Workshop Proceedings

ACM/IEEE 16th International Conference on Model Driven Engineering Languages and Systems, Miami, Florida, USA
29 Sept 2013 through 4 Oct 2013

Experiences and Empirical Studies in Software Modelling (EESSMod 2013)

October 1

Michel Chaudron, Marcela Genero,
Silvia Abrahão, Lars Pareto (Eds)

POST-PROCEEDINGS

EESSMod 2013
Third International Workshop on Experiences and Empirical Studies in Software Modelling

Michel Chaudron[1], Marcela Genero[2], Silvia Abrahão[3], Lars Pareto[1]

[1]Chalmers – University of Gothenburg
Gothenburg, Sweden
chaudron, pareto@chalmers.se

[2]ALARCOS Research Group, University of Castilla-La Mancha
Paseo de la Universidad 4, 13071, Ciudad Real, Spain
Marcela.Genero@uclm.es

[3] Department of Information Systems and Computation – Universitat Politècnica de València
Camino de Vera, s/n, 46022, Valencia, Spain
sabrahao@dsic.upv.es

Preface

Most software development projects apply modelling in some stages of development and to various degrees in order to take advantage of the many and varied benefits of it. Modelling is, for example, applied for facilitating communication by hiding technical details, analysing a system from different perspectives, specifying its structure and behaviour in an understandable way, or even for enabling simulations and generating test cases in a mode-driven engineering approach. Thus, the evaluation of modelling techniques, languages and tools is needed in order to assess their advantages and disadvantages, to ensure their applicability to different contexts, their ease of use, and other issues such as required skills and costs; either isolated or in comparison with other methods.

The need to reflect and advance on empirical methods and techniques that help improving the adoption of software modelling in industry led us to organize two editions of the International Workshop on Experiences and Empirical Studies in Software Modelling that was held in Wellington (EESSMod 2011) and Innsbruck (EESSMod 2012) conjunction with the ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MoDELS). The third edition of the workshop will be held in Miami during MODELS 2013. The main purpose of the workshop is to bring together professionals and researchers interested in software modelling to discuss in which way software modelling techniques may be evaluated, share experiences of performing such evaluations and discuss ideas for further research in this area. The workshop accepted both experience reports of applying software modelling in industry and research papers that describe more rigorous empirical studies performed in industry or academia.

These proceedings collect the papers presented at the Workshop. All the submitted papers were peer-reviewed by three independent reviewers. The accepted papers (4 full papers and 4 short papers) discuss theoretical and practical issues related to experimentation in software modelling or the use of modelling techniques in industry.

We would like to thank the authors for submitting their papers to the Workshop. Also thanks to Prof. Lionel Briand from University of Luxembourg, who will give a very interesting keynote speech. We are also grateful to the members of the Program Committee for their efforts in the reviewing process, and to the MoDELS 2013 organizers for their support and assistance during the workshop organization. More details on the Workshop are available at http://users.dsic.upv.es/workshops/eessmod13.

Gothenburg, Ciudad Real, Valencia,
24 September 2013

Michel Chaudron
Marcela Genero
Silvia Abrahão
Lars Pareto

Program Committee

Bente Anda, University of Oslo, Norway
Teresa Baldassarre, Università degli Studi di Bari, Italy
Narasimha Bolloju, City University of Hong Kong, China
Danilo Caivano, Università degli Studi di Bari, Italy
Jeffrey Carver, University of Alabama, USA
Karl Cox, University of Brighton, UK
José Antonio Cruz-Lemus, University of Castilla-La Mancha, Spain
Holger Eichelberger, Universität Hildesheim, Germany
Felix Garcia, University of Castilla-La Mancha, Spain
Carmine Gravino, University of Salerno, Italy
Brian Henderson-Sellers, University of Technology, Sydney, Australia
Yvan Labiche, Carleton University, Canada
Jan Mendling, Humboldt-University Berlin, Germay
Parastoo Mohagheghi, NTNU, Norway
James Nelson, Southern Illinois University, USA
Jeffrey Parsons, Memorial University of Newfoundland, Canada
Keith Phalp, Bournemouth University, UK
Giuseppe Scanniello, Università degli Studi della Basilicata, Italy
Dag Sjøberg, University of Oslo, Norway
Keng Siau, Missouri University of Science and Technology, USA
Marco Torchiano, Politecnico di Torino, Italy
Barbara Weber, University of Innsbruck, Austria
Jon Whittle, Lancaster University, UK

Content

Preface i

Program committee iii

Research-Based Innovation in Model-Driven Engineering: Project Experience and Lessons Learned" (Keynote Speech) ... ……...……...……..……... 1
Lionel Briand

What are the used UML diagrams? A Preliminary Survey…… 3
Gianna Reggio, Maurizio Leotta, Filippo Ricca and Diego Clerissi

Model-based Simplified Functional Size Measurement - an Experimental Evaluation with COSMIC Function Points ……... 13
Vieri Del Bianco, Luigi Lavazza, Geng Liu, Sandro Morasca and Abedallah Zaid Abualkishik

UML usage in Open Source Software Development: A Field Study…….…..... 23
Hafeez Osman and Michel R. V. Chaudron

Exploring Costs and Benefits of Using UML on Maintenance: Preliminary Findings of a Case Study in a Large IT Department…….…..... 33
Ana M. Fernández-Sáez, Michel Chaudron and Marcela Genero

Towards Reconstructing the Architecture of Software Development Tools by Runtime Analysis……... 43
Ian Peake, Jan Olaf Blech and Lasith Fernando

Industrial Adoption of Automatically Extracted GUI Models for Testing …..... 49
Pekka Aho, Matias Suarez, Teemu Kanstren and Atif Memon

What do metamodels really look like? ……..... 55
James Williams, Athanasios Zolotas, Nicholas Matragkas, Louis Rose, Dimitris Kolovos, Richard Paige and Fiona Polack

Online Img2UML Repository: An Online Repository for UML Models……...61
Bilal Karasneh and Michel Chaudron

Research-Based Innovation in Model-Driven Engineering: Project Experience and Lessons Learned (Keynote Speech)

Lionel Briand

University of Luxembourg, Luxembourg
lionel.briand@uni.lu

Abstract. Engineering research needs to be informed by practice to be relevant and have impact, and industrial innovation relies on research to fill the gaps in knowledge and to pave the way for new tools, technologies, and services. With a focus on Model-Driven Engineering (MDE), this talk will report on my experience from a number of recent successful research projects conducted in various industry sectors. I will take a retrospective approach to examine the way I collaborated with the industry partners and elaborate on the decisions that I believe contributed to the effectiveness of the collaborations. I will then summarise the lessons learned from this experience and illustrate these lessons using examples from the projects.

What are the used UML diagrams? A Preliminary Survey

Gianna Reggio, Maurizio Leotta, Filippo Ricca, Diego Clerissi

DIBRIS – Università di Genova, Italy
gianna.reggio@unige.it, maurizio.leotta@unige.it,
filippo.ricca@unige.it, diego.clerissi@gmail.com

Abstract. UML is a large notation offering many diagrams and a large set of constructs for each of them covering any possible modelling need. As a result its specification is a huge book, its metamodel is large, and defining/understanding its static and dynamic semantics is difficult. These features have a negative impact on the perception of the UML and lead in some cases to replace it by ad-hoc lean and simple DSLs. Thus, the following question arises: which are the most/less used UML diagrams/constructs? We would like to answer to this question by means of a survey, trying to detect which parts of the UML are the most used. To see how much a diagram is used we preliminarily investigate books, courses/tutorials, and tools covering UML. The less used diagrams will be then analysed to understand the reasons of their low usage.

1 Introduction

UML is a truly large notation offering many different diagrams (14 in the last approved version [15]), and for each diagram it provides a large set of constructs covering any possible need of any modeller for any possible task. As a result, the UML specification is a huge book (732 pages), the UML metamodel is large and quite complex, and the definition and the understanding of its static and dynamic semantics is a truly difficult task, with also the consequence to make difficult to teach it both at the school/university level or in the industry [4]. Moreover, the large number of constructs and the consequent very large metamodel make complex and time consuming developing transformation of UML models and building tools for the UML. Clearly, these features of the UML have a negative impact on how the UML is perceived by the modellers hindering its adoption, and leading in some cases to replace the UML by ad-hoc lean and simple DSLs (Domain Specific Languages).

On the other hand, users naturally tend to consider and use only a portion of its diagrams/constructs, and forgetting about some other ones. On his blog I. Jacobson states "For 80% of all software only 20% of UML is needed" [6]. Furthermore, few people try to learn the UML by reading its specification [15], instead the large majority of the users relies on books and courses/tutorials or just start to use some tools for drawing UML diagrams that in general do not cover the whole UML. For this reason, in many cases, the UML users will never

become aware of the existence of many specific constructs (e.g., how many of you do know the existence of the "Parameter Set" for activities or has even used this construct?).

We would like to asses by means of a survey which parts of the UML (diagrams and constructs) are the most used in practice and which the less ones, using again the words of Jacobson trying to see if an "essential UML" [6] emerges. To discover how much a UML diagram/construct is used, we chose to preliminarily investigate: (1) the books about the UML, (2) the IT University courses covering also the UML, (3) the tutorials presenting the UML to the practitioners, and (4) the tools for producing UML models. Later, similarly to [2, 4], we will conduct a personal opinion survey [5] asking to UML users of different kinds (e.g., industrial practitioners and academics) which parts of the UML they know and which they have never used.

For a given UML diagram, we have proceeded as follows. We have investigated the books to discover if they were citing such diagram, and if they were giving an example of it. Similarly for the courses/tutorials, whereas for the tools we have tried to produce a model containing such diagram. Finally, we have computed descriptive statistics to present the results.

The results of this survey, of which this work constitutes the first step, and of the future personal opinion survey should be of help to many different categories of people:

- **teachers and instructors**: allowing to offer courses and/or tutorials concentrating only a smaller language made out of the most used UML diagrams/constructs;
- **tool builders/users**: obvious advantages since the tools covering the most used diagrams/constructs will be simpler to implement/use.
- **notation designers**: interested in discovering scarcely used constructs, and understanding for which reasons they have been added to the language. Moreover, other interesting questions arise: are the scarcely used constructs derived[1] or primitive? can the scarcely used constructs be applied only in specific cases? It will be interesting to investigate whether the metamodel (and subsequently the UML specification book) may be easily simplified to cover only the most used constructs.

In our opinion having to handle a notation with a large set of constructs where a portion of them are scarcely used, if not almost unknown, is problematic because it can cause a waste of effort and of resources by who want/must use it (e.g., in Italy, some contracts with the public administration must be accompanied by a UML model). From the trivial fact that to print the reference requires 700 sheets, to the fact that understanding the metamodel/preparing for the certification/deciding what to teach to the students/reading a UML book require a large number of hours, and we do not have to forget that also maintaining the official specification and any related item requires a large amount of effort due to its size. Also the OMG has recently recognised the need to simplify

[1] A derived construct may be replaced by a combination of other constructs.

the UML with the initiative "UML Simplification" [12] which will result in the next UML version (2.5), but in this case the simplification concerns only the way UML is defined without any impact about its constructs.

In this paper we present the results of the first step of our survey, i.e., focusing on the usage level of the 14 types of UML diagrams and covering books, courses/ tutorials, and tools. The remainder of the paper is organized as follows. In Sect. 2, we present related work literature regarding empirical study about the UML. In Sect. 3, we sketch the relevant aspects of the conducted empirical work such as: goals, research questions, followed process and analysis methodology. The results of the survey are presented in Sect. 4, while threats to validity are discussed in Sect. 5. Finally, Sect. 6 concludes the paper.

2 Related Work

The systematic literature review (SLR) by Mohagheghi et al. [8] about model-based software development states that "the UML is currently the most widely used modelling language". A similar result has also been obtained in [14] where a personal opinion survey with 155 Italian professionals has been conducted. Another personal opinion survey about UML (171 professionals in total), by Dobing and Parsons [2], points out another strong statement: "regular usage of UML components were lower than expected". The authors of [2] suggest that the difficulty of understanding many of the notations "support the argument that the UML may be too complex". The same claim, in more or less different forms, is present in several blogs, where several proposals of UML simplification are arising[2]. Maybe, the most authoritative is the one of Ivar Jacobson entitled "Taking the temperature of UML" [6], where he wrote: "Still, UML has become complex and clumsy. For 80% of all software only 20% of UML is needed. However, it is not easy to find the subset of UML which we would call the 'Essential' UML. We must make UML smarter to use". The need to simplify the UML is also shown by the recently released OMG draft proposal about this topic [12].

In the tentative to find the "essential UML", Erickson and Siau [3] have conducted a Delphi study[3] with the goal of identifying a UML kernel for three well-known UML application areas: Real-Time, Web-based, and Enterprise systems. The participants to the study (44 experts in total) were asked to rate the relative importance of the various UML diagrams in building systems. UML overall results (i.e., non-domain specific) were: 100% for Class and Statechart diagrams, 95.5% for Sequence diagrams, 90.9% for Use Case diagrams. All the others diagrams received a percentage lesser than 50% (e.g., 27.3% for Activity diagrams). Another personal opinion survey (sample size = 131) about UML [4] confirms the results of Erickson and Siau. Results indicate that the three most important diagrams are Use Case diagrams, Class diagrams and Sequence diagrams.

[2] e.g., `www.devx.com/architect/Article/45694` and `blogs.msdn.com/b/sonuarora/archive/2009/11/02/simplify-uml.aspx`

[3] It attempts to form a reliable consensus of a group of experts in specialized areas.

The main conclusions from another SLR by Budgen et al. [1] about empirical evidence of the UML are two:

– while there are many studies that use the UML in some way, including to assess other topics, there are relatively few for which the UML is itself the object of study, and hence that assess the UML in some way (e.g., UML studies of adoption and use in the field).
– there is a need to study the UML and its forms much more rigorously and to identify which features are valuable, and which could be discarded.

3 Study Definition

The instrument we selected to take a snapshot of the state of the practice concerning UML usage is that of a survey. In the survey's design and execution phases we followed as much as possible the guidelines provided in [5] and used the same presentation format of [13, 11].

The survey has been conducted through the following steps: (1) goals selection, (2) goals transformation into research questions, (3) identification of the population, sample and process, (4) data extraction and, (5) analysis of results and packaging.

We conceived and designed the survey with the goal of understanding *which are the less/most used parts of the UML in practice*. Within the scope of this work, in this paper we aim at addressing the main research question related to the above described goal:

Which of the 14 types of UML diagrams are the most/less used in practice?

The first step to conduct a survey is defining a target population. The target population of our study consists of *sources* concerning UML. In particular, in this study we considered the following four kinds of sources: books, tools, courses and tutorials.

To sample the population and select the sources to consider in our study we: (1) conducted a systematic search performed using Internet resources, Web search engines and electronic databases and (2) used non-probabilistic (convenience sampling) methods [7]. Moreover, in making decisions about whether (or not) to include a source in the study, we adopted some inclusion/exclusion criteria (see subsection below).

After having collected the sources, we extracted the data of interest for our research questions and finally we performed the analysis. Given the nature of this survey, that is mainly descriptive (it describes some condition or factor found in a population in terms of its frequency and impact) and exploratory, we mainly applied descriptive statistics and showed our findings by means of charts.

3.1 The Inclusion and Exclusion Criteria

The inclusion/exclusion criteria can be common for all the kind of sources or specific. For all the sources we adopted the following inclusion criterion: only sources concerning UML versions ≥ 2.0. Concerning books, in case of different editions of the same book we opted (when possible) for the last one. Moreover, we excluded

elements of "grey" literature, i.e., books without ISBN. Concerning tools, we included only UML modelling tools (both commercial and non-commercial) and excluded: (1) general graphics editor (e.g., Inkscape), (2) tools providing only a specific type of diagram (e.g., class diagrams), (3) really unstable, not complete or preliminary tools (e.g., tools in beta version). About courses, we considered only university courses concerning IT studies. We considered courses offered also in languages different from English (e.g., Italian and Spanish). Concerning tutorials, we considered only tutorials provided on Internet as written documents (either on-line or downloadable) and video (where a person gives instructions on how to do something) but we have excluded tutorials taking the form of a screen recording (screencast) and interactive tutorial. For selecting a document of this kind we used the common meaning/perception of tutorials: a tutorial is more interactive and specific than a book or a lecture; a tutorial seeks to teach by example.

3.2 The Process

The process followed to conduct a survey should be as much as possible well defined in order to ensure that such a study be objective and repeatable. For each category of sources, we followed a different process to collect the sources.

Books. We started by the Amazon website[4] and used the search form to find UML related books. We selected the "Computers & Technology" category in the books section. Then, we experimented with several different search criteria using different combinations of strings. Finally, the one that retrieved the highest number of useful items was the simple string "UML 2". Starting from this long list of books ordered by relevance[5] we filtered out books not satisfying the inclusion criteria explained above. Then, we tried to recover them using the facilities provided by our library. Finally, we collected and analysed 30 books. Note that, 18 of them are in the top 24 books ranked ordering the list by relevance by Amazon. The complete list of books is not reported here for space reason and can be found on the online technical report [10]. It includes, for instance, *"The Unified Modeling Language Reference Manual"* and *"The Unified Modeling Language User Guide"* by J. Rumbaugh, I. Jacobson, G. Booch and *"UML Distilled"* by M. Fowler.

Tools. We started by the "List of Unified Modeling Language tools" Wikipedia page[6] containing 49 UML tools. Then, we considered also the UML-tools website[7]. A full Internet search was also carried out using Google. Also in this case, we experimented with several different search criteria using different combinations of strings to provide to Google ("UML tools", "UML tools list" and "UML free tools"). For each tool of our list, we found the official website and selected only the tools satisfying the inclusion criteria explained above. Then, we downloaded and installed the most recent version of all the selected tools. In case

[4] http://www.amazon.com

[5] 2.726 books on July 20, 2013.

[6] http://en.wikipedia.org/wiki/List_of_Unified_Modeling_Language_tools

[7] http://www.uml-tools.com

of commercial tools, we selected a "free for not commercial use" version or a version with university licence or a trial version. At the end, we collected and analysed 20 different tools. The complete list of tools is not reported here for space reason and can be found on the online technical report [10]. It includes, for instance, *"Visual Paradigm"*, *"MagicDraw"* and *"IBM Rational SW Architect"*. Finally, we tried to produce a model containing the diagrams and constructs of interest for our study (for each tool we produced the same model with the same diagrams and the same UML constructs).

Courses. We started carrying out a search using Google. The combinations of strings used were: "UML course", "UML lecture" and "UML university course". We found several university courses satisfying the inclusion criteria stated above, but in several cases it was difficult, if not impossible, to recover the slides of the lectures, and in general the material. Often, the material was not publicly available; only the content of the lessons was present on the website. For this reason, we resort also to convenience sampling, asking to our colleagues the slides of UML courses they teach. At the end, we collected and analysed 22 different University courses. The complete list of lectures is not reported here for space reason and can be found on the online technical report [10]. It includes, for instance, courses from Canada, UK, USA, Hungary, Germany, Italy, France, Spain, Argentina, Australia. Convenience sampling was also useful to balance a little the geographic origin of the UML courses (e.g., before convenience sampling we had three France courses and zero USA courses).

Tutorials. We started with the tutorials lists present in the following three websites: www.uml.org[8], stackoverflow.com[9] and www.jeckle.de[10]. Then, we integrated the obtained results with other tutorials recovered using Google (the research was conducted using the strings: "UML tutorials" and "UML guide"). Finally, we collected and analysed 18 tutorials. The complete list of tutorials is not reported here for space reason and can be found on the online technical report [10].

4 Survey Results

We preliminarily decided to interpret the results of our survey assuming that a diagram is *widely used* if it is present in the $\geq 60\%$ of the sources, similarly it is *scarcely used* if it is present in $\leq 40\%$ of the sources, having also some non-defined cases (*grey zone*). In the following subsections we present the results concerning UML diagrams.

The level of usage of the various UML diagrams in books, courses, tutorials, tools, and in the totality of the sources respectively is summarized in Fig. 1. If we consider the totality of the sources, disregarding their kind, we have that the scarcely used diagrams are timing, interaction overview and profile, listed starting from the most used; all of them were not present in UML 1.x, and the profile diagram appeared only in version 2.2. The last position of the profile

[8] http://www.uml.org/#Links-Tutorials

[9] http://stackoverflow.com/questions/1661961/recommended-uml-tutorials

[10] http://www.jeckle.de/umllinks.htm#tutorials

UML Diagram	Book Guide	Book Spec	Book Tot	Tool	Course	Tutorial	All Sources
Class	100%	100%	100%	100%	100%	100%	**100%**
Composite Structure	87%	60%	73%	80%	14%	33%	**52%**
Component	93%	80%	87%	85%	59%	89%	**80%**
Deployment	93%	80%	87%	90%	55%	89%	**80%**
Object	93%	80%	87%	70%	55%	67%	**71%**
Package	100%	79%	89%	65%	52%	67%	**70%**
Activity	100%	93%	97%	100%	95%	100%	**98%**
Sequence	100%	93%	97%	100%	100%	89%	**97%**
Communication	100%	80%	90%	90%	59%	89%	**82%**
Interaction Overview	80%	53%	67%	45%	5%	28%	**39%**
Timing	87%	53%	70%	40%	5%	33%	**40%**
Use Case	100%	93%	97%	100%	95%	89%	**96%**
State Machine	100%	93%	97%	100%	95%	89%	**96%**
Profile	7%	13%	10%	30%	0%	6%	**11%**

Fig. 1. Usage levels of UML diagrams - ("Book Tot" means all the books)

diagram is not very surprising due both to the late appearance and to the fact that this kind of diagram has a very restrict scope (indeed it is used only to present a profile) and that, it is essentially a variant of the package diagram. Also timing diagrams, see an example in Fig. 2, have a restrict scope, and UML offers other ways to model time related aspects (e.g., timed events may be used in state machines and activity diagrams; durations and time intervals may appear in sequence diagrams), and this may be the motivation for their low usage. Finally, interaction overview diagrams are quite complex and in many cases may be replaced by sequence diagrams and/or a combination of sequence and activity diagrams, and perhaps this is the reason for not being so considered.

The widely used diagrams, when considering the totality of the sources, are instead, listed again starting from the most used ones, class (100% in any kind of sources), activity, sequence, state machine and use cases, communication, deployment, component, object and package diagrams. The first position of class diagrams is not surprising, it is indeed the fundamental diagram of the UML, while the fact that activity diagrams are the second is relevant and is due, in our opinion, to the fact that they are used also for business process modelling and for SOA based systems. All the widely used diagrams, except the package diagram, were already present in the UML 1.x versions (also if the communication diagrams were before called collaboration diagrams). The only diagram in the grey area (i.e., above 40% and below 60%) is the composite structure, that allows to represent both structured classes and collaborations; again it is a new diagram appearing in the UML 2.0 and this may be a reason for its low usage. How-

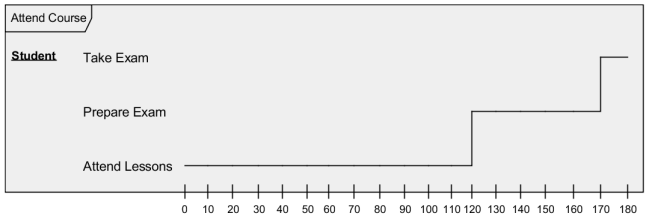

Fig. 2. Timing Diagrams, usage level 40% in the totality of the sources

ever, the result is surprising because structured classes were completely absent in UML 1.x, and this was a perceived problem, and the new collaborations are truly useful (see for example the big role that they have in representing service oriented architectures in the SoaML OMG standard profile [9]).

The results are different, see Fig. 1, if we distinguish the various kind of sources. In the case of books we have considered two kinds of books: 15 UML guides (Book Guide) and 15 books that make usage of UML but where it is not the primary subject (Book Spec). In the case of the UML guides, all diagrams except the profile diagrams are widely used. This result may perhaps be due to the fact that we count as present in a book a diagram also if it is only mentioned. Unfortunately, it is really difficult to devise a better metrics; for example trying to distinguish if a diagram is just mentioned, shortly presented, presented, or presented with all the details may to be too much depending on the personal judgement of who examines the books; also counting the occurrence of the name of a construct/diagram is in our opinion too dependent on the way the books were written, e.g., more or less verbose. We have also tried to distinguish the case of a simple mention of a construct in the text and the presence of an example of such construct, without detecting a relevant difference. Similar results, even if slightly reduced in magnitude, came from the other category of books (Book Spec).

For the tools the only scarcely used diagrams, see Fig. 1, are timing and profile diagrams, whereas the interaction overview diagram is the only one in the grey area. This result is a little surprising, since one can expect that due to the effort required to add a feature to a tool, the less relevant diagrams are omitted in many cases. In our opinion, since once you have available the functionalities to draw a class diagram it is quite easy to add the possibility to handle composite structure, component, deployment, object, package, communication and profile diagrams, whereas timing and overview requires new graphical functionalities.

Considering only the courses, we have a striking distinction between the diagrams; indeed, class, sequence, activity, state machine and use case diagrams are widely used with percentage over 90%, component diagram, deployment, object package, communication are in the grey area (all above or equal to 50%), whereas composite structure (14%), interaction overview (5%) and timing (5%) are really scarcely used. A lecturer preparing in a course has to decide which are the most relevant diagrams to present to the students in the allowed time slots, and it seems that this decision is quite homogeneous: there are five fundamental diagrams in the UML for the lecturers (class, sequence, activity, state machine and use case).

For the tutorials, we have a neat distinction between the widely used diagrams (class, activity, component, deployment, sequence, communication, use case and state machine) and the scarcely used (composite structure, timing, interaction overview and profile diagrams), but the differences in the usage level are lower than for courses, e.g., composite structure, timing, interaction overview percentages are more than 25%. In this case the tutorial writer has to decide

what is relevant, but s(he) usually has no strong constraints on the size of the tutorial itself.

5 Threats to Validity

To avoid to bias the results of our survey, we have considered only sources concerned with the use of the UML, avoiding those with different aims, for example drawing tools suitable to produce pictures of UML diagrams, or books presenting a survey on the current visual notations have been excluded; whereas instead books covering specific use of the UML or courses about software engineering where the UML was taught were included.

For the tools, instead, we are quite confident to have examined almost all the available ones; we think that a UML tool cannot exist without being presented somewhere on the Web. Notice that Argo UML, one of the most known UML tool was not included in our survey since it does support only UML 1.x.

We have considered here only four kinds of sources (books, courses, tutorials, sources) and we are aware that these are not the only ones; indeed there are also the UML users, and we are ready to launch a personal survey to investigate which constructs they know and which they use.

Finally, we have decided to define widely used (scarcely used) when a diagram was considered in the $\geq 60\%$ ($\leq 40\%$) of the sources, resulting also in a grey area. We think that this a sensible choice, using a threshold lower than 60%, e.g., 50%, should have led to have that a construct is either widely used or scarcely used without any doubt cases, and this does not sounds realistic. On the other hand, a higher threshold, e.g., 80%, should have led to a quite large number of inconclusive cases. We have also computed the widely/scarcely used on the totality of the sources, disregarding the fact that they are of very different kinds, e.g. books and courses, and so assigning to them different weights would have been more realistic. Again, we had the problem to compute these weighs in an unbiased way: is it sensible to say that a book is three times more relevant than a course, or that a tool is two times more relevant than a tutorial? To avoid to make our result too dependent on our personal judgement we have preferred to assume that all the sources have the same weight.

6 Conclusions

We have investigated, by means of a survey, how much the UML diagrams are used, considering in this paper four kinds of sources: books, tools, courses, and tutorials concerning the UML itself. The results of our survey show that the usage of the various diagrams is different . An "essential" UML seems to emerge also if its boundaries are not exactly defined. For what concerns the diagrams, class, activity, sequence, use case and state machine diagrams are widely used without any doubts, whereas interaction overview, timing and profile diagrams are really scarcely used.

In this paper we have considered only "objective" sources and examined them for checking if UML diagrams are used in an objective way (e.g., can a tool

produce a model including such diagrams?, is a course/tutorial teaching the fact that UML has such diagrams?), so these results are not biased by any personal opinion (neither ours nor of any human being taking part in the examination of the sources). We are now investigating the usage of the constructs used in the UML diagrams and launching a personal survey to investigate which UML diagram/constructs are known and used by UML users trying to cover different categories of them, and different applicative fields. The combined results of the two surveys should lead to finally determine an "essential" UML.

References

1. D. Budgen, A. J. Burn, O. P. Brereton, B. A. Kitchenham, and R. Pretorius. Empirical evidence about the UML: a systematic literature review. *Software Practice and Experience*, 41(4):363–392, Apr. 2011.
2. B. Dobing and J. Parsons. How UML is used. *Communications of the ACM*, 49(5):109–113, May 2006.
3. J. Erickson and K. Siau. Can UML be simplified? practitioner use of UML in separate domains, (white paper).
4. M. Grossman, J. E. Aronson, and R. V. McCarthy. Does UML make the grade? insights from the software development community. *Information and Software Technology*, 47(6):383–397, Apr. 2005.
5. R. M. Groves, F. J. J. Fowler, M. P. Couper, J. M. Lepkowski, E. Singer, and R. Tourangeau. *Survey Methodology*. John Wiley and Sons, 2009.
6. I. Jacobson. Taking the temperature of UML. WEB site `blog.ivarjacobson.com/taking-the-temperature-of-uml/`, 2009.
7. B. Kitchenham and S. Pfleeger. Personal opinion surveys. In F. Shull and Singer, editors, *Guide to Advanced Empirical Software Engineering*, pages 63–92. Springer London, 2008.
8. P. Mohagheghi, V. Dehlen, and T. Neple. Definitions and approaches to model quality in model-based software development - a review of literature. *Information and Software Technology*, 51(12):1646–1669, Dec. 2009.
9. OMG. *Service oriented architecture Modeling Language (SoaML) Specification Version 1.0.1*, 2012. Available at `www.omg.org/spec/SoaML/1.0.1/PDF`.
10. G. Reggio, M. Leotta, F. Ricca, and D. Clerissi. What are the used UML diagrams? A preliminary survey - Technical Report. Technical report, DIBRIS, University of Genova, July 2013. Available at `http://softeng.disi.unige.it/TR/UsedUMLTechnicalReport.pdf`.
11. G. Scanniello, F. Fasano, A. D. Lucia, and G. Tortora. Software quality assessment and error/defect identification in the Italian industry preliminary results from a state of the practice survey. In *Proceedings of 6th International Conference on Software Engineering Advances (ICSEA 2011)*, pages 589–594, 2011.
12. E. Seidewitz. Uml 2.5: Specification simplification. Presented at "3rd Biannual Workshop on Eclipse Open Source Software and OMG Open Specifications", 2012.
13. M. Torchiano, M. D. Penta, F. Ricca, A. D. Lucia, and F. Lanubile. Migration of information systems in the italian industry: A state of the practice survey. *Information and Software Technology*, 53(1):71 – 86, 2011.
14. M. Torchiano, F. Tomassetti, F. Ricca, A. Tiso, and G. Reggio. Relevance, benefits, and problems of software modelling and model driven techniques a survey in the italian industry. *Journal of Systems and Software*, 86(8):2110 – 2126, 2013.
15. UML Revision Task Force. *OMG UML, Superstructure, V2.4.1*, 2011.

Model-based Simplified Functional Size Measurement – an Experimental Evaluation with COSMIC Function Points

Vieri del Bianco, Luigi Lavazza,
Geng Liu, Sandro Morasca

Dipartimento di Scienze Teoriche e Applicate
Università degli Studi dell'Insubria – Varese, Italy

{luigi.lavazza|sandro.morasca}
@uninsubria.it
{vieri.delbianco|giulio.liu}
@gmail.com

Abedallah Zaid Abualkishik

IT department, Al-Khawarizmi
International College
Abu Dhabi, United Arab Emirates

azasoft@yahoo.com

Abstract. Functional Size Measurement methods –like the COSMIC method– are widely used but have two major shortcomings: they require a complete and detailed knowledge of user requirements and they are carried out via relatively expensive and lengthy processes. To tackle these issues, simplified measurement processes have been proposed that can be applied to requirements specifications even if they are incomplete or not very detailed. Since software requirements can be effectively modeled using languages like UML and the models increase their level of detail and completeness through the development lifecycle, our goal is to define the characteristics of progressively refined requirements models that support progressively more sophisticated and accurate measurement processes for functional software size. We consider the COSMIC method and three simplified measurement processes, and we show how they can be carried out, based on UML diagrams. Then, the accuracy of the measurement supported by each type of UML model is empirically tested, by analyzing the results obtained on a set of projects. Our analysis shows that it is possible to write progressively more detailed and complete user requirements UML models that provide the data required by simplified methods, which provide progressively more accurate values for functional size measures of the modeled software. Conclusions. Developers that use UML for requirements model can obtain an estimation of the application's functional size early on in the development process, when only a very simple UML model has been built for the application, and can get increasingly more accurate size estimates while the knowledge of the product increases and UML models are refined accordingly.

Keywords: Functional Size Measurement; Function Points; COSMIC Function Points; Simplified measurement processes; model-based measurement; UML.

1 Introduction

Functional Size Measurement (FSM) aims at providing a measure of functional user requirements. User requirements can be expressed by using various notations, including UML. It has been shown that the most popular FSM methods –namely, IFPUG Function Points (FP) and COSMIC Function Points (CFP)– can be applied to requirements written in UML, especially if the UML models have been written with FSM in mind, so that all (and only) the information required by FSM methods is suitably represented in the models [1,2].

UML models are collections of diagrams. While progressing in the development, UML models become more and more complete and detailed and in general include an increasing number of diagrams. This means that UML models convey an increasing amount of information, which can be used for FSM [3]. This is interesting for the application of simplified FSM methods, which require only a subset of the information needed to carry out the complete official measurement processes described in manuals, such as the COSMIC counting manual [5]. Different UML models (i.e., models involving different subsets of diagram types) can support different simplified FSM methods [4].

The majority of simplified FSM methods address the simplification of Function Point Analysis, since the IFPUG process of measuring FP involves activities –such as the classification of transactions and data and the evaluation of the complexity of every transaction and logic data file– that require a relevant measurement effort, and can be carried out only when the specification of user requirements is fairly complete and detailed.

However, the process of measuring CFP (which is generally faster and less expensive than FP measuring) may also need to be simplified so it can be carried out faster and at a smaller cost than the official process required by the official counting manual [5] and on incomplete requirements specifications. This may happen because size estimates are usually needed by a given deadline (e.g., for cost estimation and bidding) or because detailed requirements specifications are not available (and will not be available for a while). Simplified measurement processes for CFP have been proposed (see for instance the section on “early or rapid approximate sizing” in [6]).

The availability of “simplified” measurement processes for CFP, which require descriptions of requirements at different levels of detail, and the fact that UML models evolve through the requirements elicitation phase by growing in completeness and details suggest the following research questions:

RQ1. During the requirements elicitation and specification phase, is it possible to write progressively more complete and detailed UML models that support progressively more accurate simplified CFP measurement methods?

RQ2. What is the accuracy of different simplified CFP measurement methods, i.e., how close are the estimated sizes they provide to the actual ones?

RQ3. Do simplified CFP measurement methods provide an accuracy level that increases with the amount of information required?

Note that we do not intend to address question RQ3 quantitatively. Rather, we look for a trade-off between the information elicitation effort required by a given size estimation method and the resulting estimate accuracy that can –subjectively– be considered reasonable.

To answer questions RQ1-RQ3, we measured a set of software applications via different simplified CFP measurement methods, using progressively more detailed and complete UML models; we obtained the values of the parameters on which the estimation methods are based and computed the estimated sizes and compared them with the sizes measured according to the COSMIC counting manual.

The remainder of the paper is organized as follows. Section 2 describes simplified measurement processes for COSMIC function points. Section 3 illustrates the UML models that support the simplified measurement processes. Section 4 illustrates the experimental analysis. Section 5 accounts for related work. Section 6 discusses the threats to the validity of the study, while Section 7 draws some conclusions and outlines future work.

2 Simplified Processes for Measuring COSMIC Function Points

The COSMIC FSM method requires that:

1. The functional processes of the application being measured be identified.
2. The data groups mentioned in the user requirements be identified.
3. For each functional process, the unique data movements involving each identified data group be counted. Data movements are classified into Entries and Exits (i.e., I/O movements) and Reads and Writes (to persistent storage). A data group is considered persistent if its value is stable between two consecutive functional process executions.

2.1 Size Estimation Based on the Mean Number of Data Movements per Functional Process

A first very rough simplification of the measurement process was proposed by COSMIC [6] and requires that only the first of the activities required for CFP measurement (the identification of functional processes) is performed. The only requirement for applying this simplified process according to [6] is the availability of historical data that allow the computation of the mean number of data movements per functional process (M_{DM}). If the software application to be measured is similar to those previously measured, it is reasonable to assume that the mean number of data movements per functional process of the new application will be close to M_{DM}. Thus,

$$CFP = M_{DM} \times \#FPr \quad (1)$$

where #FPr is the number of Functional processes.

2.2 Size Estimation Based on the Number of Functional Processes and the Number of Data Groups.

It is reasonable to assume that the size in CFP increases with the number of data groups (#DG): the more data groups, the more opportunities for data movements. A simplified computation of CFP can thus be obtained via a model like the following:

$$CFP = f(\#FPr, \#DG) \quad (2)$$

that is, a model that computes the estimated size by means of some formula (to be defined) applied to #FPr and #DG. This procedure is simpler than the "full" COSMIC counting process, as data movements do not need to be identified and classified. Besides, a conceptual model of the data involved in the application is usually built very early in the requirements modeling process; thus, its availability is generally an easily satisfied assumption.

Equation (2) could be derived via regression analysis, provided that historical data reporting both #FPr and #DG are available.

2.3 Size Estimation Based on the Data Groups Involved in Each Functional Process

The two methods described above are based on measures (#FPr and #DG) that characterize the whole application. It is reasonable to expect that a more accurate estimate can be derived if information that characterizes each functional process individually – like the number of data groups involved in each functional process –is used. If the historical dataset includes such data, statistical analysis can yield models of type

$$CFP = f(\#FPr, AvDGperFPr) \quad (3)$$

where AvDGperFPr is the average number of data groups involved in each of the functional processes in the application to be measured.

3 UML Models Supporting Simplified CFP Measurement

In this section, we describe the UML models that are needed to support the simplified approaches to CFP measurement described in Section 2. We also present the model supporting the measure of CFP performed as described in the manual [5]. We use the Software Warehouse Portfolio application described by Fetcke [7] as an example.

The UML models used for measurement are models of the functional user specifications. They do not contain any design element; on the contrary, only information that is relevant to the user and is directly related to user's needs and requirements is allowed in models.

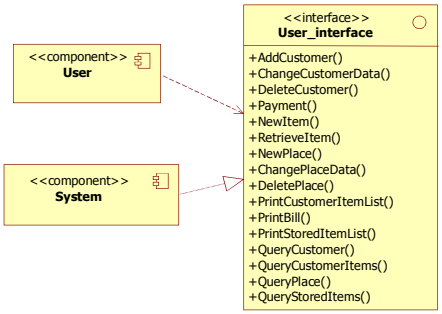

Fig. 1. UML component diagram showing the functional processes.

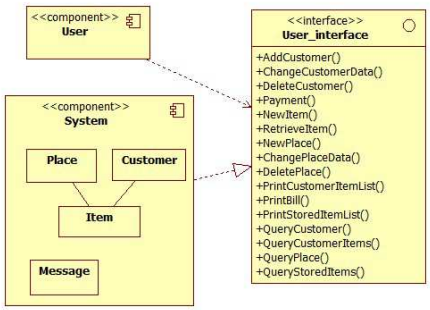

Fig. 2. UML component diagram showing the functional processes and the data groups.

Fig. 1 illustrates a UML diagram that can effectively support the first simplified measurement method, described in Section 2.1. It is a component diagram, where the interface realized by the system lists the functional processes that can be triggered by the user. So, #FPr can be obtained by counting the operations listed in the User_interface. Recall that M_{DM} can be obtained based on historical data.

Fig. 2 illustrates the same diagram as Fig. 1, in which the system component has been refined and detailed with the description of the classes that represent the data managed by the system. These classes correspond to the data groups of the COSMIC software model. It is easy to see that the diagram in Fig. 2 provides all the data needed to use equation (2), i.e. #FPr and #DG.

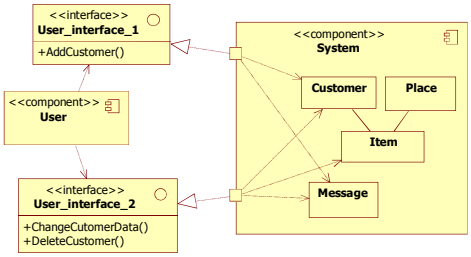

Fig. 3. UML component diagram showing the dependence of each functional process on data groups.

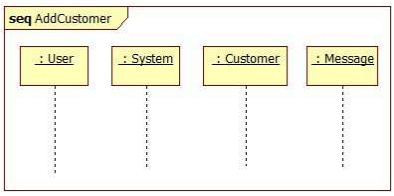

Fig. 4. UML sequence diagram showing the class (data group) instances participating in the AddCustomer functional process.

Fig. 3 illustrates a diagram providing the information needed to use equation (3). In the diagram, UML ports are used to precisely indicate which classes (i.e., data groups) are used in each functional process. To this end, sets of functional processes that use the same set of classes are grouped into a single interface. In Fig. 3, only the interfaces needed to add, change, and delete clients are shown. It can be noticed that grouping functional processes according to the used classes may lead to a rather large number of interfaces, which could decrease the readability of the diagram. This is true, but interfaces that are homogeneous with respect to the used classes not only allow for a quite precise estimation of size (see Section 4), but explicitly represent the logical

relationship between interface elements and system data: this poses the basis for the identification of important traceability information when the design model is built.

An alternative to the model shown in Fig. 3 is a sequence diagram that shows only the classes involved in the functional process (Fig. 4). In fact, the diagram represents a specific functional process (AddCustomer) and the involved class instances. We can see that AddCustomer uses two data groups: Customer and Message. This type of diagram is convenient because it can be refined into the diagram described in Fig. 5, which supports full-fledged COSMIC measurement.

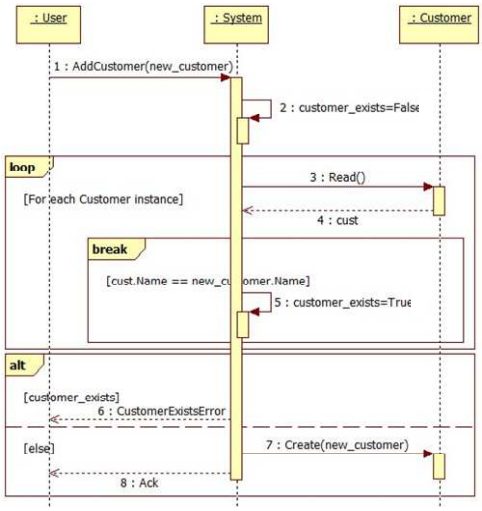

Fig. 5. UML sequence diagram showing the data movements of a given functional process

Fig. 5 illustrates a sequence diagram that contains all the information needed to measure the size of the functional process according to the COSMIC official manual [5]. Messages that cross the application boundary (in our case, messages from or to the user) are entries and exits, while messages directed to class instances representing data groups are reads or writes. Details about COSMIC measurement based on UML diagrams can be found in [1] and [2].

4 Empirical Analysis

To answer the questions defined in the Introduction, we modeled a set of software applications and measured them. Then, we applied the simplified measurement methods, obtaining size estimates that were finally compared with the measures obtained via the standard COSMIC method [5].

The projects considered were sample projects provided by COSMIC to illustrate the counting process (5 projects), academic examples used in teaching (5 projects) and project management tools (one project).

The UML models were built by a PhD student following the methodology described in [1]. The quality of the model was then checked by two of the authors. Part

of the dataset containing the measures of the models of the applications described above is given in Table 1.

Table 1. The dataset

Pid	CFP	#FPr	#DG	Avg #DG per FPr	Avg DM per DG per FPr	Avg FPr size (DMperFPr) others	Avg #DG per FPr others	Avg CFP/#DG others
1	86	16	6	2.88	1.90	5.12	3.01	8.7
2	56	11	11	3.55	1.60	5.14	2.98	9.6
3	91	15	10	4.00	1.57	5.08	2.93	9.1
4	69	19	12	2.32	1.72	5.26	3.06	9.6
5	103	19	16	3.06	1.93	5.09	3.00	9.5
6	64	14	7	2.64	1.71	5.17	3.03	9.2
7	116	20	14	3.60	1.65	5.07	2.95	9.3
8	124	20	10	2.80	2.38	5.04	3.02	8.9
9	117	19	9	3.47	1.78	5.05	2.97	8.8
10	90	13	14	3.92	1.99	5.04	2.95	9.5
11	10	3	10	2.07	2.40	5.43	3.30	9.1

In Table 1, "Avg#DGperFPr" is the average number of data groups involved in the project's functional processes; "AvgDMperDGperFPr" is the average number of data movements per data group per functional process; "AvgFPrSize (DMperFPr) others" is the mean number of data movements per functional process, computed on all the other applications; "Avg#DGperFPr others" is the mean number of data groups per functional process, computed on all the other applications; "AvgCFP/#DG_others" is the mean number of data movements (i.e., size) per data group, computed on all other applications.

When estimating the size of an application using equation (1), we used the M_{DM} of the other projects. The M_{DM} is equivalent to the mean CFP/NumFPr, i.e., to the mean size of Functional processes. Using this model we got the estimates reported in Table 2. The obtained estimates are characterized by MMRE = 17.8%, Pred(25) = 72.7%, percentage error range = [-27.8%, 44.9%].

While analyzing the dataset, we discovered that the mean number of data movements per data group involved in a functional process, computed for each application, was fairly constant throughout the applications of our dataset: the mean is 1.88 and the standard deviation 0.29 (i.e., 15% of the mean). We exploit this fact to define the following model:

$$CFP = NumFPr \times AvDGperFPr \times AvDMperDGperFPr \quad (4)$$

where (AvDGperFPr × AvDMperDGperFPr) is an estimate of the number of data movements per functional process, i.e., an estimate of the mean size of functional

processes: multiplied by the number of functional processes it yields an estimate of the number of data movements, i.e., the size of the application.

By using this model, we obtained the estimates reported in Table 2 and characterized by MMRE = 15.3%, Pred(25) = 81.8%, percentage error range [-15.3%, 33.9%].

Table 2. Estimates obtained via equations (1) and (4)

		Estimates obtained via eq. (1)			**Estimates obtained via eq. (4)**		
P.Id	**Act. Size [CFP]**	**Est. Size [CFP]**	**Err. [CFP]**	**% Err.**	**Est. Size [CFP]**	**Err. [CFP]**	**% Err.**
1	86	82	-4	-4.7%	88	2	2.3%
2	56	57	1	1.8%	75	19	33.9%
3	101	86	-15	-14.9%	132	31	30.7%
4	69	100	31	44.9%	85	16	23.2%
5	103	92	-11	-10.7%	105	2	1.9%
6	64	72	8	12.5%	71	7	10.9%
7	116	101	-15	-12.9%	139	23	19.8%
8	124	101	-23	-18.5%	105	-19	-15.3%
9	117	96	-21	-17.9%	127	10	8.5%
10	90	65	-25	-27.8%	97	7	7.8%
11	252	326	74	29.4%	218	-34	-13.5%

5 Related Work

Many techniques for early size estimation have been proposed for Function Points (e.g., the Early and Quick Function Point by Conte et al. [8]). The empirical evaluation of these techniques indicates that some actually yield reasonable estimates [11]. On the contrary, hardly any work has been devoted to defining simplified measurement processes for the COSMIC method.

In [9], the dataset published in [10] was used to derive a linear OLS regression model that can be used to estimate the size in CFP, given the number of transactions identified via Function Point Analysis. This can be considered a sort of simplified CFP measurement method, since the identification transaction functions is an activity mush simpler and shorter than both the full fledged CFP or FP counting processes

Several authors studied the possibility of basing standard CFP measurement [5] on UML models of user requirements; i.e., they consider the models that are available after the completion of the requirements elicitation and specification phase.

Hericko and Zivkovic address size estimation in iterative development [3]. Their approach enables early size estimation using UML. However, they do not consider simplified measurement processes. In fact, their method deals with the evolution of

the functionality through iterations, rather than the level of detail that can be achieved in the requirements elicitation and specification phase, as we do.

6 Threats to Validity

A possible threat to internal validity is the limited number of projects in our sample.

The main threat to the external validity of the study may come from the projects chosen, which are a limited sample of a much larger population. However, this kind of threat is typical in most empirical software engineering studies. Also, the sample of projects is a "convenience" sample, i.e., it is made of projects that were selected because the data that we needed for our study were available. Note that, however, we are not interested here in specific models (e.g., we are not interested in the coefficients of the models), but, rather, in the performance of the techniques we propose. At any rate, it is not easy to assess the extent to which our results may apply in general.

There may be a threat to construct validity due to the use of MMRE, which has been criticized in the past as an accuracy indicator [13].

One might also observe that only one of the projects used within this empirical study is a real implemented project, and that this fact could possibly decrease the reliability of the results or their generality. However, this is not actually a problem, for two reasons. One is that the requirements of all our projects were realistic: any of our projects could be implemented, thus requiring for size measurement, effort estimation, etc. The second is that we are interested in the *comparison* of measures obtained via simplified and full-fledged processes. Therefore, some characteristic requirements that affect the standard size measure are bound to affect in the same way the simplified measure, so that the results are hardly affected.

7 Conclusions

Simplified FSM methods are often used when a project manager needs an estimate of the functional size of the software application to be built before the requirement specification phase is completed, or when the cost or time allowed for measurement are limited. When UML is used in the early phases of development, it is convenient to apply simplified FSM methods to UML models. In this paper we showed that it is possible to build UML models that adequately support the application of simplified measurement methods and the standard COSMIC method. In particular, during the requirements specification phase, UML models grow in detail, thus providing the information required by progressively more accurate size estimation methods. In fact, it was possible to define quantitative size estimation models based on only the number of functional processes, or the number of functional processes and the number of data movements in each functional process.

To practitioners, our results provide an interesting hint: the information contained in the UML models illustrated in Section 3 is just the information required to document applications' requirements properly; accordingly, in the requirements specification phase the analyst *must* indicate what data are involved in each functional process,

and how they are used. Therefore, size estimates can be seen as 'by products' of the progressive refinement of UML requirements models.

Acknowledgments

The research presented in this paper has been partially funded by the project "Metodi, tecniche e strumenti per l'analisi, l'implementazione e la valutazione di sistemi software" funded by the Università degli Studi dell'Insubria and by "Progetto dote 2 - programma UNIRE - accordo per lo sviluppo capitale umano nel sistema universitario lombardo", co-funded by Regione Lombardia and Università degli Studi dell'Insubria.

References

1. Lavazza, L., del Bianco, V., Garavaglia, C.: Model-based Functional Size Measurement, ESEM 2008, 2nd Int. Symp. on Empirical Software Engineering and Measurement, Kaiserslautern, Germany. October 9-10, 2008.
2. Lavazza, L., del Bianco, V.: A Case Study in COSMIC Functional Size Measurement: the Rice Cooker Revisited, Amsterdam, IWSM/Mensura 2009, November 4-6, 2009.
3. Hericko, M., Zivkovic, A.: The size and effort estimates in iterative development, Information and Software Technology vol. 50 n.7, 2008, pp.772-781.
4. del Bianco, V., Lavazza, L., Morasca, S.: A Proposal for Simplified Model-Based Cost Estimation Models, 13th Int. Conf. on Product-Focused Software Development and Process Improvement – PROFES 2012, Madrid, June 13-15, 2012.
5. COSMIC – Common Software Measurement International Consortium: The COSMIC Functional Size Measurement Method - version 3.0.1 Measurement Manual (The COSMIC Implementation Guide for ISO/IEC 19761: 2003), May 2009.
6. COSMIC: The COSMIC Functional Size Measurement Method - Version 3.0 - Advanced and Related Topics, December 2007.
7. Fetcke, T.: The warehouse software portfolio, a case study in functional size measurement, Technical report no.1999-20, Département d'informatique, Université du Quebec à Montréal, Canada, 1999.
8. Conte, M., Iorio, T., Meli, R., Santillo, L.: E&Q: An early and quick approach to functional size measurement methods, 1st Software Metrics European Forum (SMEF 2004), Roma, January 2004.
9. Lavazza, L.: Convertibility of functional size measurements: new insights and methodological issues, 5th Int. Conf. on Predictor Models in Software Engineering, 2009.
10. Van Heeringen, H.: Changing from FPA to COSMIC - A transition framework. in Proceedings Software Measurement European Forum (SMFE), Rome, Italy, 2007.
11. Lavazza, L., Liu, G.: A Report on Using Simplified Function Point Measurement Processes. The Seventh Int. Conf. on Software Engineering Advances. Lisbon, 2012.
12. Barkallah, S., Gherbi, A., Abran, A.:COSMIC Functional Size Measurement Using UML Models. In proceeding of: Software Engineering, Business Continuity, and Education - International Conferences ASEA, DRBC and EL, pp.137-146. 2011.
13. Kitchenham, B.A., Pickard, L.M., MacDonell, S.G., Shepperd, M.J.: What accuracy statistics really measure. IEE Proceedings - Software, 148(3), June 2001, pp. 81-85.

UML Usage in Open Source Software Development : A Field Study

Hafeez Osman[1] and Michel R.V. Chaudron[1,2]

[1] Leiden Institute of Advanced Computer Science, Leiden University, the Netherlands
hosman@liacs.nl
[2] Joint Department of Computer Science and Engineering Chalmers University of Technology and Goteborg University, Sweden
chaudron@chalmers.se

Abstract. UML is the de facto standard for modeling software designs and is commonly used in commercial software development. However, little is known about the use of UML in Open-source Software Development. This paper evaluates the usage of UML modeling in ten open-source projects selected from common open-source repositories. We evaluated the usage of UML diagrams based on the information available in the open-source projects repositories. Our study covers the types of UML diagrams that are used and how frequently UML models are updated. Our findings also include the change in focus on types of diagram used over time and researches on how the size of models relates to the size of the implementation.

Keywords: UML, Open-source Software Development, Reverse Engineering

1 Introduction

UML provides the facility for software engineers to specify, construct, visualize and document the artifacts of a software-intensive system and to facilitate communication of ideas [11]. For commercial software development, the use of UML is commonly prescribed as part of a company-wide software development process while in open-source software development (OSSD), there is typically no mandate on the use of UML. Only if the community of developers of the OSSD feels needs (e.g. for their communication) then UML diagrams are produced. Even though some open-source projects employ UML diagrams, these diagrams do not completely correspond with the implementation code. For instance, the number of classes used in class diagrams is typically less than the number of classes that exist in the implementation source code. The usage of UML class diagrams also varies across projects. Almost all OSSD projects that use UML choose to produce class diagrams. Some projects also constructed other types of UML diagrams such as use case diagrams, sequence diagrams and activity diagrams.

One of the benefits of UML is to ease communication between software developers. The nature of OSSD is that software developers normally communicate

with each other using some online communication medium (e.g discussion forum, e-mail, IRC) rather than through physical interaction. There is an anecdotal belief that UML is rarely used in OSSD. However, there is no quantitative research to prove this perception. In this paper, we aim at evaluating the usage of UML diagrams in OSSD projects. We want to discover how UML is used in OSSD without the influence of the stakeholders or users of the system. We explore the publicly software documentation to answer the following questions: 1) What types of UML diagrams are used? 2) How does the size of the design relates to the size of the implementation? and 3) How does timing of changes in the implementation relate to changes in UML diagrams/documentation?

The paper is structured as follows: Section 2 discusses related work. Section 3 describes the case studies used in this research. Section 4 describes the study approach while Section 5 presents the results and findings. This is followed by our conclusion and future work in Section 6.

2 Related Work

Dobing and Parsons [7] performed a survey to find out to what extent the UML is used and for what purpose, differences in the level of diagram use and how successful UML usage is for communication in a team. The research found that the most used types of UML diagrams were use case diagram and class diagram while collaboration diagram was used the least. Dobing and Parsons also conducted another survey to discover the current practice in the use of UML in [8]. The findings of this survey highlighted that the most used UML diagrams were class diagram, use case diagram and sequence diagram.

Grossman et. al [9] performed a study to research the perspective of individuals using UML using the task-technology fit model. This study also addressed the characteristics that affect the usage of UML. Similar to [7] and [8], the result of the most three important diagrams in ranking are use case, class and sequence Diagram. Those studies also found out that it is difficult to discover whether UML provides too much detail or too little detail because it depends on the software technology (i.e. Enterprise System, Web-based system, real-time system) that requires UML to be tailored to the environment.

Yatani et. al [10] conducted an evaluation on the use of diagramming for communication among OSSD developers and also performed a semi-structured interview with developers from a large OSSD project. This study highlighted diverse type of diagram which is used for the communication between the contributors of the system. Not all diagrams used for communication purposes were updated during the project. The study extended by Chung et. al [12] with a survey participated by 230 OSSD developers and designers. For the frequency of updating designs, even though 76% agree that diagrams have value, only 27% practice diagramming very often or all the time during the software development.

Most of the related works use surveys to discover the usage of UML diagram. These surveys are based on the UML practitioners' perspective of how they use UML. In contrast, our study evaluates the use of UML modeling in

OSSD projects by mining the project documentation. Hence, this reflects the real artifacts produced by using the UML notation.

Table 1. List of Case Studies

Project	Description	No. of Releases	URL Source
Maze	Maze-solver is a Micro-Mouse maze editor and simulator.	2	http://code.google .com/p/maze-solver/
JavaClient	The project allows development of applications for Player/Stage using the Java programming language.	3	http://java-player. sourceforge.net/
xUML-Compiler (xUML)	xUml-Compiler takes a user specified data model and associated state machines and produces an executable and testable system.	13	http://code.google .com/p/xuml-compiler/
JPMC	Java Portfolio Management Component (JPMC) is a collection of portfolio management component.	1	http://jpmc.source forge .net/
Neuroph	Lightweight Java neural network framework to develop common neural network architectures.	9	http://neuroph .sourceforge .net/
Gwt-portlets	Free open-source web framework for building GWT (Google Web Toolkit) applications.	6	http://code.google .com/p/gwtportlets/
Wro4J	The project purpose is to improve web application page loading time.	3	http://code.google .com/p/wro4j/
JGAP	Genetic Algorithms and Genetic Programming package.	8	http://jgap.sourceforge .net/
ArgoUML	An open-source UML modeling tool and include support for all standard UML 1.4 diagrams.	19	http://argouml. sourceforge.net
Mars Simulation	Free software project to create a simulation of future human settlement of Mars.	26	http://mars-sim. sourceforge.net/

3 Case Studies

Based on researches by Hutchinson et. al [1], Dobing and Parsons [7], and Erickson et. al [2], we know that the most used UML diagram is the class diagram. Therefore, we performed a search for UML class diagram images using the Google search engine. In particular, we targeted our search on four open-source repositories: SourceForge, Google Code, GitHub and BerliOS. The main keyword used for the search was "class diagram". Based on the hits of these searches, we browsed the project repositories. Our initial list of candidate cases consisted of 57 projects. We refined the selection by using the following criteria:

- The project should have UML diagrams and corresponding source code (project that have multiple versions is preferred)
- The source code should be written in Java

Projects that are developed in Java is selected because we need to reverse engineer the source code to class diagram for analysis purposes. The reverse engineering tools that we used for this study performs best with Java source codes. We found ten software projects which are suitable that are listed in Table 1.

4 Approach

This section describes the approach we used in this study. We had three main activities in order to answer the following research questions :
RQ1 : What types of UML diagrams are used? Based on the project repository, we manually browsed the documentation and other provided information to find all the UML diagrams that were used in the project.
RQ2 : How does the size of the design relate to the size of the implementation? Our aim was to use one single tool for counting classes of both the design and implementation. Furthermore, for source code, we only wanted to count classes that were actually *designed* for the project's system, hence to exclude library classes that are imported, and would typically not be modeled. To this end, source codes were reverse engineered using several Computer Aided Software Engineering (CASE) tools i.e MagicDraw 17.0[3] and Enterprise Architect 7.5[4]. The reverse engineered design was then exported to XML Metadata Interchange (XMI) files. From the resulting XMI files, software design metrics were computed using the SDMetrics [5] tool and all library classes were removed.
RQ3 : How does timing of changes in the implementation relates to changes in UML diagrams/documentation? For source code, we manually extracted the dates of releases from the project repositories. For UML diagrams, we looked at the date-information provided by the system documentation, developer's manual and other related documents in the project repository.

5 Results and Findings

In this section, we present the results. We group our results into the three questions presented in the previous section.

5.1 Usage of UML Diagrams

The UML diagram that was mostly used in our set of open-source projects is the class diagram. This is to be expected because our main keyword of searching

[3] http://www.nomagic.com/
[4] http://www.sparxsystems.com.au/
[5] http://www.SDMetrics.com/

for case study was based on class diagrams. Table 2 shows which other types of diagrams were used. The term 'yes' in Table 2 means that the project used at least one instance of a UML diagram specified in the table. The use of UML in OSSD projects seems driven by a need to codify high level knowledge. For example, ArgoUML did not use sequence diagrams in their modeling until there was a new feature. Only this new feature was explained by a sequence diagram. In general, the case studies show that the most used UML diagrams in OSSD are use case, component, package, class, sequence/interaction and activity diagram. The following subsections describe the results in more detail.

Table 2. UML Diagram Usage

			Structure Diagram					Behaviour Diagram		
No	Project	Use Case	Component Diagram	Package Diagram	Class Diagram	Composite Structure Diagram	Object Diagram	Sequence/ Interaction Diagram	Activity Diagram	State Machine Diagram
1	Maze	No	No	No	Yes	No	No	No	No	No
2	JavaClient	No	No	No	Yes	No	No	No	No	No
3	xUML	No	No	No	Yes	No	No	No	No	No
4	JPMC	Yes	No	Yes	Yes	No	No	No	No	No
5	Neuroph	No	No	No	Yes	No	No	No	No	No
6	Gwt-portlets	No	No	No	Yes	No	No	Yes	No	No
7	Wro4J	No	No	No	Yes	No	No	No	No	No
8	JGAP	No	No	No	Yes	No	No	No	No	No
9	ArgoUML	No	Yes	Yes	Yes	No	No	Yes	Yes	No
10	Mars	No	Yes	No	Yes	No	No	No	Yes	No

Use Case Diagram. Use case diagrams were used by only one OSSD project: JPMC (see Table 2). A use case diagram is used to describe the desired functionality of the software product [3]. Most of these OSSD projects have specified their goal but the specification and the interaction between the user and system were explained in text.

Component Diagram. Component diagrams are used to divide the system into components and show their interrelationships through the breakdown of components into a lower-level structure [5]. ArgoUML provided one component diagram from an old design document to illustrate the interaction between early developed component and packages. The Mars Simulation project provided two component diagrams i.e. 1)'Top Level Diagram' illustrated dependencies between three components, and 2)'Simulation Component Diagram' illustrated more details about the relationship between a simulation component and other related components.

Package Diagram. Package diagrams provide a grouping construct that allows to group design elements together into higher-level units [5]. The JPMC project presents almost all main packages and their dependencies in a package diagram. Meanwhile, ArgoUML presented two package diagrams. The first package diagram in this project illustrated the dependencies between packages with

two other package representing external libraries. The second package diagram illustrated the high level package in this project.

Class Diagram. Class diagram is the most used diagram type in these case studies. Most of the case studies only show classes that are important in the system. The correspondence between classes and implementation is discussed in section 5.2.

Sequence/Interaction Diagram. Sequence diagrams were used by two OSSD projects. However, both projects have only one sequence diagram per project. ArgoUML introduced a sequence diagram after eight version releases. Table 4 shows that only after version 0.26, a sequence diagram was introduced in the documentation. Perhaps, it is difficult to construct the sequence diagram for the entire release hence, the developer of this project used sequence diagram for a new feature. The gwt-portlets project used only one sequence diagram. We assume that the described flow contains crucial information for the system because the classes that were involved in the sequence diagram were presented in a class diagram that shows the key classes of the system.

Activity Diagram. Activity diagramming or activity modeling emphasizes the flow and conditions for coordinating lower-level behaviours [4]. This study found that there were two OSSD projects used the activity diagram. However, not all activity diagrams in these projects are related to the software development. ArgoUML used an activity diagram to present the flow of managing issues in the project. Meanwhile, the Mars Simulation project used one activity diagram for specifying a specific feature of the project.

Table 3. Classes in Design versus Classes in Implementation

No	Project	No. of Classes in Design	No. of Classes in Implementation	% of Design in Implementation
1.	Maze	28	69	40.58
2.	JavaClient	57	215	26.51
3.	xUML	45	172	26.16
4.	JPMC	24	126	19.05
5.	Neuroph	24	179	13.41
6.	gwt-Portlets	20	178	11.24
7.	Wro4J	11	100	11.00
8.	JGAP	18	191	9.42
9.	ArgoUML	33	909	3.63
10.	Mars Simulation	31	953	3.25
	Total	291	3092	16.43

5.2 Ratio between Design and Implementation

This subsection presents the results of analyzing the ratio between classes in the design and classes in the implementation. Since there are multiple versions of both the design and implementation in most of the case studies, we chose a pair

with a high ratio of design to implementation. For example, for the Neuroph project, we selected version 2.3 because this version has a high number of designed classes due to the fact that the project starts updating UML diagrams at this point in time. The Maze project has the highest ratio of classes in design to classes in implementation. This is a relatively small project which consisted of 69 classes in the implementation and 40 % of these classes were represented in the UML design. In absolute numbers, the highest number of classes in a design is found in the JavaClient project with 57 classes.

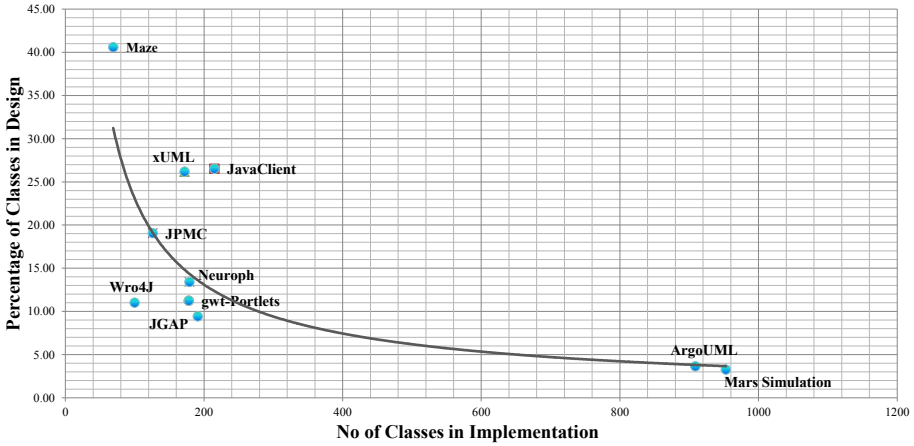

Fig. 1. Classes in Design vs Classes in Implementation

5.3 Frequency of Updating UML models

This subsection presents the frequency of updating the UML models of the case studies. We would like to know whether UML diagram is used throughout the projects or only in initial phases. We analysed the case studies that have multiple versions of releases to assess the frequency of updating the diagrams while the systems evolve through subsequent releases. Even though there were multiple versions of system releases for the Mars Simulation, JavaClient, JPMC, Gwt-Portlet, Maze and xUML-compiler project, their UML diagrams were not changed. For instance, the Mars Simulation project has released 26 versions of source code. The UML designs were only uploaded in Dec 2009. Based on that date, we assume that this design corresponds with this release version 2.87 and above. This indicates that the earlier 19 versions of the software did not have a UML model. However, we could not disregard the fact that the design may be created earlier than the date it was uploaded.

The result also shows that the frequency of updating UML diagram is low. In most of the case studies, a new UML diagram is created when there is a new feature of the system introduced in a new version or release. Only the Neuroph and ArgoUML project actually modified existing diagrams. Other project only added

new diagrams to their documentation, but did not modify previously existing diagrams. In the ArgoUML project, we found that there was an increasing amount of diagrams at the same time as the number of project contributors increases. The work by Wen Zhang et. al [6] shows that there was an increasing amount of participants in version 0.26. The ArgoUML project updated and added a lot of UML diagrams in version 0.26. We hypothesize that the documentation was elaborated to cater for a group of newcoming developers that are looking for information about the design.

Next, we discuss the ArgoUML project as an example of a project that did update their UML designs across multiple versions of releases. Table 4 shows which types of diagrams were used across subsequent versions over time. The table shows in early stages of development, diagrams were made that represent the high level structure of the system (component, package and class). As development time progresses, diagrams are added that represent the dynamic behaviour of the system through activity diagrams (v 0.16) and sequence diagrams (v 0.26). Also, at the later stages of development, component diagrams are no longer used. We believe this trend to be typical of the use of modeling in software development in general : Firstly, the developers design the overall structure and later continue to flesh out behavioural aspects of the design. Figure 2 shows the evolution of UML Diagrams in every versions of release. Figure 2 also shows the evolution of the number of classes. It is explicitly shown that the UML diagrams are created in the early stage of software release and then updated occasionally.

Table 4. List of UML Diagrams used in ArgoUML Project

No	Release Version	Date	Source	Component Diagram	Package Diagram	Class Diagram	Activity Diagram	Sequence/ Interaction Diagram
1	0.10.1	09.10.2002	Old Design Document	Yes	Yes	Yes	No	No
2	0.12	18.08.2003	Cookbook 2003 and Old Design Document	Yes	Yes	Yes	No	No
3	0.14	05.12.2003	Cookbook 2003 and Old Design Document	Yes	Yes	Yes	No	No
4	0.16	19.07.2004	Cookbook-0.16	No	Yes	Yes	Yes	No
5	0.18.1	30.04.2005	Cookbook-0.18.1	No	Yes	Yes	Yes	No
6	0.20	09.02.2006	Cookbook-0.20	No	Yes	Yes	Yes	No
7	0.22	08.08.2006	Cookbook-0.22	No	Yes	Yes	Yes	No
8	0.24	12.02.2007	Cookbook-0.24	No	Yes	Yes	Yes	No
9	0.26	27.09.2008	Cookbook-0.26	No	Yes	Yes	Yes	Yes
10	0.26.2	19.11.2008	Cookbook-0.26.2	No	Yes	Yes	Yes	Yes
11	0.28	23.03.2009	Cookbook-0.26.2	No	Yes	Yes	Yes	Yes
12	0.28.1	16.08.2009	Cookbook-0.26.2	No	Yes	Yes	Yes	Yes
13	0.30	06.03.2010	Cookbook-0.26.2	No	Yes	Yes	Yes	Yes
14	0.30.1	06.05.2010	Cookbook-0.26.2	No	Yes	Yes	Yes	Yes
15	0.30.2	08.07.2010	Cookbook-0.26.2	No	Yes	Yes	Yes	Yes
16	0.32	28.01.2011	Cookbook-0.26.2	No	Yes	Yes	Yes	Yes
17	0.32.1	23.02.2011	Cookbook-0.26.2	No	Yes	Yes	Yes	Yes
18	0.32.2	03.04.2011	Cookbook-0.26.2	No	Yes	Yes	Yes	Yes
19	0.34	15.12.2011	Cookbook-0.26.2	No	Yes	Yes	Yes	Yes

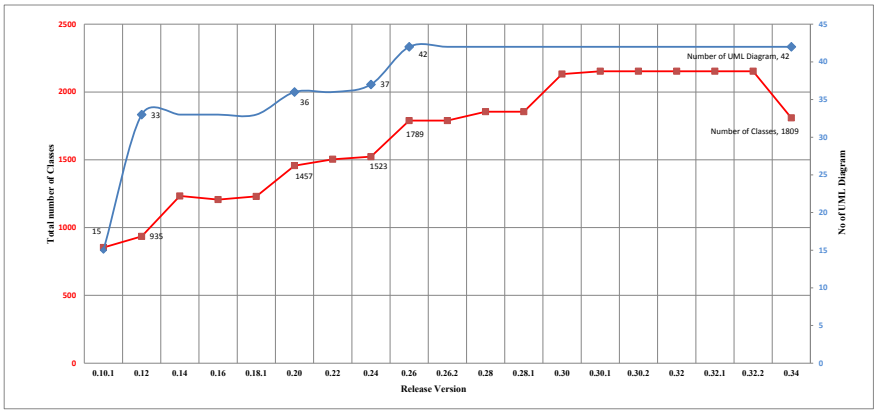

Fig. 2. ArgoUML Evolution in UML Diagrams and Number of Classes

5.4 Threats to Validity

This section describes the threats to validity of this study. In term of case study selection, there could be more case studies if we included more open-source repository. The selected projects may not represent all the OSSD because the selected case studies can be considered as small and medium type of system development. In addition, we also did not have projects with the number of classes between 250 and 800. The result could be different if more larger projects would be included in this study. The study was done based only on using the information in the project repositories and also the projects websites. It may be the case that developers use UML in their communication or for internal use without uploading their diagrams in the project repository. This study also only used the date listed as the upload-date of the documents in the repositories. The document may be created far before the uploaded date. Thus, the matching of the date of documentation and the version may not be accurate.

6 Conclusion and Future Work

This study aims to discover the usage of UML diagram in the OSSD projects. To this end, ten case studies were collected from online repositories. Three main questions were studied: What types of UML diagrams are used?, How does the size of the design relates to the size of the implementation? and How does timing of changes in the implementation relate to changes in UML diagrams/documentation? By studying the evolution of UML models across versions, we found that the focus of modeling shifts from structural aspects in the early phases of development, to dynamic behaviour in the later stages of development. The frequency of updating UML models is low. We found two triggers for updating UML diagrams: 1) if there are changes in the features of the system,

and 2) if there is a group of newcomers joining the project. The latter cause confirms the role of UML models as a way of codifying design knowledge for communicating the design. Overall, this paper shows that open-source projects can be used as empirical source for studying usage of UML modeling.

For future work, it would be interesting to extend this study by performing a broader survey or interview OSSD developers to find out the reasons for or against using UML diagrams in their development. Also, it is interesting to ask developers after their pattern in updating UML models. Finding more case studies and extending the case studies to languages other than Java. This would allow to differentiate results between programming languages.

References

1. J. Hutchinson, J. Whittle, M. Rouncefield and S. Kristoffersen, "Empirical Assessment of MDE in Industry," Proc. of the 33rd International Conference on Software Engineering(ICSE '11), pp. 471–480. ACM New York (2011)
2. Erickson, J., Siau, K.:Theoretical and Practical Complexity of Modeling Methods, In: Communications of the ACM, Vol. 50 Issue 8, pp. 46–51. ACM New York (2007)
3. Grechanik, M., McKinley, K.S., Perry, D.E.: Recovering And Using Use-Case-Diagram-To-Source-Code Traceability Links, In: ESEC-FSE '07 Proc. of the 6th joint meeting of the European softw. eng. conference and the ACM SIGSOFT symposium on The foundations of softw. eng., pp. 95–104 ACM New York (2007)
4. Object Management Group(OMG): Unified Modeling Language Specification, Superstructure Version 2.4.1, `http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF/`,pp. 303, (2011)
5. Fowler, M.: UML Distilled 3rd Edition. A Brief Guide to the Standard Object Modeling Language, pp.89, 141. Addison-Wesley (2004)
6. Zhang, W., Yang, Y. Wang, Q.: Network Analysis of OSS Evolution: an Emperical Study on ArgoUML Project. In: IWPSE-EVOL '11 Proc. of the 12th International Workshop on Principles of Software Evolution and the 7th annual ERCIM Workshop on Software Evolution, pp. 71–80, ACM New York (2011)
7. Dobing, B., Parsons,J.: How UML is used, In: Communications of the ACM, Vol. 49, Issue 5, pp. 109–113, ACM New York (2006)
8. Dobing, B., Parsons,J.: Current Practice in the Use of UML, In: Proceeding of ER 2005 Workshops AOIS, BP-UML, CoMoGIS, eCOMO, and QoIS, pp.2–11, Springer-Verlag Berlin Heidelberg (2005)
9. Grossman, M., Aronson, J.E., McCarthy, R.V.: Does UML make the grade? Insights from the software development community, In: Information and Software Technology, vol. 47, Issue 6, pp 383–397, Elsevier (2005)
10. Yatani, K. Chung, E., Jensen, C., Truong, K.N.: Understanding how and why open source contributors use diagrams in the development of Ubuntu, In: CHI '09 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 995–1004, ACM New York (2009)
11. Grady, B., Rumbaugh, J., Jacobson, I.: The Unified Modeling Language User Guide, pp.72, Addison Wesley (1998)
12. Chung, E., Jensen, C., Yatani, K., Kuechler, V., Truong, K.N.: Sketching and Drawing in the Design of Open Source Software, In: VLHCC '10 Proc. of the 2010 IEEE Symposium on Visual Languages and Human-Centric Computing, pp. 195–202, IEEE Computer Society, USA (2010)

Exploring Costs and Benefits of Using UML on Maintenance: Preliminary Findings of a Case Study in a Large IT Department

Ana M. Fernández-Sáez[1], Michel R.V. Chaudron[2], Marcela Genero[1]

[1]ALARCOS Research Group, Instituto de Tecnologías y Sistemas de Información,
University of Castilla-La Mancha, Spain
ana.fernandez@alarcosqualitycenter.com,
Marcela.Genero@uclm.es

[2] Joint Computer Science and Engineering Department,
Chalmers University of Technology & University of Gothenburg, Sweden
chaudron@chalmers.se

Abstract. UML has become the de-facto standard for graphical modelling of software. One source of resistance to model-based development in software organizations is the perception that the use of UML is not cost-effective. It is important to study what costs and benefits are experienced in industrial use, and in what context. In this paper we pay special attention to the maintenance phase, because maintenance consumes a significant part of software project resources. This paper describes a case study in an industrial context: the software department of a large multinational company. This case study presents qualitative analysis based on 2 0 out of 36 interviews performed with employees who played different roles in the company and provided different views about the use of UML. The results revealed that the investment needed for using UML in a company is relatively small and that it is mostly related to tooling and training. The principal use of UML diagrams is communication. The use of UML diagrams is also found to be related to fewer software defects. The costs of UML use should not be considered as a high investment. The paybacks of using UML are a better understanding of the problem domain, improved communication, reduction of software defects, improvement in software quality or reduction of software maintenance effort.

Keywords: UML, Software Maintenance, Modelling Languages, Case Study

1 Introduction

Modelling is a common aspect of effective software engineering, and UML is the de-facto standard notation for this. How to do software modelling effectively is still an open question. Given that a large portion of software development effort is spent on software maintenance [1], it is important to understand the impact of software modelling on software maintenance. In this paper, the term "*maintenance*" refers to those projects that modify or correct existing systems instead of creating new ones, i.e., the

focus is on repairing bugs and on creating new releases. In this study we explicitly aim to elicit factors related to the costs of using modelling, thus adding fresh findings to the hitherto scarce evidence on payoffs and costs of software modelling.

The principal goal of our research is to find out what industrial software professionals perceive as costs and benefits of software modelling, with special attention to software maintenance tasks. We focus our attention particularly on UML as a specific modelling language, because it is widely used in industry[2, 3]. In this paper we present empirical evidence obtained in the IT department of a large multinational company. This evidence was collected over a 12-month period in 2012.

Using the Goal-Question-Metrics template, we can formulate the goal of this study as follows: "***Analyze*** *the use of UML modelling* ***for the purpose of*** *investigating its costs and benefits,* ***with respect to*** *software maintenance tasks,* ***from the perspective of*** *the researcher,* ***in the context of*** *a large IT department".*

We wish to investigate whether the investment in UML is justified by benefits in software maintenance projects, such as improved productivity and improved product quality. We define the following research questions:

RQ1) What is the cost of using UML in software maintenance projects?

RQ2) What is the payback of using UML in software maintenance projects?

This paper is organized as follows. Section 2 presents the related work. Section 3 describes the case study and how it was designed. The results obtained are set out in Section 4, whilst the summary is provided in Section 5. Finally, Section 6 outlines our main conclusions and future work.

2 Related Work

After carrying out a Systematic Literature Review (SLR) [4] and later extending the search period till August 2013, we found 6 experiments related to the use of UML on the maintenance of source code. Only 2 experiments, using professionals as subjects, were discovered [5, 6], which concluded that the correctness or quality of the maintenance of the code is improved when UML diagrams are available, although the time of maintenance is not influenced. Related to the results obtained in academic environments with students, the results of Scanniello et al. [7] revealed that the availability of UML diagrams produced in the design phase positively influence the performance of maintenance tasks. But on the other hand, the presence of UML analysis diagrams does not show a clear influence on the understandability and modifiability of the source code[8]. This means that the phase in which the diagrams are created is an influential factor. But, is that difference based on the Level of Detail (LoD) presented in the diagrams? It seems that a higher LoD UML diagram improves the understanding and modifiability of source code compared to lower LoD UML diagrams, but the differences are not conclusive [9]. Focusing on the origin of the UML diagrams, in [10] we found that there is a clear preference for human-created diagrams (built during the development phase) over those generated using automatic reverse engineering tools, because they reduce the reading problems. The difference in performance is not significant, however.

The pattern that emerges from the results of these experiments is that, under controlled conditions, both students and professionals benefit to some extent from the use of UML in software maintenance. An important issue is to study if these results also hold in an industrial environment under real conditions. Pursuing this goal, we carried out the case study described in this paper.

3 Case Study Design and Execution

In this section, we discuss underlying aspects of the case study, following the suggestions provided in the literature for that purpose[11].

3.1 Specific Research Questions

It is difficult to measure the payback and costs of the use of UML precisely, because there is much noise in project administrations. We chose to aim for qualitative findings by performing interviews with different roles (software engineers, testers, developers, etc.). We broke down the research question further into the following:

1. What are the costs related to UML tooling? This question is related to RQ1.
2. What are the costs related to UML training? This question is related to RQ1.
3. What is the impact of UML diagrams on software maintainers' understanding and product quality? This question is related to RQ2.

3.2 Case and Subject Selection

For our case study we obtained data in an IT department of a multinational company. The IT department has between 800-1000 employees. In this department most projects are mainly of a software maintenance character. Following the classification of Yin[12], our study is a single, embedded case study. Our units of analysis are the different roles.

3.3 Data Collection Procedures

To obtain data about the use of UML during maintenance tasks we used two sources:

- **Department shared project files:** The IT department has a file server in which all the relevant documentation of the department and the projects is shared. Through these shared files the maintenance projects shares the project documentation and relevant documentation of the IT department.
- **Company personnel**: The researcher himself, as a temporary member of the organization and in the capacity of research intern, had direct access to the company staff and, in particular, to the people involved with the maintenance projects.

Using the first source, we obtained the quantitative data related to the investment carried out by the company for the introduction or improvement of UML modelling.

We also obtained qualitative data by interviewing personnel. We used semi-structured interviews[1] where the interviews are "guided conversations"[13]. The interviews are standardized, in the sense that each interviewee is asked similar questions, yet they are also open-ended, in that there is ample room for interviewees to elaborate.

3.4 Case Study Execution and Analysis Procedure

We performed 36 interviews of about one hour each, which were recorded and transcribed. We analysed each transcription, highlighting the important and surprising statements, using the NVivo tool. After that, we coded the statements and grouped them under more general themes. The interviews were performed with people of different roles, to obtain different points of view. The interviewee roles include: project managers, information analysts, project architects, technical lead, programmers or application developers, test engineers, delivery leads, SCRUM masters, system analysts.

4 Results

In this section we present the highlights from the findings of the study, based on the analysis of 20 of the 36 interviews. However, we already saw saturation of findings; hence we do not expect many new findings from fresh analysis.

4.1 What Are the Costs Related to UML Tooling?

We made an inventory of the tools in use in the company: Visio (15% of people using a modelling tool), Bizz Design Architect (5%) and Sparxs Enterprise Architect (80%), taking into account that one person might use more than one tool. The prices of licenses of these tools are between 135€ and 160€; a total of 150 licenses were needed in an IT department of 800-1000 employees. In addition, an amount of between 4,000€ and 6,500€ per year was paid as maintenance costs related to the use of the tools.

Although the tools used are part of the "expensive range" of tools, their costs are very small, relatively, compared to the yearly budget (mostly in manpower) of software maintenance projects. Moreover, the costs of tooling are fixed and can be paid off fast.

4.2 What Are the Costs Related to UML Training?

To answer this question, we used historical data provided by the person who manages internal/external training and courses for employees at the company; this data was from 2006 to May 2012. We selected those courses which were related to training on UML and separated them from other related topics (like Object Orientation, RUP,

[1] The interview questions can be found at: http://alarcos.esi.uclm.es/download/list-of-questions.pdf.

etc.), but sometimes those topics are taught together. Those courses usually take one week (40 hours approximately), and they do not have a learning test at the end of them.

The total amount of money spent by the company in UML adds up to 24,313€ in a period of 6 and a half years (which is approximately 3,750€ per year). Again, as for tooling, this amount is small, compared to the total budget of the department.

4.3 What Is the Impact of UML Diagrams on Software Maintainers' Understanding and Product Quality?

To answer this question, we performed interviews with different people involved in software maintenance projects. We present the results grouped by topic in the following subsections. The percentages presented below indicate the percentage of interviewees that mention this term/topic.

UML Usage.

The UML diagrams which the interviewees mentioned that they usually use during maintenance are the following: sequence diagrams (80% of interviewees), class diagrams (60%), activity and use case diagrams (50%), deployment diagrams (40%), component diagrams (30%) and collaboration diagrams (10%). These diagrams are used during the whole maintenance process, from the requirements specification starting with the design of use case diagrams, to the deployment of the system maintained in the operation environment using the deployment diagrams.

Purpose of Use of UML.

One of the questions during the interview was: "*Why do you use UML diagrams? / For what purpose is UML modelling used?*" The answers to these questions were varied. The majority of people use UML as a communication tool (22%). This communication can be between team members, including stakeholders (8%), or members of other teams (5%). UML is also used to communicate the current situation to newcomers to the project (7%). The broad use of UML as a representation for communication might be due to its being a standard notation, and also because it is well-known, both by professionals and recent graduates. At the same time, people recognize that UML diagrams are used to complement verbal communication (face to face or written), but not to replace it: "*[…] UML helps to improve the communication, but it doesn´t replace it […]*".

The next most common uses of UML diagrams are for: enhancing people's own understanding of the system under maintenance (8%), analysing risks (7%) and guiding testing (7%). Less-often mentioned are possible uses for: getting an overview (5%) or guiding implementation (5%).

Uses that were mentioned, but only rarely (2-3%), include: documenting, following the mandatory process, justifying costs, planning, supporting maintenance, determining responsibilities for success (offshore team), monitoring implementation, professional way of developing, or showing progress.

Finally, we should remark that some possible purposes which we expected to find were not actually mentioned by any of the interviewees, like certification, deployment, generation of implementation, knowledge transfer or reasoning about design.

Cost of Using UML.

We also asked the interviewees about the possible cost factors or investment related to the use of a modelling notation like UML in a software maintenance company: "*What cost factors are related to using UML modelling in your work?*"

Table 1 shows the responses to this question. The majority of those interviewed consider training as an important investment. This might be due to a fear of their own poor understanding of UML. Another investment which is often mentioned by interviewees is the cost of migration of the current situation to the new one, especially in the documentation. Formally speaking, this is related more to the introduction of UML than to the use of UML, yet it is potentially a major investment. Most comments related to migration came from people who are currently working on non-UML projects, and who would like to introduce it, but they consider the migration of the documentation to be an impassable hurdle.

Table 1. Cost factors related to the use of UML.

Cost factor	% references
Training	33%
on UML notation	22%
on modelling tool	5%
Migration	28%
Change of people's mind	11%
Tooling	11%
Central governance	5%
Learning curve	5%
Change of process	5%

Advantages and Disadvantages of UML.

We also asked the interviewees about the perceived advantages and disadvantages of the use of UML diagrams: "*Do you think UML has advantages? What are these? And disadvantages?*" The results are shown in Table 2.

Note that "high level of abstraction" is mentioned as an advantage and a disadvantage at the same time. This may be because architects feel abstraction is beneficial, but developers need diagrams which are closer to the source code.

We should take into account that the majority of the advantages commented, especially those related to the UML characteristics, are not benefits in themselves. They can, however, be considered as benefits in comparison with other modelling languages.

Some of the disadvantages mentioned (like "No semantics", "Unclear syntactics", "Difficulties in understanding the notation") might be caused by a poor understanding of UML diagrams. This problem could be solved by providing training in UML to users who do not feel comfortable with employing it.

Table 2. Advantages and disadvantages of UML.

Advantages	Disadvantages
Related to UML characteristics	
High level of abstraction High suitability for designing OO systems Shows different points of view Standardized	Not executable No/Unclear Semantics Freedom in styles - naming - layering... High level of abstraction Lack of user's point of view Low capability of designing SOA No enforcement for separation of what and how
Related to UML usage	
Helps to clarify procedures Helps in structuring the way of modelling Improves documentation Is a common language - world acceptance Is the only modelling language learnt properly Reduces misunderstandings/ gaps in offshoring	Difficulties in understanding the notation Difficulties modelling complex things Not enough expressiveness

UML Usage and the Quality of Software.

We asked the interviewees about the quality of the final product and its relationship with the use of UML diagrams: "*Do you think UML helps to improve the quality of the final product? How?*"

In this case interviewees considered quality of source code related to performing correct testing and obtaining positive results from it; i.e., obtaining a source code aligned with requirements and design: *"[…] Quality is the result of checking the result also, so UML is your reference of what this should be, but you have to check if the code that is delivered is in fact aligned with your UML diagram. […]"*

Employees of projects which are not using UML diagrams commonly believe that the presence/absence of diagrams is related to high/low quality of documentation, respectively. It is very important to note that there is universal agreement amongst all interviewees that the use of UML improves the software quality (100%).

In relation to software quality, we also asked the interviewees about the possible relationship between the use of UML diagrams and the presence of defects in the code of the system: "*Do you think that the use of modelling introduces errors?*"

17% of the interviewees considered that UML usage reduces the introduction of defects in the code of the system, i.e., prevents defects, while 8% believed that UML increases them. 8% of those interviewed think that there is no relation between software defects and UML in itself; the defects are caused by an incorrect solution, but UML is not the problem. Almost half of the interviewees (42%) are of the opinion that the use of UML is helpful when we need to find the cause of a problem in the source code.

Standardization.

We asked the interviewees about standardization in ways of working. In this case, we focussed on those standards used to document the system and the activity of diagramming. Only 10% of the interviewees considered that there is excessive standardisation, while 37% believed that there is a lack of standardization. These last respondents felt a need for more standardization related to the following:

- **Naming**: naming conventions for classes, attributes, etc. in code and diagrams.
- **Layering**: it is not clear what the recommended layering of the system is.
- **Style**: There are a lot of issues related to the style of diagramming (and subsequently of coding) which are not clear.
- **Level of detail**: it is not clear at what level of detail systems should be modelled.

Independently of their opinion on the presence of standards at the company, most of those interviewed (53%) agreed that there is a lack of conformance to the standards. Mechanisms to incentivise the correct use of standards should thus be introduced: *"If you let people choose, you lose all your advantages. So, yes, force them."*

5 Threats to Validity

We must consider certain issues which may threaten the validity of the case study[11]:

- **Internal validity**: The age, education, role or experience of the interviewees might be influential factors in being for, or against, the use of UML. This factor will be analysed in future work.
- **External validity**: the sample of the case study might be a threat to the validity of this study, although the sampling process was as randomized as possible. The generalization of the results might be extended to cases which have common characteristics.
- **Construct validity**: the transcript of interviews and observations were sent back to the interviewees to enable correction of raw data. Apart from that, analyses were presented to them and to the internal research supervisor, in order to maintain their trust in the research.
- **Reliability**: the chain of evidence from the interviews and documentation analyzed through to the synthesized evidence was maintained using a word-for-word transcription (so as not to reach mistaken interpretation while the analysis was being undertaken; this analysis took a long time to carry out). Tools were also used during the analysis of the data. In addition, randomized pieces of the analysis were discussed by the researchers, so that they could verify and reach an agreement on them.

6 Conclusions and Future Work

This work aimed to discover the costs and benefits of using UML modelling in the setting of maintenance-intensive software development.

In an effort to answer the first two research questions of this study, we have reported on the costs of use and introduction of UML modelling. In the context of a large IT department these costs related to tooling and training can be considered relatively small. In addition, the cost of building the UML documents is considered as low by the majority of interviewees. The cost of maintenance of the UML documents is zero, due to the fact that in the majority of cases the UML documents are not synchronized with the updates performed in the source code. The payback of UML use is very difficult to measure, because one of the main benefits is the improvement of communication between stakeholders. That is why we decided to investigate the impact of UML diagrams on software maintainers' understanding and product quality as a third research question. We therefore asked employees for their subjective opinion of the use of UML diagrams, as well as about their benefits. As on all issues, there are those in favour and those against the use of UML, but we detected more people in favour of using it. Proponents of modelling could be found within project architects, developers and maintenance engineers. Opponents to modelling could be found in Agile formation and people who are less familiar with UML. We speculate that people who are opposed to UML modelling are individuals who have been working at the company for a very long time, who are used to working in a certain way and thus are fearful of change.

Several benefits have been reported regarding the use of UML: better understanding of the problem domain, improved communication, reduction of SW defects, improvement in quality or reduction of software maintenance effort. We would recommend strengthening the benefits mentioned in the employees' ideas, also introducing the rest of the possible advantages to them (like reducing rework, improving the requirements, a better understanding of the solution space, etc.).

As part of the analysis of the costs and paybacks of the modelling during maintenance, several additional issues were detected, which should be dealt with in the company in the quest to improve the maintenance process. There is a need for standardization – which should focus in particular on the style of modelling: 1) Naming and layering conventions should be defined; and 2) The level of detail which should be presented on diagrams should be defined.

A very important issue which must be improved is the need to keep diagrams and the documentation in-synch with source code, representing on these all the changes performed in the system. In order to keep the diagrams updated, we recommend the use of a version management tool of diagrams. In relation to this topic, we observed that the process and responsibility for updating the documentation is often not clearly assigned. Finally, we recommend incentivizing or giving training on the long term benefits of using modelling languages (especially UML) to those subjects who do not know them and who cannot feel there is any possible benefit from a change in the process. People should also be incentivized regarding the benefits of maintaining the documentation.

Nevertheless, we will continue analysing the remaining interviews, in order to corroborate the results obtained. The analysis of the documentation of each project and its relation with employees' opinion will also be done as part of future work.

Acknowledgements

This research has been funded by the GEODAS-BC project (Ministerio de Economía y Competitividad and Fondo Europeo de Desarrollo Regional FEDER, TIN2012-37493-C03-01).

References

1. Pressman, R.S.: Software engineering: a practitioners approach. McGraw Hill (2005).
2. Dobing, B., Parsons, J.: How UML is used. Communications of the ACM. 49(5), 109–113 (2006).
3. Scanniello, G., Gravino, C., Tortora, G.: Investigating the Role of UML in the Software Modeling and Maintenance - A Preliminary Industrial Survey. Presented at the International Conference on Enterprise Information Systems (2010).
4. Fernández-Sáez, A.M., Genero, M., Chaudron, M.R.V.: Empirical studies concerning the maintenance of UML diagrams and their use in the maintenance of code: A systematic mapping study. Information and Software Technology. 55(7), 1119–1142 (2013).
5. Dzidek, W.J., Arisholm, E., Briand, L.C.: A realistic empirical evaluation of the costs and benefits of UML in software maintenance. IEEE Transactions on Software Engineering. 34(3), 407–432 (2008).
6. Arisholm, E., Briand, L.C., Hove, S.E., Labiche, Y.: The Impact of UML Documentation on Software Maintenance: An Experimental Evaluation. IEEE Transaction on Software Engineering. 32(6), 365–381 (2006).
7. Scanniello, G., Gravino, C., Tortora, G.: Does the Combined use of Class and Sequence Diagrams Improve the Source Code Comprehension? Results from a Controlled Experiment. Presented at the Experiences and Empirical Studies in Software Modelling Workshop (2012).
8. Scanniello, G., Gravino, C., Genero, M., Cruz-Lemus, J.A., Tortora, G.: On the Impact of UML Analysis Models on Source Code Comprehensibility and Modifiability. ACM Transactions On Software Engineering And Methodology (In press) (2013).
9. Fernández-Sáez, A.M., Genero, M., Chaudron, M.R.V.: Does the Level of Detail of UML Models Affect the Maintainability of Source Code? Presented at the Experiences and Empirical Studies in Software Modelling Workshop (2012).
10. Fernández-Sáez, A.M., Chaudron, M.R.V., Genero, M., Ramos, I.: Are forward designed or reverse-engineered UML diagrams more helpful for code maintenance?: a controlled experiment. Presented at the International Conference on Evaluation and Assessment in Software Engineering (2013).
11. Runeson, P., Höst, M., Rainer, A., Regnell, B.: Case Study Research in Software Engineering: Guidelines and Examples. Empirical Software Engineering, 14, 131-164 (2012).
12. Yin, R.K.: Case Study Research: Design and Methods. SAGE Publications (2002).
13. McNamara, C.: General guidelines for conducting interviews. Authenticity Consulting, LLC, Minneapolis, MN (1999).

Towards Reconstructing Architectural Models of Software Tools by Runtime Analysis

Ian Peake, Jan Olaf Blech, Lasith Fernando

RMIT University, Melbourne, Australia
{ian.peake, janolaf.blech,lasith.fernando}@rmit.edu.au

Abstract. We present a method and initial results on reverse engineering the architecture of monolithic software systems. Our approach is based on analysis of system binaries resulting in a series of models, which are successively refined into a component structure. Our approach comprises the following steps: 1) instrumentation of existing binaries for dynamically generating execution traces at runtime and connected analysis, 2) static inspection of binaries, 3) interpretation using domain knowledge, and 4) identifying component boundaries using software clustering. We motivate a generic method which covers a large class of software systems, and evaluate our method on concrete software tools for industrial automation systems development, focusing on Intel x86 and Microsoft Windows-compatible applications.

1 Introduction

We present an architectural reverse engineering approach. Instead of solely analysing binaries statically, we perform analysis at runtime thereby taking into account runtime dependencies between entities. This detailed dependency information denotes an abstract system model. Clustering is used to identify candidates for a component-based software architecture view suitable for human understanding. Additional information from binary inspection and domain specific knowledge is used to select an architectural system model. Furthermore, in the experimental part of this paper, we apply our method to tools for distributed industrial automation programmable logic control (PLC) specification and configuration. Such tools typically support specifications compliant with the IEC 61131–3 or IEC 61499 standards (e.g. CoDeSys [6] and 4DIAC [8], respectively). Restriction to a particular domain gives us information for calibrating our analysis. In this paper our novel contributions are as follows: 1) A suggested architecture reverse engineering method based on runtime instrumentation, automated clustering, hand-inspection of binaries, and domain knowledge. 2) Tailoring this method for Intel x86, in particular Microsoft Windows. 3) A case-study applying our method to PLC specification and configuration tools.

Related Work Published work on architecture reconstruction and related reverse engineering tasks focussing on derivation of component candidates and inter-dependencies is covered in existing surveys and overview papers [5, 15, 3]. Two main directions are 1) based on analysis of source code and 2) based on the analyses or execution of system binaries. In [3] a taxonomy of reverse engineering techniques by classifying according to

the artefacts used, and whether analysis is static (based on syntactic analysis of source or executable) or dynamic (based on running, observing and/or animating the system itself) is presented. We focus on runtime analysis for architecture derivation (also called dynamic analysis) [7]. DiscoTect [16, 17] is a framework that observes running systems to reconstruct their architectures. A key feature of DiscoTect is its flexibility to cope with a range of high architectural styles and a range of possible realizations in implementations. DiscoTect uses a language: DiscoSTEP to define mappings interpreting low level system events as more abstract architectural operations, which are formally defined as coloured Petri Nets. The authors note that such mappings must be provided by experts with correct domain knowledge. Reconstructing software architecture from execution traces requires the analysis of the execution traces and the identification of potential components. Combining potential *component candidates* into disjunct sets denoting suggestions for aggregation of components is known as clustering and is an important step for gaining suggestions on the original and potential future architectures. The field of clustering for software components has been studied by several authors including [10] featuring a proposition, [12] featuring the analysis of source code for component detection, [9] studying clustering in the context of software evolution. In this work we are using the Pin tool [4] for binary instrumentation and tracing hints about architecture. Other well known tools comprise the more heavy weight [13] tool which does not have native Windows support, but offers a wider range of instrumentation possibilities potentially resulting in slower code.

2 Our Approach

Our method for collecting runtime based architecture information has these steps:1) We instrument an existing tool such that dynamically loaded libraries and control flow events are tracked and collated as execution traces. These traces contain information (e.g. available methods) for dynamically loaded libraries, as well as the order of method calls. Instrumentation is done on a binary level. 2) The instrumented tool is run and execution traces are generated. A user interacts with the tool (e.g. editing, simulating, compiling). All dynamically loaded libraries and method calls (traced by memory address) and time of invocation are traced. The generated execution traces are further processed and abstracted. This involves the resolution of traced memory addresses to *primitives* such as methods, objects or executables. Calls between methods denote a graph. Here, each primitive corresponds to a node and the number of distinct caller/callee combinations in the execution trace is annoted as a weight on a directed edge. We cluster primitives into candidate components using the LIMBO algorithm (see below). This gives first candidates for a component architecture. Final clustering is based on interactions between methods, existing dll structure, analysis of names and knowledge about reference architectures. 3) The generated data is interpreted by using information from binaries and the domain, to derive information about the underlying architecture of the tool. Manual binary inspection and domain knowledge are used to complete reconstruction.Several tasks are carried out for the runtime-based analysis part of our method:

Usage Scenarios for Runtime Based Evaluation We evaluate our tools with the help of usage scenarios. These are sequences of user interactions with tools. The component

interactions are then extracted from the generated execution trace in order to gain hints on architectural details. The idea is to invoke the distinct components of a tool by user interaction. For example a user may trigger a compilation at a certain time and the execution trace may show the loading of distinct libraries and the invocation of the desired methods. We can also compare interaction sequences in order to see if different tools have a similar way of interacting e.g. with a compilation component.

Evaluating Execution Traces, Clustering and Component Candidate Identification We aim to generate a graphical view showing a few high-level components and their interactions. In Windows and similar environments components are most often associated with distinct executables and libraries (or possibly packages), and their inter-dependencies (associated with dynamic linking, import or transfer of control between their respective methods). However a high-level view at such a level is often inappropriate because the view has either too many or too few executables. We therefore tried to identify high level component candidates by clustering groups of other programming language-specific notions such as classes/object or method. We will use the term *primitives* to refer to low-level component categories selected for clustering.

Software clustering is a long-standing and commonly used method for imposing abstract, high level structure on an over-detailed view of primitives and their relationships. For software, a set of low-level components is typically clustered on the basis of properties such as which other components they call, authorship, or location in source directories. As shown, clustering may be thought of as partitioning a collection of objects based on the similarity of their properties. Typically clustering is based on static analysis, here, we are using clustering based on the dynamic call structure between primitives observed at runtime.

Figures 1 (a) and (b) depict the clustering for a usage scenario in the open source tool PLCEdit [14]. Both take the dynamic call structure between primitives into account and are generated from the same execution trace file. Primitives of each cluster are listed in nodes (boxes). The number of calls between primitives are provided as labels on the edges. The main call direction is given first. Calls in the opposite direction are in parentheses. The figures exemplify that based on the same execution trace files there are different possible ways to depict abstract system structure. Arguably, good component structures are selected based on domain specific knowledge.

We use the LIMBO clustering algorithm [2]. LIMBO is based on a generic method called Agglomerative Information Bottleneck (AIB). It has been used for the analysis of large systems across scientific disciplines. LIMBO and the underlying AIB method are generic in the sense that they operate fundamentally on a set of objects O, a set of attributes A and relation $R \subseteq O \times A$ with non-negative real number weighting $w : O \times A \rightarrow \mathbb{R}^+ \cup \perp$. In our approach we represent primitives as follows. Each primitive is modelled both by an object in O and an attribute in A. The weighting w reflects the number of different ways an object o in O calls a different object o' in A. R and w are constructed from the execution traces in an application-dependent way. LIMBO uses a generic information-theoretic approach as its basis for clustering. First, weights are modified via a suitable weighting transformation such as TF.IDF, which transforms weights according their significance (the more rarely held an attribute A is overall by all objects, and the more frequently by some given object O, the more sig-

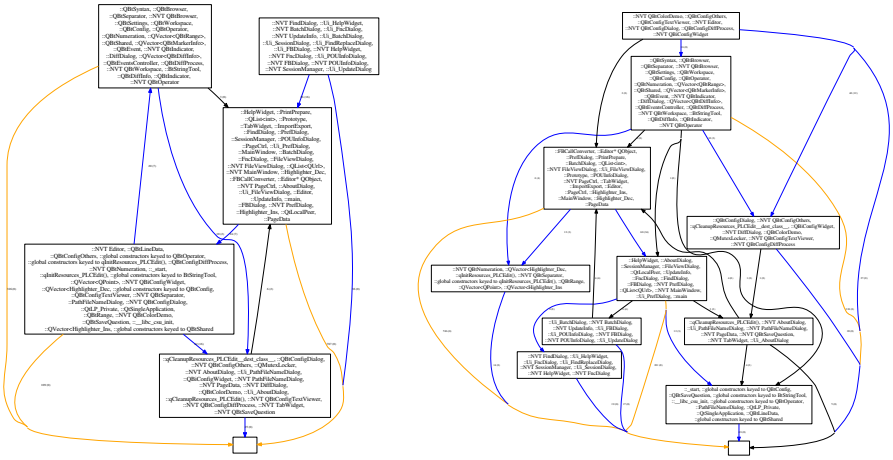

Fig. 1: Example control flow graphs for 5 and 10 components

nificant, thus heavily weighted, A is for O.) Next, the new weights are converted to probabilities such that the sum of all weights per object is 1. Finally, LIMBO attempts to compress its representation of R by iteratively merging the *closest* pair of objects and generating a new relation R' which approximates R under merging. The closest pair is the one for which merging minimises information loss in R'. LIMBO's genericity enables it to support both "structural" and "non-structural" attributes. Structural attributes reflect program dependence structure as described above. In our work so far clustering is purely on a structural basis. Non-structural attributes refer to the general case and cover properties such as a time stamp or authorship. There is ambiguity about whether it is best to generate structural attributes by interpreting the primitive call graph as directed or undirected—That is, whether two primitives which call each other have the same value in both directions (sum of the number of ways they can call each other) or possibly-distinct values. In our work the call graph is interpreted as undirected.

Additional Static and Domain Specific Information Binaries like .dll files can encapsulate multiple components and provide hints on development history. Names and size of components can indicate usage. Binaries can contain method names and plain text that hint on component functionality. A major source of knowledge in our reverse engineering method is the PLC development domain. For example we know what types of components to expect. We started with the following expected components: *Source and target code storage* manage the modeling, storage and exchange of source and target specification models and code by using a file system or a database. *Compilers, Analyzers and Simulators* parse specification models and perform operations on them, like generating target code, interacting with a GUI component in order to visualize behaviour or properties. *Editors* manage the editing of models by the user. *License Management* and other miscellaneous functionality can be realized as a separate component e.g. that may interact with a third party license server. The *GUI* provides a user

interface. It does not have to be realized as a separate component inside the tool, since existing GUI frameworks can be used.

3 Analysis and Evaluation

Instrumentation of binaries is done by using the Intel Pin tool [4]. We instrument the binaries of our analyzed tools to extract: (i) A list of the loaded binaries and the names of the methods (called routines) inside these binaries, if available, including their memory addresses. (ii) A list of control flow operations that occurred during the execution of the tool, and in particular the source and destination addresses.

Case Study Tools We have used our method for analysing the architecture of a mix of proprietary and open source tools. Tools are designed for performing at least some of the following operations for the development of PLC software: 1) Editing PLC specification models, 2) Saving and loading of PLC specification models, 3) Analysing and compiling PLC specification models, 4) Simulating PLC specification models. We initially expected that this functionality is provided by distinct software components as described in the previous section. Open source systems considered included PLCEdit [14], Beremiz [1] and MATIEC [11]. Of these, PLCEdit and MATIEC (also a Beremiz component) were immediately suitable, consisting of plain binary executables. Beremiz is written in Python and thus the Pin-based method is not immediately suitable.

Example Runs An Example usage scenario (Section 2) consists of the steps: Start tool; Create new project file; Add ladder diagram; Invoke editor; Add coil (lamp) and contact (switch) to ladder diagram, add connections; Save project; Compile and check project; Close project; Start simulation of saved project; Close tool.

Evaluation and Improvements The method was applied to different PLC development tools. Execution of the usage scenarios was done manually, while the processing of the execution traces was done automatically to generate models – one single model for each usage scenario and number of desired components – comprising component candidates and their interactions. Clustering based on dynamically linked libraries and executables did not always provide the right granularity, since several major components are typically encapsulated in a main executable. Determining the begin and end of entities like methods or classes in the binaries as a basis for clustering was sometimes possible. In some cases e.g. due to the use of different programming languages, additional information on the location of entities for the basis of clustering was provided by the tool developers and used by us. For example for some applications while symbol table information is not available in the executable, a ".map" file provides similar information for debugging purposes. There are a number of possible reasons why a clustering may not reflect a system's true architecture, for example there may be insufficient data in the run time call graph, or architectural anti-patterns may be present. It may be desirable to associate each component with a meaningful name or feature. As discussed in clustering literature, this depends on understanding what abstractions (e.g. aspects) are semantically common to all objects of a component, or the principle abstraction of

the component, which can be difficult. Currently all attributes are structural, derived from the Pin call graph, however the graph is created by exercising tools using just a few use case scenarios. There is scope to assign so-called non structural attributes, that is, properties other than call relationships on which clustering could be based, possibly based on manual assessment and with input from domain experts. These could pertain to specific features or aspects such as GUI or safety. For example if several objects are clearly GUI components, an additional GUI attribute could be assigned to those objects and taken into account during clustering. There are existing aspect mining approaches in the literature which may be applicable or adaptable to this purpose.

The LIMBO algorithm was implemented by us in few hundred lines of Python. This is supported by additional scripts which process output from our pin plugin.

References

1. Beremiz IDE. Version 1.1 RC3. Downloaded from `beremiz.org` (July 2013)
2. Periklis Andritsos and Vassilios Tzerpos. Information-Theoretic Software Clustering. In IEEE Trans. on Software Eng., 31(2): 150-165 (2005)
3. Gerardo Canfora, Massimiliano Di Penta, Luigi Cerulo: Achievements and challenges in software reverse engineering. Commun. ACM 54(4): 142-151 (2011)
4. Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, Kim Hazelwood. Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation. Programming Language Design and Implementation (PLDI), Chicago, IL (June 2005)
5. Elliot J. Chikofsky, James H. Cross II: Reverse Engineering and Design Recovery:A Taxonomy. IEEE Software 7(1): 13-17 (1990)
6. CoDeSys — industrial IEC 61131-3 PLC programming: `www.codesys.com`
7. Bas Cornelissen, Andy Zaidman, Arie van Deursen, Leon Moonen, Rainer Koschke: A Systematic Survey of Program Comprehension through Dynamic Analysis. IEEE Trans. Software Eng. 35(5): 684-702 (2009)
8. 4DIAC IDE. Version 1.3: `fordiac.org` (Accessed July 2013)
9. Rainer Koschke. Atomic architectural component recovery for program understanding and evolution. Software Maintenance (2002).
10. Chung-Horng Lung. Software Architecture Recovery and Restructuring through Clustering Techniques. 3rd International Software Architecture Workshop (ISAW): 101-104 (1998)
11. MATIEC compiler. Source from `bitbucket.org/mjsousa/matiec` (July 2013)
12. Brian S. Mitchell, Spiros Mancoridis. Comparing the decompositions produced by software clustering algorithms using similarity measurements. Software Maintenance (2001)
13. Nicholas Nethercote and Julian Seward. Valgrind: a framework for heavyweight dynamic binary instrumentation. ACM SIGPLAN conference on Programming language design and implementation (2007)
14. PLCEdit Editor. Version 2.1.1. Downloaded from `www.plcedit.org` (July 2013)
15. Damien Pollet, Stéphane Ducasse, Loïc Poyet, Ilham Alloui, Sorana Cîmpan, Hérve Verjus: Towards A Process-Oriented Software Architecture Reconstruction Taxonomy. Conference on Software Maintenance and Reengineering: 137-148 (2007)
16. Hong Yan, David Garlan, Bradley R. Schmerl, Jonathan Aldrich, Rick Kazman. DiscoTect: A System for Discovering Architectures from Running Systems. International Conference on Software Engineering: 470-479 (2004)
17. Bradley Schmerl, Jonathan Aldrich, David Garlan, Rick Kazman and Hong Yan. Discovering Architectures from Running Systems. IEEE Trans. Software Eng. 32(7): 454-466 (2006)

Industrial Adoption of Automatically Extracted GUI Models for Testing

Pekka Aho[1,2] pekka.aho@vtt.fi, Matias Suarez[3]
matias.suarez@f-secure.com, Teemu Kanstrén[1,4] teemu.kanstren@vtt.fi,
and Atif M. Memon[2] atif@cs.umd.edu

[1] VTT Technical Research Centre of Finland, Oulu, Finland
[2] University of Maryland, College Park, MD, USA
[3] F-Secure Ltd, Helsinki, Finland
[4] University of Toronto, Toronto, Canada

Abstract. Crafting the models for effective model-based testing (MBT) requires deep understanding of the problem domain and expertise on formal modeling, and creating the models manually from the scratch requires a significant amount of effort. When an existing system is being modeled and tested, there are various techniques to automate the process of producing the models based on the implementation. Especially graphical user interface (GUI) applications have been a good domain for reverse engineering and specification mining approaches, but the existing academic approaches have limitations and restrictions on the GUI applications that can be modeled, and none of them have been adopted by the industry for testing commercial software. Although using implementation based models in testing has restrictions and requires special consideration, the generated models can be used in automated testing and supporting various manual testing actions. In this paper we introduce an industrial approach and platform-independent Murphy tool set for automatically extracting state models for testing GUI applications.

1 Introduction

Model-based testing (MBT) is a technique of generating test cases from behavioral models of the system under test (SUT). The idea is to provide more cost-effective means for extensive testing of complex systems. Instead of manually writing a large set of test cases, a smaller set of test models are built to describe generally the behavior of the SUT and how it should be tested. A test generator tool is then used to automatically generate test cases from these models. There are several benefits, including easier test maintenance due to fewer artifacts to update, higher test coverage from the generated test cases, and documenting the SUT behavior in higher level models which helps in sharing the information and understanding the system [1]. However, crafting the models requires a great deal of expertise in formal modeling and a deep understanding of the problem domain. Constructing the models manually from the scratch requires also a significant amount of effort [2].

There are several approaches aiming to reduce the time required for designing the test models for MBT by automating some parts of the modeling process, such as creating models through reverse engineering or specification mining. Especially in the area of graphical user interface (GUI) software, there are promising academic approaches to automatically construct models based on observing an existing application and using the models for testing purposes, such as [3] [4] [5] [6]. Unfortunately most of these approaches have limitations and restrictions on the GUI applications that can be modeled, and so far none of them have been adopted by the industry to test commercial software products.

As the generated models are based on the behavior of the observed implementation, instead of the specifications or expected behavior, it is challenging to automatically generate meaningful test oracles. In most of the dynamic GUI reverse engineering approaches for testing, the test oracle is based on the observed behavior of an earlier version of the GUI application. Using this kind of test oracle, changes and inconsistent behavior of the GUI can be detected, but validation and verification against the specifications is problematic. Although using implementation based models in testing has restrictions and requires special consideration, the generated models can be used in automated testing and supporting various manual testing actions.

In this paper we introduce a platform-independent industrial approach and Murphy tool set for automatically extracting finite state machine (FSM) based models for testing GUI applications. The approach is based on observation and analysis of the GUI during automated interaction and execution of the application.

2 Background and Related Work

There are a few approaches using static analysis of the source code for automatically constructing models of the GUI software, such as [7], but the dynamic approaches that involve executing the GUI application and observing the application during the run-time are better suited for extracting the behavior of GUI applications [2].

Grilo et al. [2] describe a dynamic approach for reverse engineering GUI applications using a combination of manual and automated steps in the modeling process. The tool uses Microsoft UI automation library for the automated steps and the created model has to be manually validated and completed with the expected behavior. The final models are in Spec# format and can be used for model-based GUI testing (MBGT).

Memon et al. [8] have extensively published their research on GUI Ripping, a technique for dynamically reverse engineering models of GUI applications for test automation purposes. Memons team has implemented GUITAR tool set, a model-based system for automated GUI testing, to execute and observe Java GUI applications to generate models for MBGT. The main target of GUITAR tool set has been Java desktop applications, but it can also be used to model other GUI applications, such as web and android applications, to some extent.

Miao et al. [5] propose a finite-state machine (FSM) based GUI Test Automation Model (GUITAM). In GUITAM, a state of the GUI is modeled as a set of opened windows, GUI objects (widgets) of each window, properties of each object, and values of the properties. Events or GUI actions performed on the GUI may lead to state transitions and a transition in GUITAM is modeled with the starting state, the event or GUI action performed, and the resulting state. To reduce the amount of states into computationally feasible level, not all different property values are considered for distinguishing different states of GUITAM. The authors provide only a short introduction of the tool for automatically constructing the models and the tool is not publicly available.

Aho et al. [9] present GUI Driver, a dynamic reverse engineering tool for Java GUI applications. In [3] the authors introduce an iterative process of manually providing the valid input values into the GUI application and automatically improving the created models. They highlight the importance of increasing the level of GUI automation in order to include all parts of the GUI application in the created models. Unfortunately the tool is currently restricted to Java based GUI applications only.

GUITAR, GUI Driver and GUITAM model a GUI window in the same way: A window consists of widgets, properties of the widgets, and values of the properties. GUITAM and GUI Driver use a similar FSM-based GUI state model where the nodes are states of the GUI and the edges are user actions that trigger transitions between the states. In event flow graph (EFG) [8] or event interaction graph (EIG) [10], created by the GUITAR tool set, the nodes of the model are events or user actions, and the edges capture the flow of the GUI events.

Amalfitano et al. [6] have researched automated modeling and testing of Android applications. The approach is based on a tool that explores the application GUI by simulating real user events on the user interface and reconstructs a GUI tree model. The nodes of the tree represent individual user interfaces in the Android application, while edges describe event-based transitions between interfaces. The GUI exploration technique supports the automatic derivation of test cases that can be executed both in crash testing and regression testing processes.

3 Automated Extraction of GUI Models for Testing

F-Secure Ltd is a software company from Finland having both client and server side products related to safety and security, such as virus protection, including applications with GUI for the end users. F-Secure have developed a tool set called Murphy for automatically extracting models of GUI applications from the user interface (UI) flow, and using the created models for GUI testing.

Murphy dynamically analyses the GUI while automatically interacting with the application, as if it were an end user trying out all the possible user interactions, such as entering text in a text field, pressing a button or a link, selecting items or ticking checkboxes. The main idea is to traverse through all the possible states of the GUI application and automatically construct a finite state machine

(FSM) based model of the observed behavior during the execution, or as we call it, crawling the GUI.

The goal of the Murphy tool is not to find all the possible paths to reach a specific node, but to discover as many nodes as possible. The reason for this is that the GUI applications tend to have a very large number of paths between the nodes, making it impractical to try to reach a specific node again through a different path. Instead, with an appropriate level of abstraction, covering all the states of the GUI is more practical approach. However, it is possible to customize the scripts of Murphy tool to contemplate special cases, for example if the models are meant to be used for testing all the possible transitions between specific states.

To accomplish platform independency, Murphy uses various approaches called drivers for recognizing elements and windows of the GUI application. Among the already implemented drivers, one uses Windows APIs for UI element detection and enumeration, another uses a proprietary API developed by F-Secure for enumerating and querying windows and elements of the UI, and a third driver simulates the end user by cycling through the UI elements by pressing the 'tab' key and analyzing the changes in the screen to determine the elements and behavior of the GUI. The idea is to compare automatically taken screenshots to find the changing areas, such as the bounding rectangle of the selected element on the screen and the shape of the mouse cursor, and reason the structure and behavior of the GUI based on the clues that the GUI application offers for the end user.

Internally Murphy creates a directed graph to model the behavior of the GUI application. The nodes of the model are states of the GUI screen or window, and the edges are actions that the end user could perform in that specific state of the GUI, in a similar way as in [9] and [5]. An edge of the model could be for example pressing an OK button, selecting a specific item in a drop down box, or entering a predefined value in a text field. A node of the model, i.e. the state of a GUI application, is defined mainly by its appearance, excluding data values, such as texts in the text fields, selected values of drop down boxes or status of check boxes. In other words, a dialog that has an OK button enabled represents a different node than otherwise similar dialog with the OK button disabled, but a dialog would represent the same node regardless of the value in a text field of the dialog. During the UI crawling, screenshots of the GUI are automatically captured after each interaction, and they are used for visualization of the resulting graph; a picture of the GUI in that specific state is presenting the node in the graphical presentation of the model. The images are also used by one of the drivers for detecting changes in the UI and the interactions that are available for the end user.

Murphy provides generic UI crawling and window scrapping services as a library. The process of extracting the model is mostly automated and fully customizable, and Murphy provides hooks and callbacks for such customizations. The script that is used for invoking the generic UI crawling library can be modified with application specific rules, but often the generic UI crawler library is

sufficient for generating the models. For simple GUI applications, the invocation script will merely setup the initial state, such as start up the application to be modeled, and then invoke the generic UI crawler library. For more complex GUI applications, application specific modifications can be used for adjusting or correcting the behavior of the generic UI crawler library, for example adding extra edges to a node when all interactions were not properly recognized, instructing the crawler of the values to be used in certain text fields, removing edges from a node, or instructing the crawler not to visit specific edges of the specified nodes. For example, in a dialog for selecting the language for the installation of an application, the invocation script could be modified to instruct the UI crawler library to crawl only the English version of the UI flow. The model extraction would have defined the edge for selecting the installation language into the model, but the UI crawler would ignore it and select English.

In order to limit the scope of the UI crawler into the areas of interest, Murphy has the notion of boundary nodes. Boundary nodes are used for marking the nodes that Murphy should not go beyond during the UI crawling, for example when the GUI application launches a web browser and opens a web page, or when it is not feasible to crawl through the whole help system of the GUI application. Boundary nodes are manually specified into the invocation script. In our experience, it was also useful to add edges representing special actions with environment considerations, for example performing certain UI action when access to the network is not available, or when running low on memory. These special edges have to be manually inserted into the invocation script but in some cases they enriched the resulting model in useful ways.

The Murphy tool has been able to satisfactory extract most parts of the UI flow of the GUI applications of F-Secure with very little user intervention. The invocation scripts were usually between 10 and 200 lines of Python code and produced models capable of exercising in the range of 80% of the possible UI flows. It is important to notice, that most of the GUI applications used in experimenting the approach and Murphy tool set were flow based applications having a relatively low amount of the possible values that the end user may use as input into the system.

4 Discussion and Conclusion

In this paper we have introduced an approach and Murphy tool set for automatically extracting GUI models that can be used in testing GUI applications. Compared to the related approaches, the main advantage of Murphy tool set is that it does not restrict the modeled GUI application to a specific programming language or execution platform. To accomplish platform independency, Murphy uses various approaches called drivers for recognizing elements and windows of the different types of the modeled GUI applications.

We have promising preliminary experiences on using the generated models for testing commercial software products in industrial testing environment. It seems that using the automatically generated models for automating the GUI

testing would reduce the amount of hand written code related to GUI testing compared to manually written test scripts, which reduces the maintenance effort related to test code. Also, with the help of Murphy tool set, it is possible to use the models to support and reduce the effort required for manual GUI testing.

So far, we have used the approach and Murphy tool set only on desktop and web applications, but in future we plan to use the approach also on mobile applications.

Acknowledgment

This work was partially supported by grant number CNS-1205501 by the US National Science Foundation, and a part of ITEA2/ATAC project funded by the Finnish Funding Agency for Technology and Innovation TEKES.

References

1. M. Utting, and B. Legeard, *Practical Model-Based Testing: A Tool Approach*, Morgan Kaufmann Publishers Inc, San Francisco, CA, USA, 2006.
2. A.M.P. Grilo, A.C.R. Paiva, and J.P. Faria *Reverse Engineering of GUI Models for Testing*, 5th Iberian Conference on Information Systems and Technologies (CISTI), Santiago de Compostela, Spain, 2010.
3. P. Aho, N. Menz, and T. Raty, *Enhancing generated Java GUI models with valid test data*, 2011 IEEE Conference on Open Systems (ICOS 2011), 25-28 Sep 2011, Langawi, Malaysia.
4. A. M. Memon *An event-flow model of GUI-based applications for testing*, Software Testing, Verification and Reliability, Volume 17, Issue 3 (Sep 2007).
5. Y. Miao, and X. Yang *An FSM based GUI Test Automation Model*, 11th Int. Conf. Control, Automation, Robotics and Vision Singapore, 7-10th Dec 2010.
6. D. Amalfitano, A. R. Fasolino and P. Tramontana *A GUI Crawling-Based Technique for Android Mobile Application Testing*, 3rd Int. Workshop on Testing Techniques & Experimentation Benchmarks for Event-Driven Software, IEEE CS Press, 2011, pp. 252-261.
7. J.C. Silva, C. Silva, R.D. Gonalo, J. Saraiva, and J.C. Campos *The GUISurfer tool: towards a language independent approach to reverse engineering GUI code*, Proc. 2nd ACM SIGCHI symposium on Engineering interactive computing systems, Berlin, 2010, Germany, pp. 181-186
8. A. M. Memon, I. Banerjee, and A. Nagarajan *GUI ripping: reverse engineering of graphical user interfaces for testing*, Proc. 10th Working Conference on Reverse Engineering (WCRE'03). IEEE Comp Society, Washington DC, USA.
9. P. Aho, N. Menz, T. Raty, and I. Schieferdecker *Automated Java GUI Modeling for Model-Based Testing Purposes*, 8th Int. Conf. on Information Technology : New Generations (ITNG2011), April 11-13, 2011, Las Vegas, Nevada, USA.
10. Q. Xie, and A. M. Memon *Rapid crash testing for continuously evolving GUI-based software applications*, Proc. 21st IEEE Int. Conf. on Software Maintenance (ICSM'05), IEEE Computer Society, Washington DC, USA, 473-482.

What do Metamodels Really Look Like?

James R. Williams, Athanasios Zolotas, Nicholas Matragkas,
Louis M. Rose, Dimitios S. Kolovos, Richard F. Paige, and Fiona A. C. Polack

Department of Computer Science, University of York, UK
{james.r.williams,amz502,firstname.lastname}@york.ac.uk

Abstract. Model-Driven Engineering promotes the use of tailor-made modelling languages for software and systems engineering problems, with *metamodels* that encapsulate domain knowledge. Despite the importance of metamodelling in MDE, there is little empirical analysis of metamodels. What are the common characteristics of metamodels? Do modellers follow best practices? How do metamodels evolve over time? How does the size and structure of a metamodel affect the models that conform to it? This paper takes a first step towards answering these questions by automatically analysing the structural characteristics of a corpus of more than 500 publicly available Ecore metamodels.

1 Introduction

A common activity in Model-Driven Engineering (MDE) is *metamodelling* – the process of capturing the concepts and structures of a particular domain in a *metamodel* in order to construct models of that domain. Metamodels exist for general modelling languages (GMLs), such as UML, and for a range of domain-specific modelling languages (DSMLs), created to address specific software engineering domains. However, there is little guidance on the desirable or undesirable characteristics of metamodels for GMLs or DSMLs. There has been much research into the quality of models, but there is little empirical analysis of metamodels. If we can analyse different properties and calculate various metrics of metamodels, we may to be able to identify and detect good and bad practice, and understand the ways in which people are commonly structuring their metamodels today. The work here was motivated by the need to understand the common structural aspects of metamodels in order to tailor a model generation tool towards generating realistic metamodels for testing purposes.

In this paper we reveal common characteristics of metamodels that we have identified from an automated analysis of a corpus of over 500 publicly-available Ecore [9] metamodels. This is a first step: once we can analyse metamodels in different contexts and for different purposes, we can identify patterns of metamodelling best practice, and metamodel refactorings that facilitate model operations such as transformation. We can also develop an understanding of how metamodels evolve over time, and seek to control the complexity of evolving metamodels to minimise the effects on model artefacts, operations and tools. Our plan is to produce a set of standard metrics and analyses for metamodels

– similar to what exists in other domains (e.g. OO source code metrics) – and develop a supporting automated metamodel measurement workbench.

We use a general-purpose model management language, the *Epsilon Object Language* [6] (EOL), to compute counts or descriptive statistics on metamodel characteristics. The analysis shows the range of structural characteristics of metamodels, identifies some of the common practices of metamodellers – the most used parts of the metamodelling language, and the ways in which domain concepts are typically expressed – and raises many further questions about the commonalities and differences across the metamodelling corpus.

Section 2 introduces a set of metrics, focusing for now on structural analysis of metamodels. Section 3 presents the results of analysing the corpus of metamodels, and explores how the structure of one metamodel changed during its evolution. Section 4 describes related research.

2 Foundations: Structural Properties of Metamodels

The metrics considered in this paper focus on structural properties of metamodels – understanding how people structure their metamodels and answering the question *what do metamodels really look like?* The 19 metrics are examples of what can be achieved using simple EOL programs. We use EOL as it provides an executable query language, akin to OCL, that can easily be executed on metamodels. Our metrics are grouped into two categories: those related to meta-classes, and those concerning meta-features (attributes and references). The full list of metrics can be found on our website: `www.jamesrobertwilliams.co.uk/mm-analysis`. We summarise them now.

Our initial set of meta-class metrics focuses on the frequencies of meta-classes with various properties in a metamodel. This includes the *total number of meta-classes* metric, which gives an indication of the *size* of a metamodel, whilst the *total number of concrete meta-classes* and the *total number of abstract meta-classes* metrics provide more detail. Incidentally, though unintentionally, our metrics overlap and extend the metrics defined in recent work by Ma et al [7]. We also define metrics to inspect the number of features in a meta-class. Featureless classes may be considered to be bad design; detecting these in metamodels would highlight bad practice. We define metrics on two kinds of featureless meta-class: *immediately* featureless classes – those that have no attributes or references, but may inherit features from a superclass; and *completely* featureless classes have absolutely no features. Further metrics might explore the frequency of reference features, as compared to attribute features, or the distribution of features across hierarchies. In addition to counting, we can create descriptive statistics such as means and medians. We also calculate the average number of features per class, broken down by feature kind (attribute or reference). These metrics can be used to analyse whether there is a tendency to create many small classes, develop 'God' classes, or distribute features across classes.

The metrics concerning meta-features are *global* – referring to the number of occurrences of features in an entire metamodel and illustrate how metamod-

ellers commonly define the data (attributes) in metamodels and how they relate meta-classes to one another. These metrics include: counts of the total number of features in a metamodel (attribute, references, and combined); the types of references being defined (*containment* or *non-containment* – *compositions* or *associations* in UML terms); and examinations of the upper multiplicity bounds of references.

3 Analysis: What do Metamodels Really Look Like?

This section uses the metrics overviewed in the previous section to analyse, firstly, a large number of metamodels in an attempt to the common structural properties of metamodels. By computing these properties, we hope to inform the community of how people are modelling domains and attempt to learn how to improve current practice. Secondly, we analyse the evolution of a large metamodel over 11 minor versions and see how these properties change over time. The analysis script, the corpus of metamodels, and more detailed results are available online at: `www.jamesrobertwilliams.co.uk/mm-analysis`.

3.1 Analysing the Corpus of Metamodels

We have accumulated a corpus of 537 publicly available Ecore [9] metamodels. The corpus is made up of metamodels collected from GitHub, Google Code, the AtlantEcore Zoo, the EMFText Zoo, and from internal projects[1]. The corpus includes many well known modelling languages – such as the UML, DODAF, and Marte – as well as metamodels for many programming languages such as Java, C#, C, and Pascal, and many domain-specific metamodels. We then collated the scores and now describe the results. Due to space limitations, graphical visualisations of these statistics can be found at the web page above.

Meta-class Metrics The median total number of meta-classes in the corpus is 13, with a mean of 39.3, a maximum of 912, and a minimum of one. This suggests that metamodels (at least, in this corpus) are often fairly small. Twelve of the 537 metamodels have a single meta-class. Five of these metamodels are meaningless and should be removed, four were extensions of other metamodels, and three were domain-specific metamodels which also defined custom data types or enumeration types. Although small, a single-class metamodel can still define a suitable modelling language for some domains. The corpus showed that abstract meta-classes were not popular: 44% of metamodels did not contain a single meta-class denoted as being abstract. Furthermore, 96% of the corpus has fewer than 20 abstract meta-classes, whereas only 69% of the corpus has fewer than 20 concrete meta-classes. This is arguably due to the small average size of the corpus:

[1] GitHub: `github.com`; Google Code: `code.google.com`; AtlantEcore Zoo: `www.emn.fr/z-info/atlanmod/index.php/Ecore`; EMFText Zoo: `www.emftext.org/index.php/EMFText_Concrete_Syntax_Zoo`

smaller metamodels are likely to contain only concrete classes, whereas large metamodels are more likely to utilise abstract classes. Featureless classes were uncommon: 58% of the corpus has no completely featureless classes, and 27% have no immediately featureless classes. Interestingly, in the UML metamodel (developed by the Eclipse UML2 project (`http://www.eclipse.org/uml2/`)) 50 of the 227 meta-classes were immediately featureless, 40 of those were concrete. Immediately featureless classes are much more common than completely featureless ones, and it is more likely that these immediately featureless classes are concrete. Further analysis would likely show that these are specialisations of abstract classes, perhaps to provide some extra semantics to the hierarchy.

Meta-feature Metrics The median number of meta-features per metamodel is 23.5, with a mean of 69.2, a maximum of 2410, and a minimum of zero. Metamodels in the corpus commonly have more references (median 13.5, mean 43.0) than attributes (median 8, mean 26.2). The average metaclass has 2.1 features: 1.15 references and 0.95 attributes. The large number of featureless classes present in the corpus affects these data. If we exclude featureless classes when calculating the average features per class, we obtain the same distributions, although the mean number of features per meta-class increases slightly to 2.3, with 1.3 references and 1.0 attributes.

On average metamodels contain more non-containment references (median 6, mean 27.3) than containment references (median 5, mean 15.7). With respect to reference upper bounds, we find that they are set to 'one' 52% of the time, to 'many' 47% of the time, and are explicitly given a value just 1% of the time. The trend towards selecting 'many' as opposed to explicitly defining an upper bound might be attributed to the inherent uncertainty in modelling [11] (of course, sometimes specifying an upper bound as 'many' is perfectly acceptable and not related to domain uncertainty).

3.2 Analysing the Evolution of a Metamodel

The previous analysis considered only one fixed state of each metamodel, and doesn't capture how these properties change over time. Understanding how metamodels evolve can provide many insights, such as highlighting smells or anti-patterns [4]. Moreover, developers of metamodelling tools can use the information to provide the most appropriate support for practitioners, such as for the development of model migration [8] tools. We analyse 11 versions of the *Graph* metamodel, part of the *Graphical Modeling Framework* [5], an Eclipse project for developing graphical editors for modelling languages. We analyse versions 1.23 to 1.33 inclusive. More detailed results can be found on our web page.

The analysis exposed some major structural refactorings that occurred at version 1.29. The total number of meta-classes stays constant, however the number of concrete classes decreases by 25%. These structural refactorings also manifest in the total number of features, increasing at version 1.29, whilst the aver-

age number of features per classes stays fairly constant. Perhaps most revealing, however, are the featureless classes metrics. Many newly introduced classes were immediately featureless, and the change in numbers of abstract and concrete meta-classes suggests that concrete-classes were refactored to abstract. The number of totally featureless classes, however, stayed constant, suggesting that meta-classes were introduced as specialisations, and these refactorings were a reorganisation of the class hierarchy.

This analysis only considers the evolution of a single metamodel. It would be interesting to discover whether the behaviour shown in this example is commonly found in other metamodels, or to see whether we can discover patterns of evolution by analysing a large number of metamodel evolutions. Unfortunately, however, these intermediate models may not be available in all cases.

3.3 Threats to Validity

Although we have analysed a large number of metamodels, we need to be careful with our conclusions. Almost all of the metamodels in the corpus were publicly available and downloaded from the internet. These metamodels may not be representative of the metamodels that are used in industry. Many of the metamodels were of well known modelling languages (e.g. UML, CORBA's IDL, DODAF) which are used in industry, but the corpus may not represent the bespoke modelling languages that are developed in practice. Furthermore, all of the metamodels that were analysed were Ecore metamodels. Different metamodelling technologies may have differing properties, and the metamodelling language itself may cause the kinds of behaviour shown in this paper. Ecore, however, is arguably the current de facto modelling language and is equivalent to EMOF so the insights that it offers are still useful.

4 Related Work

While there is a significant amount of work in the field of analysis of MDE artefacts, the majority of the related work we have encountered has a different focus to this paper. The closest work to ours is Cadavid et al [2] who present an empirical analysis of the ways MOF and OCL are used together. They define metrics to analyse the complexity of 33 metamodels, their constraints, and the coupling between the two. The work in this paper aims to complement the work of Cadavid et al. with deeper analysis of the metamodel structure (as opposed to its relationship with its constraints). Vepa et al. [10] measure a set of metamodels that is stored in the Generative Modeling Technology/ATLAS MegaModel Management (GMT/AM3) Repository. This work focuses on the model repository and the measuring technique, rather than the presentation of the results of the analysis. Finally, Arendt et al. [1] describe an Eclipse plugin, *EMF Metrics*, that can be used to assess the quality of EMF metamodels based on nine quantitative criteria. The aforementioned approaches focus mainly on model quality (as with software), while we (and Cadavid) are interested in understanding the usage of metamodelling languages and how metamodels are constructed.

5 Conclusion

In this paper we have posited the need for a deeper understanding of metamodels. We illustrate structural analysis on a corpus of over 500 metamodels, gaining insight into how metamodels are commonly structured, and how they evolve over time. We are now in a position to start the analysis of good and bad practice in metamodelling, for GMLs or DSMLs, and in different model management contexts. To facilitate development of further metrics, we are creating a metrics metamodel. We plan to create a web-based automated metamodel measurement workbench that allows users to upload and analyse their own metamodels, which will automatically augment to the results given here. We plan to devise a comprehensive set of metrics, and develop state-of-the-art analyses for metamodels, taking inspiration from similar domains, such as bad smell detection [4] and design patterns [3].

Acknowledgements

This research was part supported by the EPSRC, through the Large-Scale Complex IT Systems project (EP/F001096/1) and by the EU, through the Automated Measurement and Analysis of Open Source Software (OSSMETER) FP7 STREP project (318736).

References

1. T. Arendt, P. Stepien, and G. Taentzer. Emf metrics: Specification and calculation of model metrics within the eclipse modeling framework. In *BENEVOL 2010*, 2010.
2. J. Cadavid, B. Baudry, and B. Combemale. Empirical evaluation of the conjunct use of MOF and OCL. In *EESSMod 2011*, 2011.
3. H. Cho and J. Gray. Design patterns for metamodels. In *Proc. DSM'11*, 2011.
4. M. Fowler. *Refactoring: improving the design of existing code.* Addison-Wesley Professional, Boston, MA, USA, 1999.
5. R. C. Gronback. *Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit.* Addison-Wesley Professional, 2009.
6. D. S. Kolovos, R. F. Paige, and F. A. C. Polack. The Epsilon Object Language (EOL). In *ECMDA-FA*, volume 4066 of *LNCS*, pages 128–142. Springer, 2006.
7. Zhiyi Ma, Xiao He, and Chao Liu. Assessing the quality of metamodels. *Frontiers of Computer Science*, pages 1–13, 2013.
8. L. M. Rose, R. F. Paige, D. S. Kolovos, and F. A. C. Polack. An analysis of approaches to model migration. In *Proc. MoDSE-MCCM 2009*, October 2009.
9. D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks. *EMF Eclipse Modeling Framework.* The Eclipse Series. Addison-Wesley, second edition, 2009.
10. E. Vépa, J. Bézivin, H. Brunelière, and F. Jouault. Measuring model repositories. In *Proc. MoDELS/UML 2006, Workshop on Model Size Metrics*, October 2006.
11. J. R. Williams, F. R. Burton, R. F. Paige, and F. A. C. Polack. Sensitivity analysis in model-driven engineering. In *Proc. MODELS 2012*, 2012.

Online Img2UML Repository: An Online Repository for UML Models

Bilal Karasneh[1] and Michel R. V. Chaudron[2,1]

[1] Leiden Institute of Advanced Computer Science, Leiden University, the Netherlands
bkarasne@liacs.nl
[2] Joint Department of Computer Science and Engineering,
Chalmers University of Technology and Gothenburg University, Sweden
chaudron@chalmers.se

Abstract. The Img2UML repository is a repository of UML models. A huge amount of UML models is available on the Internet – mostly in the form of images. This repository aims to offer these UML class diagram as a searchable XMI Database. The information that is in the XMI files is stored in the repository database. This repository will be useful for research as the first corpus of UML models. This repository will provide a good place for researchers and students to study and analyze UML models. This improves UML studying in research, education, and industry. In this paper, we outline the Img2UML repository, illustrate some of the research made possible, and discuss future plans.

Keywords: Model Repository, UML, XMI

1 Introduction

The UML (Unified Modeling Language) language enables the graphical, high-level representation of software. UML models are created during different stages of the software development process. Often, a UML design is the blueprint of the software, and a good UML design helps to realize good software implementations.
Many aspects of software development across the software lifecycle can be measured with a high degree of automation and efficiency. Most software measurements are focused on code instead of on the design of the software, because: 1) Numerous metrics like complexity, maintainability, and readability have been developed for code, but for software design metrics still suffer of many problems. 2) Sharing of the source code artifacts is well supported through platforms such as GitHub [19], and there is no adequate support for sharing modeling artifacts.

One of the main problems of studying UML models is the lack of sharable software development software [1]. In Software Engineering there is a need to share modeling artifacts [2]. Until now, there is no public repository for models. The collection of models from commercial software development is difficult because for different reasons companies like to keep their system design confidential. In open source software, development use of UML is not as common as the (inevitable) use of source

code. This makes collecting UML models more difficult, and this difficulty makes empirical research of UML challenging. Moreover, there is no open technology for creating model-repositories as there exist for source code. Many free code repositories are available, which improves the ability of developing code metrics, and facilitates empirical research for source code domain in general.

To facilitate the studying UML models, a set of UML models must be collected. It is challenging to collect UML models because there are a large variety of representations (both graphically and in terms of XMI) of UML models by different UML-CASE (Computer Aided Software Engineering) tools.

In our proposed repository, we start with focusing on one type of UML model, which it is UML class diagram. This selection is done based on the importance of this diagram in software development and its availability. Class diagrams are ubiquitous in UML modeling. UML class diagrams are the most important structural model of the UML, as it shows the static description of the system in terms of classes, relationships and constrains in the relationships [3].

We found that UML models are available in abundance on the Internet, but rather than in CASE-tool format, they are stored in image formats. The problem with image formats is that the model-content of the images cannot be easily extracted out of them. Although many CASE tools support features like creating, modifying and exporting UML models into different formats, current CASE tools cannot recognize UML in images. This inability of CASE tools limits the usability of the availability of UML models in images. For our repository, we are collecting UML class diagram in images from the Internet, and use an image recognition tool [4] that coverts UML class diagram in images into UML models. After this transformation, the tool then saves the images, XMI files and the content of XMI files into the repository. In this way we unlock a huge number of UML class diagrams, which gives a great opportunity for empirical UML research.

The paper is structured as follows: Section 2 describes related work. Section 3 motivates the usefulness of the repository. Section 4 describes the construction of the repository. The conclusion and future work are in Section 5.

2 Related work

Nowadays, a few UML repositories are available. These repositories are supported by CASE tools vendors [17][18] on a commercial basis. Because of the associated costs, these models are not considered attractive from the viewpoint of academic research.

Another kind of repository is a general model repository. In [1], authors proposed repository for model-driven development (ReMoDD) that contains many documented case studies. This repository is a great asset for researchers where they can find many examples of models as well as research studies. However, UML models in ReMoDD are stored as files, so that models are not searchable, and to see a model you have to download it and then open it using compatible CASE tool. In addition, some of case studies do not contain UML models. To complement this, we propose the idea of

creating a repository for UML class diagrams as first step towards creating a repository for UML models.

Companies have a huge amount of information at their disposal that is stored in as paper or poorly structured format as PDF, and they need to convert at least most important information into richer format that can be easily searched and modified [5]. In software engineering, this challenge is bigger as software documentation is rich in graphical content. UML models are one of these contents that are mostly available as images in software documentation and on the Internet. The problem is the lack of mapping from a pixel-based diagram to the underlying engineering model conveyed by the diagram [6].

The area of converting engineering diagrams into engineering models has received some attention [5-11]. Some of this research is oriented at recognizing graphic objects or symbols in images [7-10]. Other research aims at recognizing entire models in images [11]. For specifically for UML diagrams, researchers have focused on converting hand-drawn sketching of UML class diagram into models [12-15]. In general, hand-drawn tools are an easy and fast way to create and (re)draw UML class diagram than UML CASE tool. However, redrawing UML models from paper in order to enable editing them again is very time-consuming. The algorithms that are used in recognizing UML diagrams in hand-drawn tools typically make use of information regarding the movement and order of drawing elements in the diagram. This makes these algorithms unsuitable for extracting UML models from 'finished' diagrams.

In our earlier work [4] we proposed the Img2UML tool that converts UML class diagrams (including class names, attributes, relationships) that are represented in image format into XMI (version 1.1, the UML version is 1.3). The resulting XMI files generated by the tool are compatible with StarUML [16]. This tool also contains functionality to save models as XMI files.

3 Usefulness of the repository

This new Img2UML repository aims to be a source for empirical studies of UML class diagrams. This repository opens up a lot of uses:

- It can be the basis for corpus studies for UML modeling, all available models are validated manually, and any mistake in the recognition can corrected manually.
- It can be the basis of metrics used in benchmarking for quality assurance of UML models, such as the average number of classes per model and the average number of attributes and operations per class. This provides an empirical basis for UML quality assurance.
- It can serve as a source of UML models that can be used in and shared across empirical studies in UML modeling
- It can serve as a source of examples of UML design that can be used for educational purposes, e.g. learning UML by examples.

This repository already contains 1000 class models. The repository can be used to analyze class diagrams, measure qualities, study typical flaws and their frequency of occurrence, compare quality-models, etc. Although more rare, the availability of dif-

ferent versions of models for one software system provides an opportunity to study the evolution of versions of the class designs. The repository can also be used for studying UML class diagram in software engineering classes. Students can reuse available class diagrams, share their knowledge, and engage in discussions about models.

Our system offers functionality for querying and searching the repository of models based on different keys such as model information (class name, attributes, etc.). Models in the repository can be classified and analyzed automatically by using some queries on the repository. For example, which class diagram contains a high number of classes, a high number of relationships (dependency or inheritance), or some design pattern. These queries can show common characteristics of class diagrams.

The content of ReMoDD and our repository are different in: First, ReMoDD contains documents, model files and codes, and in our repository only models are available. Second, all models in our repository are editable and searchable. Third, our models are collected from software documents on the Internet. However, models in ReMoDD seem to come from (industrial) case studies. Forth, our repository supports querying models, because models information (contents of XMI) such as names of classes, attributes and operations with relationships are stored in the database. Fifth, although our repository contains models that are created using different CASE tools, it is not obligatory to have these CASE tools to use these models because all XMI files are compatible with StarUML. Moreover, we could easily add a feature for exporting to other versions of XMI.

4 Repository Description

4.1 Collecting UML class diagrams

Different UML class diagram images are collected from the Internet using Google Image Search. These images vary in color, type, size, and resolution. Images are collected together with their URLs. These URL's are used as keys in the database and are used to prevent including duplicates of images. After this, additional manual checking is performed to assure that indeed no duplicate images end up in the repository.

4.2 Inserting model information into the database

The process of inserting models in the database can be divided into two parts: First, the UML class diagram is extracted from an image. For this, the Img2UML tool converts class diagrams in images into XMI files. Second, the XMI content is saved into the database. From XMI, classe names, attributes names, operations names and relationship types are read, and saved in the database. Figure 1 shows the structure of the database. The repository contains 10 tables, where the image_Table contains image_IDs, the available UML class diagram images, URLs of images and images prop-

erties such as width, height, and resolution. The xmi_Table contains XMI files and generals comments about the models. This comments will improved to be more classified, as comments about layout, understandability, complexity, recognition, etc.
Both attributes_Table and operations_Table contains attributes names and operations names and where this attributes and operations are available in which classes. The remains tables are related to the relationships, where each relationship is saved in the in details, for example in the generalization_Table, the generalization_Child and generalization_Parent show inheritance relationships between classes and xmi_ID shows that this relation is available in which model.

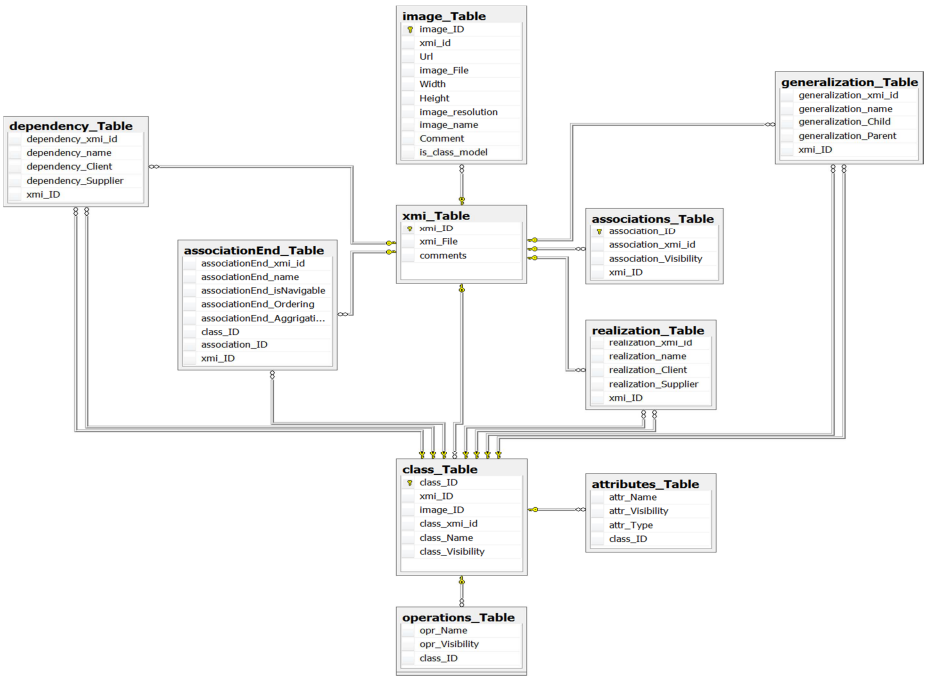

Fig. 1. Img2UML database structure

5 Conclusion and Future Work

In this paper, we proposed the Img2UML repository, a repository for UML diagrams based on UML class diagram in images that have been converted to XMI. The repository contains UML class diagrams images that are collected from the internet. The repository also contains the images' URLs and the corresponding XMI files that are generated via special tool created for recognizing UML models from diagrams. A web-based user interface will make the repository more available and accessible. The goal of the repository is to be a basis for UML models that can be used and shared across empirical studies.

For future work, we will evaluation different aspects of the repository. Also we aim to develop an API for uploading more UML class diagrams in images by users. More information and classification about available UML class diagrams will be supported, like information about the related software development project.

6 References

1. France, R., Bieman, J., Cheng, B.H.: Repository for model driven development (ReMoDD). Models in Software Engineering, pp. 311-317. Springer (2007)
2. Buse, R.P., Zimmermann, T.: Information needs for software development analytics. In: Proceedings of the 2012 International Conference on Software Engineering, pp. 987-996. IEEE Press, (2012)
3. Maraee, A., Balaban, M.: Efficient recognition of finite satisfiability in UML class diagrams: Strengthening by propagation of disjoint constraints. In: Model-Based Systems Engineering, 2009. MBSE'09. International Conference on, pp. 1-8. IEEE, (2009)
4. Karasneh, B., Chaudron, M. R. V.: Extracting UML Models from Images. In: 5th International Conference on Computer Science and Information Technology, CSIT 2013. (2013)
5. Tombre, K., Lamiroy, B.: Graphics recognition-from re-engineering to retrieval. In: Document Analysis and Recognition, 2003. Proceedings. Seventh International Conference on, pp. 148-155. IEEE, (2003)
6. Fu, L., Kara, L.B.: From engineering diagrams to engineering models: Visual recognition and applications. Comput. Aided Des. 43, 278-292 (2011)
7. Barrat, S., Tabbone, S.: A Bayesian network for combining descriptors: application to symbol recognition. International Journal on Document Analysis and Recognition (IJDAR) 13, 65-75 (2010)
8. Barrat, S., Tabbone, S., Nourrissier, P.: A bayesian classifier for symbol recognition. In: Seventh International Workshop on Graphics Recognition-GREC'2007. (2007)
9. Luqman, M.M., Brouard, T., Ramel, J.-Y.: Graphic symbol recognition using graph based signature and bayesian network classifier. In: Document Analysis and Recognition, 2009. ICDAR'09. 10th International Conference on, pp. 1325-1329. IEEE, (2009)
10. Yang, S.: Symbol Recognition via Statistical Integration of Pixel-Level Constraint Histograms: A New Descriptor. IEEE Trans. Pattern Anal. Mach. Intell. 27, 278-281 (2005)
11. Yu, Y., Samal, A., Seth, S.C.: A System for Recognizing a Large Class of Engineering Drawings. IEEE Trans. Pattern Anal. Mach. Intell. 19, 868-890 (1997)
12. Chen, Q., Grundy, J., Hosking, J.: SUMLOW: early design-stage sketching of UML diagrams on an E-whiteboard. Softw. Pract. Exper. 38, 961-994 (2008)
13. Hammond, T., Davis, R.: Tahuti: a geometrical sketch recognition system for UML class diagrams. ACM SIGGRAPH 2006 Courses, pp. 25. ACM, Boston, Massachusetts (2006)
14. Lank, E., Thorley, J., Chen, S., Blostein, D.: On-line Recognition of UML Diagrams. In: Proc. 6th ICDAR (2001) 356-360
15. Lank, E., Thorley, J.S., Chen, S.J.-S.: An interactive system for recognizing hand drawn UML diagrams. Proceedings for CASCON 2000; 2000. p. 7
16. StarUML - http://staruml.sourceforge.net/en/
17. Enterprise Architect - http://www.sparxsystems.com/
18. Visual Paradigm - http://www.visual-paradigm.com/
19. GitHub - https://github.com/