color-map clone pairs in the original code.

REFERENCES

Monitoring an OOP Course Through Assignments in a Distributed Pair Programming System

STEELIOS XINOALOS, MAYA SATRATZEMI, DESPINA TSOMPANOUDI AND ALEXANDER CHATZIGEORGIOU, University of Macedonia

Distributed Pair Programming (DPP) is widely known to promote collaboration and knowledge sharing among novice programmers, while it engages them in carrying out programming assignments. Moreover, DPP is a means of experiencing agile software development techniques that are considered important in the software market. In this paper, we share some experiences on using the DPP system of SCEPPSys for carrying out assignments in an undergraduate Object Oriented Programming (OOP) course. Specifically, we focus on the information recorded during problem solving and the statistics reported by the system and how this information can be utilized for monitoring both the course regarding the fulfillment of its goals and the programming habits and progress of students. Some proposals are made towards extending the possibilities of SCEPPSys for generating automatically more sophisticated reports that would support instructors in more successfully monitoring a course and students. The ultimate goal of such an enhanced monitoring is to improve students’ software quality.

Categories and Subject Descriptors: D.2.3 [Software Engineering]: Coding Tools and Techniques – Object-Oriented Programming; D.2.8 [Software Engineering]: Metrics; K.3.2 [Computers and Education]: Computer and Information Science Education – Computer science education

General Terms: Education, Measurement

Additional Key Words and Phrases: Distributed Pair Programming, scripted collaboration, monitoring, metrics

1. INTRODUCTION

Distributed Pair Programming (DPP) systems allow two programmers to collaborate remotely in order to apply the Pair Programming (PP) [Cockburn and Williams 2000] technique from separate locations. The model of PP originated from the software industry as a part of Extreme Programming (XP) [Williams et al. 2000]. It involves two programmers working on the same workstation and sharing one computer in order to develop software. In the context of pair programming, the team members adopt two specific roles: one programmer acts as the “driver” and the other one as the “navigator” (also called “observer”). The driver has possession of keyboard and mouse and types the programming code, while the navigator reviews the inserted code and gives guidelines to the driver. Driver and navigator are in constant collaboration in order to design and develop the program code in common, while they should frequently alternate roles. PP is based on close collaboration, continuous knowledge transfer, negotiation and sharing of programming skills. Programmers not only receive the advantages of group work, they also produce high quality software in a shorter time [Sanjay and Vanshi 2010, Zacharis 2009]. The only difference between DPP and PP is DPP’s allowing geographically-distributed teams to collaborate and share program code. However, such collaboration is only feasible if an underlying infrastructure enables for all necessary interactions.

A considerable number of solutions used to implement DPP were built as plugins for the Eclipse IDE. These plugins support DPP for the Java programming language, within a very popular development environment. A detailed review of all available DPP solutions reveals that most tools lack effective instructional or student support [Tsompanoudi and Satratzemi 2011]. Although they cover the basic requirements of DPP, they cannot address common problems of PP, such as unequal contributions from each member of the student pair.

Authors’ addresses: S. Xinogalos, M. Satratzemi, D. Tsompanoudi, A. Chatzigeorgiou, Department of Applied Informatics, School of Information Sciences, University of Macedonia, Egnatia 156, 54636 Thessaloniki, Greece, email: {stelios, maya, despinats, achat}@uom.gr

Copyright © by the paper’s authors. Copying permitted only for private and academic purposes.
For this purpose SCEPPSys (Scripted Collaboration in an Educational Pair Programming System) was developed using an existing Eclipse plugin for DPP. Compared to other plugins like Sangam [Ho et al. 2004] and XPairtise [Schümmer and Lukosch 2009], SCEPPSys saves and analyzes users’ interactions, and helps educators in organizing and monitoring DPP classes. To the best of our knowledge only RIPPLE [Boyer et al. 2008] provides logging capability of users’ interactions, allowing researchers to reconstruct user sessions for further study. However, the logged information is provided only as raw data and further processing is required in order to extract useful information.

In this paper, we share some experiences on using the DPP system of SCEPPSys for carrying out assignments in an undergraduate Object Oriented Programming (OOP) course based on Java. Specifically, we focus on the information recorded during problem solving and the statistics reported by the system and how this information can be utilized for monitoring both the course regarding the fulfillment of its goals and the programming habits and progress of students. Some proposals are made towards extending the possibilities of SCEPPSys for reporting automatically more sophisticated reports that would support instructors in more successfully monitoring a course and students.

2. SCEPPSYS

SCEPPSys is based on a typical client-server architecture and consists of: a server for dispatching messages between the clients; a database for storing user’s accounts, information about courses, groups and assignments, shared projects and statistics; a web-based authoring tool used by instructors for scripting DPP [Tsompanoudi and Satratzemi 2014]; and an Eclipse plugin installed by students. In the next sections the process that an instructor applies for setting up a course, as well as a typical DPP session carried out by students are briefly described. Interested readers can find further information in a detailed description of SCEPPSys [Tsompanoudi et al. 2015].

2.1 Setting up a Course

The instructor can easily set up a course using the web-based authoring tool of SCEPPSys. First of all the list of learning goals (e.g. constructor definition, object construction, inheritance) that will be used for characterizing the various tasks assigned to students must be defined. Then the assignments can be defined and scheduled in the form of collaboration scripts. A collaboration script involves the definition of: (1) participants - students enrolled to the course; (2) groups or pairs that can be formed randomly, with comparable skill or contribution levels, or freely; (3) assignments that comprise of problem solving tasks or steps characterized by one of the learning goals (not visible to students) and having an accompanying hint that can be optionally consulted by students; (4) task distribution policies - rotating role switching of driver/observer, balanced knowledge switching aiming at achieving symmetry in skill acquisition (learning goals) or free switching. The system monitors the problem solving approach adopted by students and reports various statistics described in the next section.

2.2 Carrying out a Typical DPP Session

Students use an Eclipse plugin for applying DPP. A DPP session starts when group members meet online and request a pair programming session. Then a shared project is automatically generated inside the workspace of both students and the programming tasks are displayed in a separate area. Students solve the tasks by adopting the roles of the driver and navigator and switch roles according to the task distribution policy. During the session a text-based chat provides a means of communication and coordination between the team members. To motivate students, metrics like driving time and individual participation rates are displayed in the workspace and students may retrieve useful hints for each step during the problem solving process. Students may submit the assignment on session close or continue the DPP session at another time.
3. STATISTICS REPORTED BY THE SYSTEM

SCEPPSys records a variety of information during the problem solving process and calculates statistics for each student per project. The statistics reported are presented in Table I. Moreover, the mean value of the aforementioned elements is calculated along with the number of projects submitted per assignment for each one of the utilized distribution policies (none/free, rotating roles, balanced).

### Table I. Statistics reported by SCEPPSys.

<table>
<thead>
<tr>
<th>Status (project submitted, not submitted, not found)</th>
<th>Contribution of first student (in %)</th>
<th>Contribution of second student (in %)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task Distribution policy (roles rotating, balanced knowledge, free)</td>
<td>Contribution of first student (in %)</td>
<td>Contribution of second student (in %)</td>
</tr>
<tr>
<td>Total time spent to solve a project (min)</td>
<td>Number of steps solved according to role distribution policy</td>
<td></td>
</tr>
<tr>
<td>Driving time spent to solve a project (min)</td>
<td>Driving time of first student</td>
<td></td>
</tr>
<tr>
<td>Driving / Total time ratio</td>
<td>Driving time of second student</td>
<td></td>
</tr>
<tr>
<td>Number of sync runs</td>
<td>Non driving time of first student</td>
<td></td>
</tr>
<tr>
<td>Number of role switches</td>
<td>Non driving time of second student</td>
<td></td>
</tr>
<tr>
<td>Number of retrieved hints</td>
<td>Messages sent by first student</td>
<td></td>
</tr>
<tr>
<td>Contribution of first student (number of characters)</td>
<td>Messages sent by second student</td>
<td></td>
</tr>
<tr>
<td>Contribution of second student (number of characters)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The statistics calculated and reported by SCEPPSys in combination with the projects’ grades can be used as metrics for:

- monitoring the fulfillment of the courses’ goals in general
- detecting difficulties with specific OOP concepts/constructs
- detecting students’ progress in programming
- detecting undesirable behaviors (e.g. plagiarism)
- detecting problems in collaboration between the members of a pair

In the following sections the indications provided by the most important of the aforementioned statistics are briefly analyzed.

3.1 Total and Driving Time

Both the total time for developing a project and the time for actually writing code (driving time) provide indications for the level of difficulty of an assignment, as well as the difficulties of students with the underlying OOP concepts. Moreover, the total and driving time of an assignment help realize students’ workload that must be in accordance with the ECTS of the course.

The difference of the total and driving time, or else the driving/total time ratio, helps instructors detect odd or even extreme behaviors. For example, a very small total time or a total time that is approximately equal to the driving time are strong indications of either “copying a solution” or “working offline” (most probably in isolation) and logging in the system just for copying and pasting either the plagiarized or offline solution correspondingly. Both behaviors are problematic and denote non-academic behavior, non-collaborative problem solving or cooperation problems. Instructors can draw safer conclusions if the minimum time necessary to collaboratively solve a specific assignment through the system is recorded. This can be ideally accomplished by having themselves or knowledgeable assistants solve the assignment, or else by estimating the necessary time based on their experience from lab sessions.

3.2 Number of Retrieved Hints

Every step of an assignment can have a corresponding hint, providing support to students. Depending on the difficulty of a specific step and whether it deals with a new OOP concept/construct the help provided can be more or less verbose. Moreover, it can contain small pieces of source code. The number of hints retrieved by students is another indication of the degree of difficulty or students’
confidence on their solution for an assignment. This means that the more difficult an assignment is, or alternatively the less confident students are for their solution, a bigger number of hints is retrieved.

3.3 Messages Sent During Problem Solving
One of the main advantages of DPP is the collaboration and exchange of knowledge between the members of a pair. The communication between the collaborators can be accomplished using the incorporated chat tool. The number of messages sent during code development provides indications for the degree of cooperation and communication between the members of a pair, as well as for the difficulty of an assignment. Of course, students can and do use other means for communication during problem solving, such as Skype and Facebook.

3.4 Number of Synchronized Program Executions
The number of synchronized program executions (sync runs) provides information regarding the problem solving strategy adopted from students. Instructors always emphasize the advantages of incremental development and testing, but unfortunately students tend to write large portions of a program before they actually test it. When implementing programs in OOP languages this delayed testing is even more prominent due to the fact that usually an important portion of a class or classes has to be implemented along with the main method in order to test the program. Students tend to complete the definition of a class, or even the classes needed, before they write the main method and test their solution. SCEPSSyS can contribute to changing students’ problem solving strategies through scripted collaboration. Specifically, students can be asked to incrementally implement and test a class by appropriate steps in the collaboration script and moreover they can be asked to check the results until then by running their project. However, it is clear that students can overlook the script’s suggestion and not test their project. So, examining the number of synchronized program executions can help monitor students’ problem solving strategies.

Moreover, the number of sync runs in combination with other statistics, such as the total and driving time, number of hints retrieved and messages sent, can indicate potential difficulties in achieving the goals of an assignment. For example, a large number of sync runs combined with an average (or less than average) total and driving time can indicate the application of the desirable incremental development and testing strategy. On the other hand, a large number of sync runs combined with a large amount of total and driving time (above average) can indicate difficulties in achieving the goals of the assignment.

4. CASE STUDY
SCEPPSys was used during the academic year 2015-16 in a 3rd semester compulsory course on OOP at the Department of Applied Informatics, University of Macedonia, Greece. Students were taught for 3 hours per week in the lab and had the chance to carry out programming assignments during the 13 weeks of the course. The assignments were optional and their grade was calculated only in the case that students passed the final exams. The previous years the assignments were carried out individually by the participating students as homework. This year students had the chance to select whether they would like to solve the assignments individually, or in pairs using the DPP system of SCEPPSys. Initially 48 pairs of students registered for the DPP assignments, while 47 pairs submitted at least one out of the six projects assigned. At the end of the course a questionnaire was delivered to the participating students in order to obtain their feedback on the DPP assignments.

The six projects covered the main learning units of the course and utilized small and manageable steps with corresponding hints. The only exception was the sixth project that included just five steps, one for each one of the necessary classes. In Table II the learning unit covered by each project is presented, along with the mean grade (in the scale of 0 to 10) and the mean values of the metrics briefly analyzed in the previous section.
### Table II. Statistics for the Projects

<table>
<thead>
<tr>
<th>PROJECT</th>
<th>LEARNING UNIT</th>
<th>NUMBER OF CLASSES (STEPS)</th>
<th>LOC (lines of Code)</th>
<th>NUMBER OF PROJECTS</th>
<th>GRADE (in scale 0..10)</th>
<th>TOTAL TIME (MIN)</th>
<th>DRIVING TIME (MIN)</th>
<th>DRIVING / TOTAL TIME RATIO</th>
<th>NUMBER OF SYNC RUNS</th>
<th>NUMBER OF RETRIEVED HINTS</th>
<th>MESSAGES SENT BY EACH GROUP</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>Class definition, main</td>
<td>2 (13)</td>
<td>90</td>
<td>45</td>
<td>9.25</td>
<td>190</td>
<td>36</td>
<td>25%</td>
<td>8</td>
<td>9 (69%)</td>
<td></td>
</tr>
<tr>
<td>#2</td>
<td>Class associations - relationship</td>
<td>3 (16)</td>
<td>120</td>
<td>46</td>
<td>8.72</td>
<td>231</td>
<td>55</td>
<td>28%</td>
<td>13</td>
<td>13 (81%)</td>
<td></td>
</tr>
<tr>
<td>#3</td>
<td>Object collections – ArrayList</td>
<td>3 (23)</td>
<td>160</td>
<td>39</td>
<td>9.15</td>
<td>262</td>
<td>63</td>
<td>30%</td>
<td>25</td>
<td>15 (65%)</td>
<td></td>
</tr>
<tr>
<td>#4</td>
<td>Inheritance &amp; polymorphism</td>
<td>4 (16)</td>
<td>114</td>
<td>35</td>
<td>9.21</td>
<td>127</td>
<td>40</td>
<td>35%</td>
<td>9</td>
<td>9 (56%)</td>
<td></td>
</tr>
<tr>
<td>#5</td>
<td>GUI, event handling (+inheritance)</td>
<td>6 (24)</td>
<td>135</td>
<td>28</td>
<td>9.36</td>
<td>243</td>
<td>50</td>
<td>25%</td>
<td>18</td>
<td>16 (67%)</td>
<td></td>
</tr>
<tr>
<td>#6</td>
<td>Binary files (+inheritance, ArrayList, Comparator)</td>
<td>5 (5)</td>
<td>210</td>
<td>25</td>
<td>8.76</td>
<td>174</td>
<td>34</td>
<td>21%</td>
<td>15</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

**Grades**

Students’ mean grades in all the projects were very good (at least 8.72). Even at the last project the mean grade was 8.76, with the difference that it was submitted by fewer pairs. As already mentioned, the last project was a special case, since it included one step for each class and no hints. Moreover, it included most of the OOP concepts/constructs taught in the course.

**Number of projects**

Students tend to give up the effort as the course reaches the end. As can be seen in Table II the number of projects submitted decreases importantly in the last two assignments (approximately the last three weeks of the course). It is quite certain that instructors teaching programming courses, and especially OOP, would agree that such courses are cognitively demanding and the cognitive overload of students leads a certain percentage of them in dropping out of their responsibilities at some point, such as attending lectures and carrying out assignments. Of course, we must remind that in the specific case, assignments were not obligatory. Moreover, we must mention that 41% of the students that participated in the DPP assignments had not passed the first semester “Procedural Programming” course based on C. It is clear that understanding the reasons for dropping out needs a much deeper analysis of the available data in order to draw solid conclusions.

**Duration**

Implementing programs is widely known that is a time consuming activity, especially for novices. Based on the data collected by the system, it comes out that students working in pairs spent approximately two to four hours for writing the code for an assignment. However, nearly one fourth to one third of this time was spent on actually writing code. The only exception was the last and most demanding - in terms of the concepts covered, LOC, lack of detailed steps (each step corresponded to a whole class) and hints – project, where just 21% of the total time was spent on writing code. It is reasonable to assume that most of the time was spent on studying the corresponding material, reading hints, communicating with the collaborator and of course thinking about the solution.
Number of sync runs
Incremental development and testing, as already mentioned, is an important aspect that should be reinforced among students. However, especially in OOP, this is not always easy. Students tend to implement the classes needed and then write the main method for testing their code. In scripted collaboration the instructor has the chance to reinforce the idea of incremental development and testing by asking students partially implement a class and create objects for testing each method from the very first steps. Moreover, students can be reminded in the textual description of the steps to run their project. Of course, students can just ignore such advice. From the data recorded by the system it turns out that students do run the programs, but it cannot be told in certainty that they do it during problem solving for testing their program, or because they have to make continuous corrections due to raised exceptions and/or logical errors that lead to wrong output. However, at least 8 sync runs were recorded in average for each project, which is an indication of incremental development and testing. If the time of each sync run and its (un)successful completion was recorded we could draw safer conclusions regarding students’ problem solving strategies.

Messages sent by each group
Although several students use alternative means for communication, and especially Skype and Facebook, the average number of messages sent by the members of the pairs was unexpectedly high. Specifically, 86 to 153 messages were sent in average for each project. This is definitely a strong indication of collaborative work and exchange of perceptions and knowledge.

Coverage of syllabus (assignments)
An important feature of SCEPPSys that helps the instructor to monitor the coverage of the intended OOP concepts/constructs through the assignments, is reporting the frequency that each learning goal has been addressed in the context of the assignments so far. This feature gives the chance to adjust the assignments and the learning goals they aim at. Ideally, this must be done in combination with the recorded difficulty for each learning goal as the course progresses. Since the projects are graded on-line in a step by step manner and each step is characterized by its main learning goal, the average grade per learning goal can be calculated and used for detecting students’ difficulties and taking the appropriate actions both in lectures and assignments.

In the on-line questionnaire that was answered voluntarily by 58 out of the 94 students that participated in the DPP assignments, the majority agreed or completely agreed that the quality of the assignments was good (86.3%) and that the assignments covered in a high degree the content of the course (84.5%).

Difficulty of the assignments
Based on the assumptions made in the previous section about the various statistics recorded and using the data reported in Table II, some conclusions regarding the difficulty of the assignments can be drawn.

Considering the number of projects submitted in combination with the grades of the assignments it is obvious that the last assignment (#6) was the most difficult one. This was anticipated since it combines most of the OOP concepts taught: inheritance, ArrayList, event handling and for the first time concepts from the Collection Framework (Comparator) and binary files. This makes it clear that students should be given more time (usually it is ten days) for this last assignment.

Based on the reduced number of projects submitted (28) the next most difficult assignment was the one referring to GUI and event handling (#5), which also utilized inheritance. The number of submitted projects decreased gradually from 45/46 to 39, 35 and finally less than 30 in the last two projects. It seems that this is more or less the time point during the course at which the amount of required knowledge overwhelms a number of students. As a consequence, dropping out appears and instructors should pay special attention in finding ways to support and even more motivate students to keep on the effort. However, the performance of the pairs that submitted the last two projects was still very good.
Although students’ performance in the assignment regarding object collections and more specifically the ArrayList collection was very good (average grade 9.15), there are strong indications that it was a demanding assignment (#3) for students, maybe difficult for some of them. Specifically, this assignment demanded more than 4 hours of work in average for each pair, and a big number (153) of messages sent through the incorporated chat tool in order to collaboratively reach a solution. Moreover, 65% of the available hints were retrieved by the students and 25 sync runs of their projects were made. The number of sync runs indicates that various problems came up during problem solving that required correcting and re-running the project. Of course, we must stress that students finally managed to collaboratively reach a good solution, as well as the fact that difficulties with ArrayList collections have been recorded in previous studies (Xinogalos et al. 2008).

The conclusions drawn from the statistics calculated by the system are partially confirmed by students’ replies on the questionnaire presented in Table III.

Table III. Students replies regarding the difficulty of the assignments

<table>
<thead>
<tr>
<th>PROJECT</th>
<th>LEARNING UNIT</th>
<th>NOT SUBMITTED</th>
<th>EASY</th>
<th>OF LOW DIFFICULTY</th>
<th>OF MEDIUM DIFFICULTY</th>
<th>DIFFICULT</th>
<th>OF HIGH DIFFICITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>Class definition, main</td>
<td>6.90%</td>
<td>53.40%</td>
<td>29.30%</td>
<td>6.90%</td>
<td>3.40%</td>
<td>0%</td>
</tr>
<tr>
<td>#2</td>
<td>Class associations – relationship</td>
<td>1.70%</td>
<td>36.20%</td>
<td>51.70%</td>
<td>8.60%</td>
<td>1.70%</td>
<td>0%</td>
</tr>
<tr>
<td>#3</td>
<td>Object collections – ArrayList</td>
<td>1.70%</td>
<td>17.20%</td>
<td>39.70%</td>
<td>37.90%</td>
<td>3.40%</td>
<td>0%</td>
</tr>
<tr>
<td>#4</td>
<td>Inheritance &amp; polymorphism</td>
<td>5.20%</td>
<td>8.60%</td>
<td>20.70%</td>
<td>50%</td>
<td>13.80%</td>
<td>1.70%</td>
</tr>
<tr>
<td>#5</td>
<td>GUI, event handling (+inheritance)</td>
<td>15.50%</td>
<td>0%</td>
<td>10.30%</td>
<td>15.50%</td>
<td>43.10%</td>
<td>15.50%</td>
</tr>
<tr>
<td>#6</td>
<td>Binary files (+inheritance, ArrayList, Comparator)</td>
<td>17.20%</td>
<td>1.70%</td>
<td>5.20%</td>
<td>12.10%</td>
<td>32.00%</td>
<td>31%</td>
</tr>
</tbody>
</table>

Out of the 58 students that filled in the questionnaire:

- The last assignment (#6) that demanded the usage of inheritance, ArrayList collections, event handling, the Collection Framework (Comparator) and binary files was the most difficult for students. Specifically, 63% considered this assignment as difficult (32%) or of high difficulty (31%), while another 17.2% did not submit this assignment.
- The assignment #5 was the second more difficult assignment. It referred to GUI, event handling and also utilized inheritance. 58.6% of the students considered this assignment to be difficult (43.1%), or even of high difficulty (15.5%).
- The next more difficult assignment according to students was the one referring to inheritance and polymorphism (#4). Specifically, 50% of the students considered the assignment of medium difficulty and 15.5% as difficult or of high difficulty. The statistics recorded by the system show that this assignment demanded the less time, while the smallest percentage (56%) of hints was retrieved in comparison with the rest of the assignments. Probably, the assignment demanded off-line studying and preparation.
- Finally, the assignment on object collections and ArrayList (#3) was considered by 37.9% of the students to be of medium difficulty, while the statistics recorded indicated a higher degree of difficulty.

Despite the insights provided by students’ replies, we must mention that the questionnaire was filled in after the end of the course. If students were asked to rate the difficulty of each assignment just after finishing it (for example, by entering the specific information as a comment at the end of their project) and not recalling after several weeks their difficulty with assignments, the results of the specific question would be more trustworthy.
5. CONCLUSIONS

The benefits of DPP are numerous and have been recorded in the literature. The system of SCEPPSys utilized in this study has the advantage of recording various data and reporting statistics that could be used for monitoring an OOP course based on Java regarding the fulfillment of its goals and taking the appropriate actions in time. Such statistics can be used for detecting difficulties with specific concepts, the quality of collaboration between pairs and potential undesirable behaviors, as well as students' progress in programming. An important extension of SCEPPSys would be the ability to provide more advanced reports towards this direction that would automate the process of detecting:

- **potential difficulties with specific OOP concepts/constructs**: by calculating the average grade achieved by students for each learning goal in the context of each assignment and all the assignments collectively.
- **detecting students’ progress in programming**: by calculating and reporting important changes in the grades of his/her projects, as well as the contribution of a student in the projects.
- **detecting undesirable behaviors** (e.g. copying a solution, or working offline): by comparing the total and driving time for a project (if there is a small difference between them then probably a solution was just entered in the system and not collaboratively developed), and also with a minimum required time defined by the instructor.

It is obvious that in order to achieve this goal and provide really comprehensive reports that will save a great deal of time to instructors and give them even more valuable information both for monitoring the course and each student, the system has to be used in real situations. This study had exactly this goal, to start exploring the indications provided by the reported statistics, validating and using them for automatically creating reports that will be a useful tool in the hands of instructors. A step further would be to examine the possibility of incorporating to the system abilities for calculating simple educational software metrics with the aim of enhancing the quality of students' programs [Xinogalos and Ivanović 2013].

REFERENCES


