Research Reports in Software Engineering and Management 2012:03 ISSN 1654-4870

Proceedings of the 2nd Workshop on Experiences and Empirical Studies in Software Modelling

Michel Chaudron, Marcela Genero,
Silvia Abrahão, Lars Pareto

CHALMERS | UNIVERSITY OF GOTHENBURG
Department of Computer Science and Engineering

Proceedings of the 2nd Workshop on Experiences and Empirical Studies in Software Modelling

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

Report no 2012:03
ISSN: 1651-4769
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden

Telephone + 46 (0)31-772 1000

Göteborg, Sweden 2012

Workshop Proceedings

ACM/IEEE 15th International Conference on Model Driven Engineering Languages and Systems

Sept. 30th – Oct. 5th 2012
Innsbruck, Austria

Experiences and Empirical Studies in Software Modelling (EESSMod 2012)

September 2

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

EESSMOD 2012
Second International Workshop on Experiences and Empirical Studies in Software Modelling

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

chaudron@chalmers.se, Marcela.Genero@uclm.es, sabrahao@dsic.upv.es, pareto@chalmers.se

[1] The Software Engineering Division,
Chalmers University of Technology and University of Gothenburg,
SE-412 96 Gothenburg, Sweden

[2]ALARCOS Research Group, University of Castilla-La Mancha
Paseo de la Universidad 4, 13071, Ciudad Real, Spain

[3] ISSI Research Group, Department of Information Systems and Computation,
Universitat Politècnica de València
Camino de Vera, s/n, 46022, Valencia, Spain

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, and for enabling simulations and generating test cases in a model-driven engineering approach. Evaluation of modelling techniques, languages and tools is needed to assess their advantages and disadvantages, to ensure their applicability to different contexts, their ease of use, and other aspects such as required skills and costs – both isolated evaluations and comparisons with other methods.

The need to reflect upon the adoption of software modelling in industry and a growing understanding of the role of empirical research in technology adoption led us to organize the International Workshop on Experiences and Empirical Studies in Software Modelling (EESSMod) as a satellite event of the ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MoDELS), with professionals and researchers interested in software modelling as intended audience, and with the objective to

- build a better understanding of the industrial use of modelling techniques, languages and tools;
- start building theories about the effectiveness of modelling approaches;
- identify directions for future research in the field;
- facilitate and encourage joint research and experimentation in software modelling.

The 1st workshop was held in Wellington, NZ, and the 2nd (presented in these proceedings) in Innsbruck, AU. In all, 18 papers were submitted to the 2nd workshop. Each paper was peer reviewed by three independent PC members (from a committee of 23). The review process resulted in 9 submissions being accepted for publication, and 6 submissions for poster presentation (out of which three accepted). The accepted papers were categorized, in search for common research themes, which resulted in the following categories of research problems:

1. **The fitness for purpose of modeling.** We know that modeling is great - but when, where and for what? (3 papers)
2. **The cognitive aspects of modeling.** Models support mental activities better than code - but which, how and to what degree? (3 papers)
3. **Modeling and process improvement.** Modeling enables process improvement - but where do these improvements lead? (3 papers)

These proceedings collect the papers presented at the workshop as well abstracts for the poster presentations. We would like to thank the authors for submitting their papers to the Workshop. We are also grateful to the members of the Program Committee for their efforts in the reviewing process, and to the MoDELS2012 organizers for their support and assistance during the workshop organization. More details on the Workshop are available at http://www.eesmod.org

Gothenburg, Ciudad Real, Valencia
27 September 2012

Michel Chaudron
Marcela Genero
Silvia Abrahão
Lars Pareto

Program Committee

Silvia Abrahao, Universitat Politècnica de València, Spain
Bente Anda University of Oslo, Norway
Teresa Baldassarre, Università degli Studi di Bari, Italy
Narasimha Bolloju, City University of Hong Kong, China
Danilo Caivano, Universitàdegli Studi di Bari, Italy
Jeffrey Carver, University of Alabama, USA
Michel Chaudron, Chalmers | University of Gothenburg, Sweden
José Antonio Cruz Lemus, University of Castilla-La Mancha, Spain
Holger Eichelberger, Universität Hildesheim, Germany
Félix Garcia, University of Castilla-La Mancha, Spain
Marcela Genero, University of Castilla-La Mancha, Spain
Carmine Gravino, University of Salerno, Italy
Brian Henderson Sellers, University of Technology, Sydney, Australia
Jan Mendling, Humboldt-University Berlin, Germany
Parastoo Mohagheghi, Norwegian University of Science and Technology, Norway
James Nelson, Southern Illinois University, USA
Lars Pareto, Chalmers | University of Gothenburg, Sweden
Jeffrey Parsons, Memorial University of Newfoundland, Canada
Keith Phalp, Bournemouth University, UK
Giuseppe Scanniello, Università degli Studi della Basilicata, Italy
Keng Siau, Missouri University of Science and Technology, USA
Dag Sjøberg, University of Oslo, Norway
Marco Torchiano, Politecnico di Torino, Italy

Workshop Program

Session I: THE FITNESS FOR PURPOSE OF MODELING *Modeling is great - but when, where and for what?* Chair: Michel Chaudron
Marco Torchiano, Federico Tomassetti, Filippo Ricca, Alessandro Tiso and Gianna Reggio. *Benefits from Modeling and MDD Adoption: Expectations and Achievements*
Rut Torres Vargas, Ariadi Nugroho, Michel Chaudron and Joost Visser. *The Use of UML Class Diagrams and Code Change-proneness*
Adrian Kuhn and Gail Murphy. *Lessons Learned from Evaluating - MDE Abstractions in an Industry Case Stu*
Session II: COGNITIVE ASPECTS OF MODELING *Models support mental activities better than code - but which, how and to what degree?* Chair: Marcela Genero
Giuseppe Scanniello, Carmine Gravino and Genny Tortora. *Does the Combined use of Class and Sequence Diagrams Improve the Source Code Comprehension? Results from a Controlled Experiment*
Hafeez Osman, Arjan van Zadelhoff, Dave Stikkolorum and Michel Chaudron. *UML Class Diagram Simplification: What is in the developer's mind?*
Stefan Zugal, Jakob Pinggera, Hajo A. Reijers, Manfred Reichert and Barbara Weber. *Making the Case for Measuring Mental Effort*
Session III: MODELING AND PROCESS IMPROVEMENT Modeling enables process improvement - but where do improvements lead? Chair: Silvia Abrahão
R.J. Macasaet, Manuel Noguera, Maria Luisa Rodriguez and Jose Luis Garrido. Sam Supakkul, Lawrence Chung, *Micro-business Behavior Patterns associated with Components in a Requirements Approach*
Gianna Reggio, Maurizio Leotta, Filippo Ricca and Egidio Astesiano. *Five Styles for Modelling the Business Process and a Method to Choose the Most Suitable One*
Lamia Abo Zaid and Olga De Troyer. *Modelling and Managing Variability with Feature Assembly – An Experience Report*
Session IV: POSTER AND NETWORKING SESSION *What else is going on in EESSMOD community?* *Ideas for cross site collaborations?* Chair: Lars Pareto
Daniel Méndez Fernández and Roel Wieringa, *Empirical Design Science for Artefact-based Requirements Engineering Improvement*. (Poster)
Ana M. Fernández-Sáez, Peter Hendriks, Werner Heijstek and Michel R.V. Chaudron, *The Role of Domain-Knowledge in Interpreting Activity Diagrams – An Experiment* (Poster)
Vinay Kulkarni, *Modeling and Enterprises – the past, the present and the future.* (Poster)

Content

Benefits from Modelling and MDD Adoption: Expectations and Achievements

Marco Torchiano
Politecnico di Torino
Torino, Italy
marco.torchiano@polito.it

Federico Tomassetti
Politecnico di Torino
Torino, Italy
federico.tomassetti@polito.it

Filippo Ricca
DIBRIS, Università di Genova
Genova, Italy
filippo.ricca@unige.it

Alessandro Tiso
DIBRIS, Università di Genova
Genova, Italy
alessandro.tiso@unige.it

Gianna Reggio
DIBRIS, Università di Genova
Genova, Italy
gianna.reggio@unige.it

ABSTRACT

The adoption of Model Driven Development (MDD) promises, in the view of pundits, several benefits. This work, based on the data collected through an opinion survey with 155 Italian IT professionals, aims at performing a reality check and answering three questions: (i) Which benefits are really expected by users of modeling and MDD? (ii) How expectations and achievements differ? (iii) Which is the role of modeling experience on the ability of correctly forecasting the obtainable benefits?

Results include the identification of clusters of benefits commonly expected to be achieved together, the calculation of the rate of actual achievement of each expected benefit (varying dramatically depending on the benefit) and the "proof" that experience plays a very marginal role on the ability of predicting the actual benefits of these approaches.

Categories and Subject Descriptors

D.2.10 [**Software Engineering**]: Design—*methodologies*

General Terms

Measurement, Languages

Keywords

Industrial survey, Model Driven Development (MDD)

1. INTRODUCTION

Models are used in software development with the general goal of raising the level of abstraction. The approaches resorting on models are various and fall under different names: from simple modeling to model-driven development (MDD) [16], model-driven engineering (MDE) [17], and model-driven architecture (MDA) [13]. In practice, models can be transformed and code can be generated from them by means of (semi) automatic transformations. Alternatively, models can also be directly executed/interpreted (in that case they are called executable models). In the following, we will address all these related techniques with the abbreviation MD* [22].

There is a number of benefits commonly associated with the usage of models: they range from an improvement in the quality of documentation, to huge gains in productivity and reduction of defects [1]. Hype is frequently associated to software development processes/techniques until they are not yet mainstream and fully understood [4]; we think it is also the case for modeling and MD*. In our opinion it is important to distinguish which benefits associated with modeling and MD* are real and which contribute just to create hype.

The literature reports different success stories about MD* (e.g., [1, 8]). Those stories tell us which benefits we can get in the best-case scenario. What about the other cases? How frequently are the failures? How many practitioners were disappointed with MD* usage? How frequently the promises of MD* are not maintained in reality? We think it is important to answer these questions to provide guidance to practitioners and clarify what can be reasonably expected from modeling and MD* and what can possibly, but not so easily, be obtained.

The large number of methods under the MD* name is considered to be still evolving and not yet completely mature. The first success stories were heard a long time ago but the knowledge to make those successes consistently repeatable is still missing. Being the discipline not yet fully understood, and the underlying knowledge not yet codified, expertise is the only resource we can rely on when a MD* solution is designed. Thus, another interesting element to investigate is the role of expertise. Being expertise difficult and controversial to measure directly, we can use the number of years of experience as an approximation. The resulting question is: does the level of experience help when adopting modeling and MD*? In particular, does it help when forecasting the outcome of modeling?

In the next section, we present the design of the general survey, the research questions addressed in this work and the analysis we performed to answer them (Sect. 2). Then, we present the results found (Sect. 3). In Sect. 4, we discuss the results and later we compare them with previous work (Sect. 5). Finally, we draw our conclusions (Sect. 6).

2. SURVEY DESIGN

We conceived and designed the study with the goals of understanding:

G1 the actual diffusion of software modeling and MD* in the Italian industry,

G2 the way software modeling and MD* are applied (i.e., which processes, languages and tools are used), and

G3 the motivations either leading to the adoption (expected benefits) or preventing it (experienced or perceived problems).

The above goals cover a wide spectrum, which base been partly considered in previous works [19, 21]. The cited articles provide also more details about the design of the survey. In this work, we consider only a limited portion of those goals, in particular we focus on the benefits, that is the first part of goal G3.

2.1 Research questions

The goal we investigate in this paper, i.e., examine expectations and real achievements of benefits due to the adoption of modeling and MD*, can be detailed into three main research questions. First of all, we consider what benefits the adopters expect from modeling (RQ1), then we examine what is the actual frequency of achievements (RQ2). Finally, we consider if the experience can lead to more realistic expectations (RQ3).

- **RQ1: Which are the benefits expected from modeling and MD* adoption?**
  - **RQ1.1: Which are the most expected benefits?** We want to understand which are the anticipated benefits that also represent plausible motivations for adopting modeling and MD*.
  - **RQ1.2: Which are the relations between expectations?** We envision group of related benefits, i.e., benefits that are supposed to be achieved together.
- **RQ2: Which are the most frequently fulfilled expectations?** We aim at understanding how well confirmed benefits match expectations, in order to understand the capability of participants to predict the results and spot possibly hard-to-gain benefits.
- **RQ3: Does experience in modeling improves accuracy of benefits achievement forecasts?** Correctly forecasting achievable benefits is a key factor, e.g., in cost estimation, therefore we are aim to understand whether (or not) experience improves (or affects) the performance in this respect.

2.2 Instrument

We selected an opinion survey [6] with IT practitioners, administered through a web interface, as instrument to take a snapshot of the state of the practice concerning industrial MD* adoption. In the design phase of the survey we drew inspiration from previous surveys (i.e., [20] and [9]) and we followed as much as possible the guidelines provided in [12].

The survey has been conducted through the following six steps [12]: (1) setting the objectives or goals, (2) transforming the goals into research questions, (3) questionnaire design, (4) sampling and evaluation of the questionnaire by means of pilot executions, (5) survey execution and, (6) analysis of results and packaging.

For the specific purpose of this paper we analysed a few items contained in the questionnaire (a more detailed description is available in [19]).

An initial item (*Dev08*) concerned whether models are used in the respondent organization for software development. For the respondents who provided a positive answer to such item we administered a further item measured using the question "*What are the benefits expected and verified from using modeling (and MD*)?*". This was designed as a closed option question; the list of benefit that we presented the respondents was compiled on the basis of the literature and includes:

- Design support
- Improved documentation
- Improved development flexibility
- Improved productivity
- Quality of the software
- Maintenance support
- Platform independence
- Standardization
- Shorter reaction time to changes

For each benefit the respondent could indicate whether the benefit was expected and/or verified.

To evaluate the experience in modeling we considered one item that measured the years since the initial adoption of modeling or MD* in the work-group of the respondents.

2.3 Analysis

Whenever possible we addressed the research questions with the support of statistical tests. In all the tests we used we considered an $\alpha = 0.05$ as a threshold for statistical significance, that is we accept a 5% probability of committing a type I error.

RQ1.1: to answer this RQ we simply ranked the benefits by the number of respondents expecting that benefit in descending order. In addition, using the proportion test, we compute the estimate proportion of respondents who expect the benefit and the corresponding 95% confidence interval.

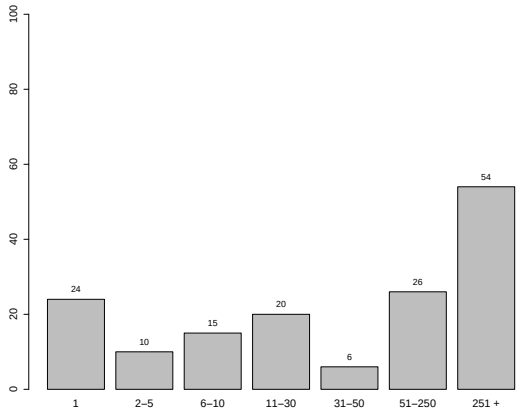


Figure 1: Size of respondents' companies

The interval is useful to understand the precision of the result.

RQ1.2: we looked at the relations between all possible pairs of benefits. We calculated the Kendall rank correlation coefficient between the expectations of each pair of benefits, obtaining a symmetrical measure of the strength of association between the expectations of the two benefits. Positive values represent a positive association while negative values represent a negative association. The absolute value of the correlation represent the strength of the association and it can vary from zero to one.

RQ2: to answer this question we examined for each benefit how frequently it was achieved when expected. We can look at the issue as a classification problem – expected benefits correspond to predictions and verified benefits to observations – then the above measure corresponds to the precision of the classifier.

RQ3: in this case we considered the factor experience in modeling, so we divided the respondents in two groups: low experienced practitioners (i.e., < 5 years of experience in modeling) and high experienced practitioners (i.e., ≥ 5 years of experience in modeling). Finally, we built the contingency table and performed the Fisher test considering the two groups (low and high experience) and the number of correct and wrong forecasts done by each group.

3. RESULTS

In summary, over a period of 2 months and half, we collected 155 complete responses to our questionnaire by means of an on-line survey tool[1].

The most of the companies where the respondents work are in the IT domain (104), then come services (15) and telecommunications (11). The distribution of the companies size where the respondents work is presented in Figure 1.

Among the respondents, on the basis of item *Dev08* we were able to identify 105 respondents using modeling and/or MD* techniques. We apply the analysis described above only to the information collected from respondents who adopted modeling.

[1]LimeSurvey: http://www.limesurvey.org

3.1 RQ1: Which are the benefits expected from modeling adoption?

RQ1.1: Which are the most expected benefits? In Table 1 we report for each benefit the frequency of expectation (column *Freq.*) and the corresponding percentage of respondents (column *Estimate*).

Improved documentation is the most expected benefit, with almost 4 out of 5 respondents anticipating it. Also *Design support*, *Quality of the software*, *Maintenance support*, and *Standardization* are frequently expected. For all of the top 5 benefits we are 95% sure that more than 50% of modeling adopters expect them: in fact the confidence interval (C.I.) lower bounds are larger than 50%. The remaining benefits, *Improved development flexibility*, *Improved productivity*, *Shortened reaction time to changes*, and *Platform independence* are less popular, with the latter typically expected by less than 40% of respondents.

RQ1.2: Which are the relations between expectations? We report the statistically significant relations among benefits in the graph shown in Figure 2: the nodes represent the individual benefits, the edges represent a statistically significant relation which is reported as edge label. The layout of the nodes is computed considering the Kendall rank correlation coefficient (KC) (the length of the edge should be as much as possible inversely proportional to the Kendall distance) and additional constraints to improve the readability avoiding the overlaps of nodes and labels.

The benefit expected together ($KC > 0$) are linked by continuous black lines, while the benefits whose expectations tend to exclude each other ($KC < 0$) are linked by dashed red lines, with circles at the ends.

All the significant relations were positive except one, that between *Improved documentation* and *Improved development flexibility*: who expects one of these two benefits tend to not expect the other one.

By observing Figure 2, we can note two distinct clusters: the first includes *Improved documentation*, *Design support* and *Maintenance support*. The second one includes *Improved development flexibility*, *Shorter reaction time to changes*, *Platform independence*, *Standardization* and *Improved productivity*. *Quality of the software* appears to be a transversal benefit, connecting the two clusters.

The two cluster contain three maximal cliques[2]: the smallest (left side) cluster correspond to a three-vertexes maximal clique, while the largest one (right side) correspond to a four-vertexes and a three-vertexes cliques that share a node (*Reactivity to changes*).

[2]From Wikipedia: in the mathematical area of graph theory, a clique in an undirected graph is a subset of its vertices such that every two vertices in the subset are connected by an edge.

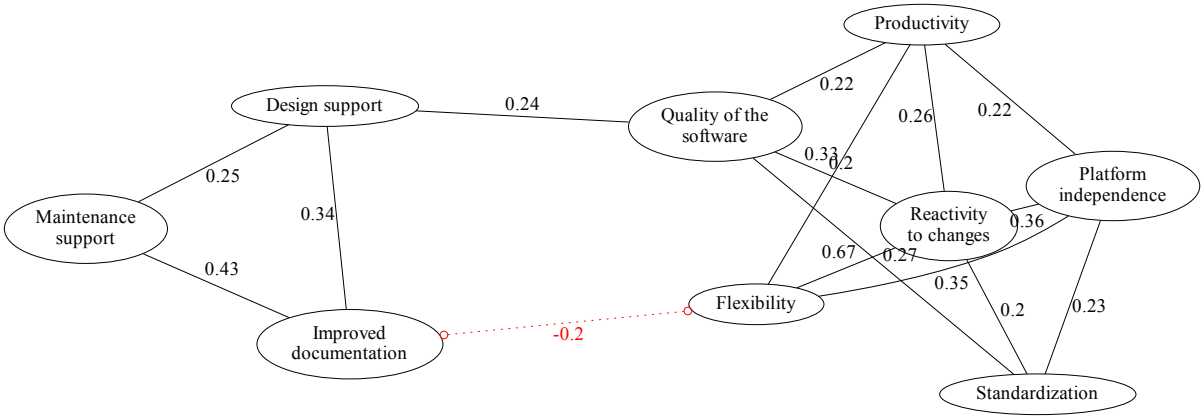


Figure 2: Relations among benefits expectations.

3.2 RQ2: Which are the most frequently fulfilled expectations?

This research question concerns how often the verification of a benefit met the expectation. It is measured as the frequency of verified benefit given the benefit was expected. Results are reported in the rightmost column of Table 1 (Fulfillment rate).

Design support has the highest fulfilment rate: 60 respondents out of the 81 who reported to expect it (i.e., 78%) actually achieved the benefit. Also *Documentation improvement* is consistently verified when expected, the same is not true for all the other benefits. *Standardization* and *Maintenance support* are just above the parity (it means are slightly mainly achieved than not achieved, when expected) and all the others are more often not achieved than achieved. *Platform independence* and *Reactivity to changes* have a really low fulfilment rate, representing very often a delusion for practitioners.

3.3 RQ3: Does experience in modeling improves accuracy of benefits achievement forecasts?

The low experienced practitioners group (< 5 years of experience in modeling) is constituted by 50 persons, whereas the high experienced practitioners group (i.e., ≥ 5 years of experience in modeling) by 55. Thus, the two groups are balanced.

Applying the Fisher test to the built contingency table, even adopting a looser threshold of 0.1, it is not possible to find any statistically significant difference. Therefore, we conclude that experience does not improve the precision in forecasting the obtainable benefits.

4. DISCUSSION

The rate of expectation among benefits varies considerably. The most commonly expected are the benefits deriving from a descriptive use of models (e.g., *Improved documentation* and *Design support*) as opposed to those deriving from a prescriptive use of models (e.g., *Improved productivity* and *Shorter reaction time to changes*). This tells us indirectly how practitioners use models and for what.

It is interesting to note how this distinction between the usage of models in a descriptive or a prescriptive way emerges also from the relation between benefit expectations, where two distinct clusters are clearly depicted (Figure 2). These strong relations between benefit expectations suggest us that practitioners are trying to achieve a set of different benefits at the same time. It remains to understand how often those benefits are contrasting and how difficult is to devise MD*

Table 1: Frequency of expectations

Benefit	Freq.	Proportion Estimate	95% C.I.	Fulfillment Rate
Improved documentation	81	77%	(68% , 85%)	68%
Design support	77	73%	(64% , 81%)	78%
Quality of the software	75	71%	(62% , 80%)	49%
Maintenance support	66	63%	(53% , 72%)	52%
Standardization	64	61%	(51% , 70%)	52%
Improved development flexibility	51	49%	(39% , 58%)	45%
Improved productivity	42	40%	(31% , 50%)	45%
Shorter reaction time to changes	41	39%	(30% , 49%)	37%
Platform independence	32	30%	(22% , 40%)	34%

approaches able to permit the achievements of all those benefits at the same time.

The strongest relation is between *Improved development flexibility* and *Shorter reaction time to changes* (KC = 0.67), the intensity of this relation is so strong that we can deduce the two benefits are either essentially considered synonyms or they are intimately related. The next strongest relation (KC = 0.43) is between *Improved documentation* and *Maintenance support*, this link seems to implicitly confirm the common wisdom about documentation being an enabler of maintenance activities.

The rate of achievement is constantly higher than 50% for benefits of descriptive models while it is much lower for benefits of prescriptive models. In the latter case, the rate of achievement can be as low as one out of three for *Platform independence* and slightly higher for *Reactivity to changes* and *Improved flexibility*. A few pragmatic questions arise from the perspective of a project manager, that deserve further investigation:

- is it reasonable to expect those less fulfilled benefits from the adoption of modeling and MD*?
- what are the possible causes of low fulfilment rate for those benefits?
  - limited experience in modeling,
  - lack or inadequacy of tools,
  - simply not obtainable through MD* approaches.

In Table 2 we show, side by side, the position of each benefit among the rank of the most expected benefits (Table 1, 2nd column) and the rank of the most reliably predictable benefits (Table 1, last column). As can be seen, the two rankings are very similar, with most expected benefits being also the most reliably predictable, and the least expected being also the least reliably predictable.

The only relevant difference involves *Quality of the software* and *Standardization*. The former is the 3^{rd} more frequently expected benefit but it proved to be not so easily attainable, while the relation is inverted for the latter. Therefore we can say that concerning the improvements of the software quality through the usage of modeling there are greater expectations than it is realistic, while the benefits in terms of standardization are generally underestimated.

Finally, the lack of effect of experience on the ability of predicting the outcome could be due to the immaturity of model-driven techniques, which are still evolving. Is it possible that developers who have more experience rely on assumptions which were valid for old-fashioned model-driven approaches and are not more valid with the most recent ones.

5. RELATED WORK

In the literature is possible to find anecdotal reports of inflated expectations on software by stakeholders [2]. High expectations and consequent disillusion were reported also for other highly-hyped approaches, as for example for agile methods [5]. We believe this is true also in the SOA context [14].

Table 2: Comparison between expectations and rate of achievement.

Benefit	Exp.	Rate ach.
Improved documentation	1°	2°
Design support	2°	1°
Quality of the software	3°	5°
Maintenance support	4°	4°
Standardization	5°	3°
Improved development flexibility	6°	7°
Improved productivity	7°	6°
Shortened reaction time to changes	8°	8°
Platform independence	9°	9°

The effects of expertise on forecast of the outcome were proved to be at the best uncertain in different domains. Camerer and Johnson state that in many domains expert judgments is worse than the simplest statistical models [3]. Hammond [7] stated that "in nearly every study of experts carried out within the judgment and decision-making approach, experience has been shown to be unrelated to the empirical accuracy of expert judgments"; such a statement fits very well the findings of our study, and in particular with RQ3.

While in general, expert judgment seems not to work particularly well, in the context of software development, effort estimation conducted by experts outperforms sophisticated formal methods [11]. The reasons provided by Jørgensen in [10] are: (i) the importance of highly context-specific knowledge in software development, (ii) the instability of relationship in software development (e.g., between effort and size) which lead to a very unpredictable field. The effect of expertise on judgment of other aspects of the software development process are rarely studied, as reported by Loconsole and Borstler in [15]. In their work they examine how expectations on requirements volatility matched the actual number of changes, resulting in a lack of statistical correlation between the expectation and the real outcome.

We have no data for explaining why it is so difficult forecasting the benefits of modeling and MD*. We can only report the work from Shanteau and Stewart [18]; they suggest that experts rely on heuristics in making judgments that could lead to systematic biases.

6. CONCLUSIONS

In conclusion, the results of this survey reveal that:

RQ1: *Improved documentation* and *Design support* are the most expected benefits from practitioners using modeling and/or MD*. Also *Quality of the software*, *Maintenance support*, and *Standardization* are frequently expected. On the contrary, other important benefits, such as *Improved productivity* and *platform independence*, are not so much expected. That result tell us, indirectly, for which reason IT practitioners use models.

RQ2: The benefits having the highest fulfilment rate are still *Improved documentation* and *Design support* (Fulfilment rate > 65%). However, considering all the benefits the average fulfilment rate is not high.

RQ3: Experience in modeling does not help in forecasting the benefits.

Probably the expectations are currently inflated by the amount of hype around MD*. It is possible that in the future practitioners will learn to focus on a smaller set of benefits and they will be able to actually achieve them more reliably.

All in all, this uncertainty about the outcomes of modeling and the fact that it affects also practitioners with many years of experience in the field is probably hampering the adoption of these approaches, which are always predicted to become mainstream in a never reached next future.

As a future work, it could be interesting to understand how much of the difficulty in forecasting the benefits of modeling and MD* depends on the immaturity of those approaches. Is that difficulty inherent in experts' judgement or is it worse in this particular field?

7. REFERENCES

[1] P. Baker, L. Shiou, and F. Weil. Model-driven engineering in a large industrial context - Motorola case study. In L. Briand and C. Williams, editors, *Model Driven Engineering Languages and Systems*, volume 3713 of *Lecture Notes in Computer Science*, pages 476–491. Springer Berlin / Heidelberg, 2005.

[2] B. Boehm. The art of expectations management. *Computer*, 33(1):122 –124, jan 2000.

[3] C. F. Camerer and E. F. Johnson. The process-performance paradox in expert judgment: How can the experts know so much and predict so badly? In K. A. Ericsson and J. Smith, editors, *Towards a general theory of expertise: Prospects and limits*. Cambridge University Press, 1991.

[4] T. Dowling. Are software development technologies delivering their promise? In *IEE Colloquium on "Are Software Development Technologies Delivering Their Promise?"*, pages 1–3, mar 1995.

[5] H. Esfahani, E. Yu, and M. Annosi. Capitalizing on empirical evidence during agile adoption. In *Agile Conference (AGILE), 2010*, pages 21 –24, aug. 2010.

[6] 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.

[7] K. R. Hammond. *Human Judgment and Social Policy: Irreducible Uncertainty, Inevitable Error, Unavoidable Injustice*. Oxford University Press, USA, Oct. 2000.

[8] J. Hossler, M. Born, and S. Saito. Significant productivity enhancement through model driven techniques: A success story. In *IEEE International Enterprise Distributed Object Computing Conference (EDOC '06)*, pages 367–373, oct. 2006.

[9] A. Jelitshka, M. Ciolkowski, C. Denger, B. Freimut, and A. Schlichting. Relevant information sources for successful technology transfer: a survey using inspections as an example. In *First International Symposium on Empirical Software Engineering and Measurement, 2007. (ESEM 2007)*, pages 31–40. IEEE, September 2007.

[10] M. Jørgensen. Estimation of software development work effort:evidence on expert judgment and formal models. *International Journal of Forecasting*, 23(3):449–462, 2007.

[11] M. Jørgensen and S. Grimstad. Software development effort estimation: Demystifying and improving expert estimation. In O. L. Aslak Tveito, Are Magnus Bruaset, editor, *Simula Research Laboratory - by thinking constantly about it*, chapter 26, pages 381–404. Springer, Heidelberg, 2009.

[12] 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.

[13] A. G. Kleppe, J. Warmer, and et al. *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison-Wesley Longman Publishing Co., 2003.

[14] M. Leotta, F. Ricca, M. Ribaudo, G. Reggio, E. Astesiano, and T. Vernazza. SOA Adoption in the Italian Industry. In *Proceedings of 34th International Conference on Software Engineering (ICSE 2012)*, pages 1441–1442. IEEE, 2012.

[15] A. Loconsole and J. Borstler. Are size measures better than expert judgment? an industrial case study on requirements volatility. In *Software Engineering Conference, 2007. APSEC 2007. 14th Asia-Pacific*, pages 238 –245, dec. 2007.

[16] S. Mellor, A. Clark, and T. Futagami. Model-driven development - guest editor's introduction. *Software, IEEE*, 20(5):14 –18, sept.-oct. 2003.

[17] D. C. Schmidt. Guest editor's introduction: Model-driven engineering. *Computer*, 39:25–31, 2006.

[18] J. Shanteau and T. R. Stewart. Why study expert decision making? some historical perspectives and comments. *Organizational Behavior and Human Decision Processes*, 53(2):95–106, 1992.

[19] F. Tomassetti, A. Tiso, F. Ricca, M. Torchiano, and G. Reggio. Maturity of software modelling and model driven engineering: a survey in the italian industry. In *Int. Conf. Empirical Assessment and Evaluation in Software Eng. (EASE12)*, 2012.

[20] M. Torchiano, M. Di Penta, F. Ricca, A. De Lucia, and F. Lanubile. Migration of information systems in the italian industry: A state of the practice survey. *Information and Software Technology*, 53:71–86, January 2011.

[21] M. Torchiano, F. Tomassetti, A. Tiso, F. Ricca, and G. Reggio. Preliminary findings from a survey on the MD* state of the practice. In *International Symposium on Empirical Software Engineering and Measurement (ESEM 2011)*, pages 372–375, 2011.

[22] M. Völter. MD* best practices. *Journal of Object Technology*, 8(6):79–102, 2009.

The Use of UML Class Diagrams and Its Effect on Code Change-proneness

Rut Torres Vargas
Leiden Institute of Advanced Computer Science
Leiden, The Netherlands
r.e.torres.vargas@liacs.nl

Ariadi Nugroho
Software Improvement Group
Amsterdam, The Netherlands
a.nugroho@sig.eu

Michel Chaudron
Leiden Institute of Advanced Computer Science
Leiden, The Netherlands
chaudron@liacs.nl

Joost Visser*
Software Improvement Group
Amsterdam, The Netherlands
j.visser@sig.eu

ABSTRACT

The goal of this study is to investigate the use of UML and its impact on the change proneness of the implementation code. We look at whether the use of modeling using UML class diagrams, as opposed to not doing modeling, relates to change proneness of (pieces of) source code. Furthermore, using five design metrics we measure the quality of UML class diagrams and explore its correlation with code change proneness. Based on an industrial system for which we had UML class diagrams and multiple snapshots of the implementation code, we have found that at the system level the change proneness of code modeled using class diagrams is lower than that of code that is not modeled at all. However, we observe different results when performing the analysis at different system levels (e.g., subsystem and sub subsystem). Additionally, we have found significant correlations between class diagram size, complexity, and level of detail and the change proneness of the implementation code.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics—*Product Metrics*; D.2.10 [**Software Engineering**]: Design—*Methodologies, Representation*

General Terms

Design, Documentation, Measurement

Keywords

Unified Modeling Language, Code Churn, Quality

*Joost Visser is also with the Radboud University Nijmegen, The Netherlands

1. INTRODUCTION

Modeling software systems is believed to give benefits in downstream software development in terms of higher software quality and development productivity. Some research exists that has tried to empirically validate whether such benefits can actually be found — see for example [3][5][10][11][12].

Our study is based on empirical data an industrial software project that is currently in its maintenance phase. In this study we focus on two research questions regarding the effect of UML modeling on maintenance of that software:

- RQ1: Does implementation code that is modeled in UML class diagrams have a higher change proneness than code that is not modeled?
- RQ2: How do UML class diagram metrics relate to change proneness of the implementation code?

Our study is different from the aforementioned previous works in two ways. Firstly, our study looks at change proneness (by means of code churn; i.e. the total number of added and changed lines of code) rather than numbers of defects in evaluating the effect of UML modeling. The assessment of code churn is performed across multiple snapshots of a system. Secondly, we propose a novel way of measuring the quality of a UML model, namely by defining quality metrics at the level of diagrams (rather than individual classes or entire models).

At the same time, we learn from earlier research that software developers focus their modeling effort on classes that are more important and classes that are more critical to the system.

The rest of this paper is organized as follows. In Section 2, we discuss the goal and the design of the study. In Section 3, we present the results of the study, and in Section 4 we further discuss the results and their limitations. Section 5 discusses related work, and finally in Section 6 we outline conclusions and future work.

2. DESIGN OF THE STUDY

In this section we discuss the goal and the setup of the study.

2.1 Goal and Research Questions

The goal of this study according to the GQM template [1 can be formulated as follows:

Analyze the use of UML class diagrams
for the purpose of evaluating its effect
with respect to code change proneness
from the point of view of the researcher
in the context of an industrial software system

Based on the above goal we formulate the following resear questions:

- RQ1: Does implementation code modeled in UML cla diagrams have higher change proneness than not mo eled code?
- RQ2: How do UML class diagram metrics relate change proneness of the implementation code?

2.2 Measured Variables

In this section we explain the variables measured in o study. It is important to mention that in the measureme of class diagrams, the unit of analysis is diagrams. In t measurement of the code, the unit of analysis is classes (i. Java classes).

2.2.1 Measured Variables in RQ1

The type of study we used for answering RQ1 is a qua experiment. A quasi experiment is designed to assess caus impact, but it lacks the random assignment to the treatme groups (i.e., in our study it is the assignment of classes the modeled and not-modeled groups).

Independent Variable. The independent variable in RQ1 is the use of class diagram (UMLCD). UMLCD is a nominal variable that indicates whether a given class in the implementation code is modeled or not modeled in a class diagram. Hence the value of this variable is either 'modeled' or 'not modeled'.

Dependent Variable. The dependent variable is the average relative code churn of an implementation class (AvgRelChurn). Relative code churn of a class is the total number of added and changed lines in a particular class divided by the total lines of the whole system. Because there are multiple versions of the same class, we take the average of relative code churn across versions to represent change proneness in a class. A justification of using relative code churn is reported by Nagappan and Ball [9] who show the superiority of relative code churn metrics over absolute code churn metrics to predict defect density. Although the context of the study conducted by the authors was different from ours, the use of relative code churn is justifiable. Relative code churn takes into account the size of the code base, hence controlling the effect of system size. This is particularly important because multiple system snapshots will be used in the analysis.

Co-factor. Two confounding factors are considered in the analysis, namely code complexity and code coupling. The degree of complexity and coupling of software modules can indicate their change-proneness [2]. As such, we want to control for their effects in order to observe a more pure contribution of using UML class diagrams on code change-proneness. In order to account for the complexity of the source code we take the average percentage of lines of code with a McCabe [8] value above 10 as a confounding factor (RiskyMcCabe). In order to account for coupling in the code we take the percentage of lines of code with fan-in value above 16 (RiskyFanIn).

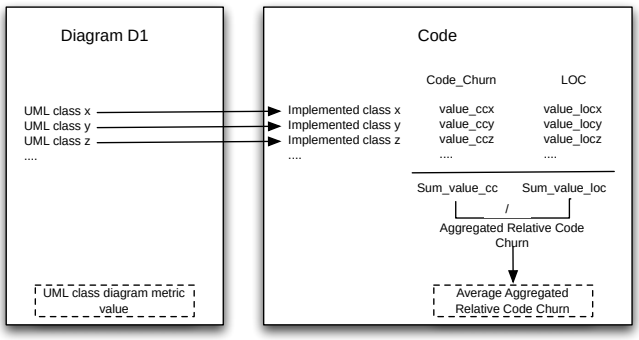


Figure 1: Mapping between class diagrams metrics and the code churn metrics

Note that all code metrics are calculated automatically using the Software Analysis Toolkit (SAT) developed by the Software Improvement Group (SIG). These metrics are automatically calculated for every snapshot of a system and hence the differences in the code metrics across snapshots can be obtained easily.

2.2.2 Measured Variables in RQ2

The design of study to answer RQ2 is a correlational study. Correlational studies do not aim to establish causal relationships. Therefore, in RQ2 there is no distinction in terms of independent and dependent variables.

Based on previous work, we selected five metrics that represent the quality of UML class diagrams. These metrics are calculated automatically using SDMetrics [15].

- Diagram Size (CDSize). Defined as the total number of classes and interfaces in a class diagram. Ambler [1] suggests a rule of thumb that a diagram should contain 7 +/- 2 elements.
- Internal Connectivity (CDIntConn). Defined as the percentage of elements that are relations (associations, generalization and dependencies). This metric measures the complexity of class diagrams and is adapted from metric definition of SDMetrics [15].
- Lonely Classes (CDLoneClass). Defined as the percentage of classes that are not connected with any other class/interface in the diagram. This metric measures cohesiveness of class diagrams and is adapted from metric definition of SDMetrics [15].
- Associations Without Role (CDAscNoRole). Defined as the percentage of associations without role name (adapted from [12]). This metric measures the level of detail in class diagrams.

- Operations Without Parameters (CDAvgOpsNoPar). Defined as the average percentage of operations without parameters in the classes that are part of the diagram (adapted from [12]). This metric also measures the level of detail in class diagrams.

Another measured variable is the average relative code churn (CDAvgRelChurn). This variable measures the average of total code churn over time of a set of implementation classes that are modeled in a single class diagram.

As mentioned previously, the measurement of the code churn is at the class level (Figure 1). Since the class diagram metrics are measured at the diagram level, we follow the next steps to determine CDAvgRelChurn:

1. Map each UML class into the corresponding implementation class.
2. Calculate the total code churn and total lines of code of all implementation classes per diagram.
3. Divide the total code churn by the total LOC, resulting in the relative code churn per diagram.
4. Calculate the average of relative code churn over time per diagram (CDAvgRelChurn).

2.3 Analysis Method

To answer RQ1, classes in the implementation code are divided into two groups: modeled and not modeled. Next we compare the AvgRelChurn between the two groups to check whether there is a difference that is statistically significant. We use the Mann-Whitney test to determine the significance of the difference in AvgRelChurn between the modeled and not modeled groups. In order to account for confounding factors, we perform an Analysis of Covariance (ANCOVA) with the complexity (RiskyMcCabe) and coupling (RiskyFanIn) metrics as co-factors.

To answer RQ2, we perform a correlation analysis between each class diagram metric and code churn (CDAvgRelChurn). We use the Spearman correlation test because our data is not normally distributed. Finally, we perform a multiple regression to account for code complexity and coupling as co-factors.

2.4 Description of the Case Study

The case study is a system for registering business organizations in the Netherlands. The technical quality of the system started being monitored by SIG in May 2010. The development of the system started around July 2008 and the system went live in May 2010. The developed system is replacing an old system, which is still running in parallel. Currently the new system is in maintenance mode but new functionality is still being transferred from the old version. The system is divided in three sub-systems, which we will call A, B and C. The total LOC for the three sub-systems is around 321 KLOC. The programming language used is Java.

In terms of modeling, not all implemented classes were modeled in UML. Only 23 class diagrams are available, and all of them correspond to a sub-part of sub-system A. Figure 3 shows the division of the system into three sub-systems (A, B, C). Furthermore, sub-system A has a set of 22 packages, which we will call sub-A, consists of modeled (the striped part) and not modeled classes. The rest of sub-system A, as well as the whole sub-system B and C consist of packages of not modeled classes.

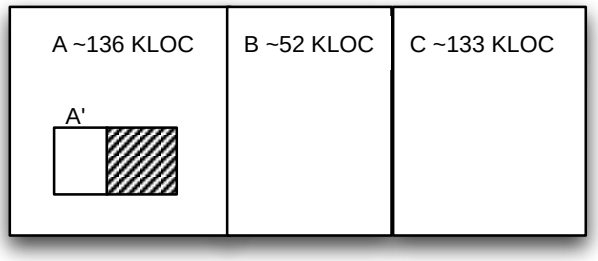


Figure 3: Division of the system into subsystems

In total there are 100 snapshots of this system in the software repository. Among the code metrics being monitored are code churn, code complexity and coupling.

3. RESULTS

3.1 The Use of Class Diagram and Its Impact on Code Change-proneness

The analysis to compare change proneness between the modeled and not modeled classes in the case study is performed at three levels: sub subsystem A', subsystem A, and the whole system (Figure 3). Figure 2 show the boxplots of AvgRelChurn of the modeled and not modeled for each of the three areas of comparison (sub subsystem A', subsystem A and system).

Looking at the median in Figure 2 (bold horizontal lines), we can observe that in the first two cases (sub-system A' and sub-system A), on average, modeled classes change more than not modeled classes, while in the third case (system), not modeled classes change more. In order to determine if the difference in AvgRelChurn is significant between the modeled and not modeled classes, we perform the Mann-Whitney test. The results of Mann-Whitney test show that the difference in AvgRelChurn in the three analyses is statistically significant ($p \leq 0.01$).

However, the fact that modeled implementation classes have higher or lower change-proneness might also be explained by other factors such as the complexity of the code. To account for such confounding factors we conduct an analysis of covariance (ANCOVA) considering the complexity and coupling of the code as co-factors. From the ANCOVA analysis the Modeled/Not Modeled variable is still significant for the sub-system A and system area, but not for the sub-system A' area. Also, it is important to mention that the RiskyMcCabe metric is not significant in any case, and the RiskyFanIn metric is significant only in the sub-system A area.

The different results about the relation between the use of

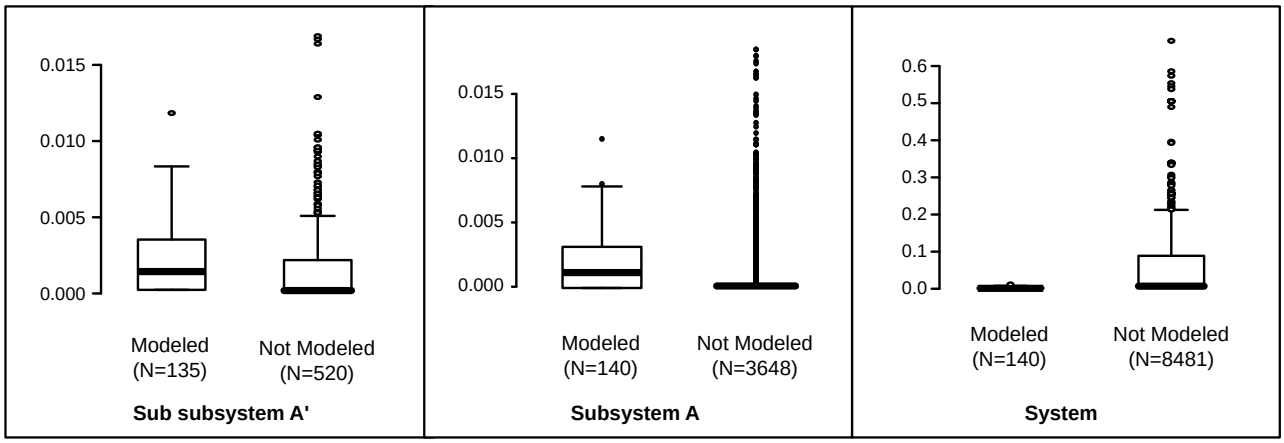


Figure 2: Boxplots of AvgRelChurn comparing modeled and not modeled classes in sub subsystem A, subsystem A, and the whole system

Table 1: Results of the Spearman correlation analyses between class diagram metrics and code change proneness (CDAvgRelChurn)

	r	p-value
CDSize	-0.476	0.039*
CDIntConn	-0.488	0.033*
CDLoneClass	-0.085	0.727
CDAscNoRole	0.494	0.031*
CDAvgOpsNoPar	0.241	0.318

** indicates significance at 0.05 level*

class diagrams and code change proneness presented above might indicate the influence of sampling.

3.2 The Relations between UML Diagram Metrics and Code Change-proneness

We explore the relations between five UML class diagram metrics and change proneness of the respective code, i.e. the implementation classes that are part of the UML class diagram. We run the Spearman correlation test between each one of the selected class diagram metrics and CDAvgRelChurn. The results of the correlation test are shown in Table 1.

The results of the correlation analyses in Table 1 show there are three significant correlations:

- CDSize correlates negatively with CDAvgRelChurn: the bigger the size of a class diagram, the less change prone the implementation classes that are part of that diagram.
- CDIntConn correlates negatively with CDAvgRelChurn: the more complex a class diagram, the less change prone the implementation classes that are part of that diagram.
- CDAscNoRole correlates positively with CDAvgRelChurn: the more detailed a class diagram is modeled, the less change prone the implementation classes that are part of that diagram.

The previous correlation analyses do not take into account confounding factors that can influence the change proneness of the code. For this reason, we perform a linear regression analysis taking the code complexity and coupling as co-factors. As with the CDAvgRelChurn metric, we need to aggregate the RiskyMcCabe and RiskyFanIn metric at the diagram level—referred to as AvgRiskyMcCabe and AvgRiskyFanIn respectively. We perform three multiple regression analyses with each of the three UML metrics (CDSize, CDIntConn and CDAscNoRole) as a predictor, and take into account the complexity and coupling of the code in the respective analysis. The results show that only CDSize ($p = 0.049$) and CDAscNoRole ($p = 0.026$) remain significant predictors of code churn.

4. DISCUSSION

4.1 Interpretation of Results

With respect to RQ1, we observe different outcomes as a consequence of taking different scopes of analysis. Going from smaller to larger system scope, we have compared modeled versus not modeled classes at the level of sub subsystem, subsystem, and system (see Figure 3).

At the level of sub subsystem, we have found that there is no significant difference in the average relative code churn of modeled and not modeled classes. This result can be explained by the fact that classes that are part of the same package probably change together. Therefore, the fact that a class has been modeled or not will not make a difference in the change proneness of the code. At the level of subsystem, we found that there is a significant difference in the average relative code churn of modeled and not modeled classes. However, modeled classes change more than not modeled ones. This result contradicts our assumption that modeled classes will change less due to the positive impact of modeling on understandability of the system. Higher understandability will lead to a correct implementation from the beginning that leads to fewer rework. At the system level, we have found that there is a significant difference in the average relative code churn of modeled and not modeled classes, and not modeled classes change more than modeled ones. This result confirms our assumption that modeled

classes will change less than not modeled ones, but is not consistent with the result from the subsystem level.

The differences in the above results can be explained by the representativeness of the classes in the sample. The result of analysis at the same sub subsystem does not show a significant result because the classes are quite similar in type (e.g., data classes), complexity and importance. At the level of a sub system, there might be more diverse class types and hence modeled classes might be the classes that are significantly more important or critical to the system, therefore the higher code churn. At the system level, the diversity of classes is even higher and it is very likely that there are more critical and important classes that are not modeled. This might explain why in the analysis at the system level not modeled classes have significantly higher code churn. Including all classes of the system in the analysis increases the representativeness of the data set and therefore gives more reliable results.

With respect to RQ2 our results show that three out of five UML class diagram metrics have significant correlations with the average code churn of the implementation classes. However, after accounting for the effect of code complexity and coupling, only two of those metrics remained significant, namely CDSize and CDAscNoRole.

The fact that CDSize metric has a negative correlation with code churn suggests that classes that are part of bigger class diagrams tend to change less often than classes that are part of smaller class diagrams. In the context of software maintenance, this result may not be very surprising. Big class diagrams indicate poor modularization, which typically leads to tightly coupled and incomprehensible classes. It is not uncommon that maintainers tend to avoid changing brittle parts of a system and thus changes are made around these parts instead. However, further validation of this finding is required.

With respect to the CDascNoRole metric, the result shows a significant and positive correlation. This finding suggests that classes that are part of less detailed diagrams (in terms of detail in the associations) tend to change more than classes in more detailed diagrams. This finding supports the idea that level of detail in UML diagrams is beneficial to the understanding of the system to be implemented [10]. A higher understandability will lead to fewer changes since the system has been correctly implemented from the beginning.

4.2 Threats to Validity

Construct Validity. The threat to the construct validity in this study is mainly related to the variable chosen to measure change proneness of the code in RQ2. The average relative code churn (CDAvgRelChurn) is measured for a set of implementation classes that appear in a particular class diagram. In this case, we assume that the change proneness of classes modeled in a diagram is influenced by the quality of that diagram. We are aware that this assumption has some limitations, particularly if some classes appear in a class diagram for trivial reasons (they should have been modeled elsewhere). Further investigation is needed to find better ways to map UML diagram quality properties to code quality properties.

Internal Validity. Internal validity in this study comes from differences in the nature of the implemented classes. Even when all classes are part of the same system, some of them can be considered more critical than others and prone to change more. We try to address this threat by taking into account confounding factors such as complexity and coupling of the code. However a future study can go further by asking the development team to classify each class according to its criticality following a predefined scale. Additionally, there are some cases where classes in the UML model can not be mapped to any of the implementation classes. However, in our study this issue does not occur very often and hence is not expected to introduce serious bias.

External Validity. Threats to external validity come from the limitation to generalize the results because of the use of a single case study. Although based on a single case study, the use of a real industrial system as a case study increases our confidence about the generalizability of the results. Further replications of this study using more industrial systems will help validate the results.

Conclusion Validity. Conclusion validity refers to the ability to draw correct conclusions from an experiment. We have addressed this threat by using carefully selected statistical tests to determine the significance of the results. We have also considered the applicability of each test in order to avoid violation of assumptions.

5. RELATED WORK

Previous works that looked into the impact of UML documentation on software development have focused primarily on model comprehension and system comprehension—see for example in [4][6][7]. Because the focus of this study is on the effect of UML modeling on software maintenance, in the following passages we focus on previous work that studied the effect of UML modeling on software maintenance.

The work by Tryggeseth [13] explored how the use of textual system documentation (requirements specification, design document, test report, user manual) affects the time needed to understand how to perform maintenance tasks. The results show that the time needed is reduced by approximately 20 percent when documentation is available.

The work by Arisholm et al. [3] investigated the impact of UML availability on the maintenance of source code. The authors performed two experiments using students with knowledge on UML and programming. The results show that UML has a significant, positive impact to increase the functional correctness of code changes. However in terms of time saving, there were no significant benefits when the time to modify the UML model is included.

The work by Fernandez-Saez et al. [5] compared how UML diagrams with different level of detail influence the maintenance of the source code. The results of the experiment carried out with 11 students showed better results when using low level of detail UML models. The authors however did not consider the results significant due to the small size of the group of subjects and their lack of experience in using UML and Java code.

Another closely related work was performed by Nugroho and Chaudron [11]. The authors investigated the impact of using UML (class and sequence diagrams) on defect density of the implementation code. Based on an industrial case study, the authors found that classes modeled in either class or sequence diagrams have significantly lower defect density compared to classes that are not modeled at all. Another work by the authors defined level of detail metrics for UML class and sequence diagrams [12]. Based on data from an industrial software system, the authors found that higher level of detail in sequence diagrams corresponds to lower defect density in the implementation code.

The current paper extends the above previous works by investigating the effect of using UML on code change proneness in an industrial case study.

6. CONCLUSION AND FUTURE WORK

In this paper we report our exploratory investigation into the impact of the use of UML on the change proneness of the implementation code. Our study aim to answer two research questions:

- RQ1: Does implementation code modeled in UML class diagrams have higher change proneness than not modeled code?
- RQ2: How do UML class diagram metrics relate to change proneness of the implementation code?

To answer RQ1, we compare the average relative code churn of classes that have been modeled in class diagrams and those that have not been modeled. This comparison was made at three different levels: sub subsystem, subsystem and system. At the system level, we have found a significant difference with not modeled classes changing more than modeled ones. However, the results vary according to the level of comparison, which we suspect is due to the representativeness of the sample at the respective level.

To answer RQ2, we use five UML class diagram metrics defined in previous works and explore their correlations with code change proneness. The five metrics are class diagram size (CDsize), class diagram internal connectivity (CDIntConn), class diagram lonely classes (CDLonClass), class diagram operations without role (CDAscNoRole) and class diagram operations without parameters (CDOpsNoPar). Three of these metrics show significant correlations with the change proneness of the code. After accounting for the effects of code complexity and coupling, only two of the metrics remained significant, namely CDSize (positive correlation) and CDAscNoRole (negative correlation).

We are aware that this study requires further validations in future research. An obvious future work is to replicate this study using more industrial systems. Further refinement of this study is to separate the analysis based on diagram types—that is, to evaluate the difference in code churn or other code quality indicators based on the types of diagram used in modeling the system (e.g., class versus sequence diagrams). Finally, we also aim to investigate the effect of UML diagram quality on the productivity in performing maintenance activities such as bug fixing.

7. REFERENCES

[1] S. Ambler. *The elements of UML 2.0 style*. Cambridge Univ Pr, 2005.

[2] E. Arisholm, L. Briand, and A. Foyen. Dynamic coupling measurement for object-oriented software. *Software Engineering, IEEE Transactions on*, 30(8):491–506, 2004.

[3] E. Arisholm, L. Briand, S. Hove, and Y. Labiche. The impact of uml documentation on software maintenance: An experimental evaluation. *Software Engineering, IEEE Transactions on*, 32(6):365–381, 2006.

[4] L. Briand, Y. Labiche, M. Di Penta, and H. Yan-Bondoc. An experimental investigation of formality in uml-based development. *Software Engineering, IEEE Transactions on*, 31(10):833–849, 2005.

[5] A. Fernández-Sáez, M. Genero, and M. Chaudron. Does the level of detail of uml models affect the maintainability of source code? In *Proceedings of EESSMod 2011*, 2011.

[6] M. Genero, J. Cruz-Lemus, D. Caivano, S. Abrahão, E. Insfran, and J. Carsí. Assessing the influence of stereotypes on the comprehension of uml sequence diagrams: A controlled experiment. *Model Driven Engineering Languages and Systems*, pages 280–294, 2008.

[7] L. Kuzniarz, M. Staron, and C. Wohlin. An empirical study on using stereotypes to improve understanding of uml models. In *Program Comprehension, 2004. Proceedings. 12th IEEE International Workshop on*, pages 14–23. IEEE, 2004.

[8] T. J. McCabe. A complexity measure. In *Proceedings of the 2nd international conference on Software engineering*, ICSE '76, pages 407–, Los Alamitos, CA, USA, 1976. IEEE Computer Society Press.

[9] N. Nagappan and T. Ball. Use of relative code churn measures to predict system defect density. In *Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on*, pages 284–292. IEEE, 2005.

[10] A. Nugroho. Level of detail in uml models and its impact on model comprehension: A controlled experiment. *Information and Software Technology*, 51(12):1670–1685, 2009.

[11] A. Nugroho and M. Chaudron. Evaluating the impact of uml modeling on software quality: An industrial case study. *Model Driven Engineering Languages and Systems*, pages 181–195, 2009.

[12] A. Nugroho, B. Flaton, and M. Chaudron. Empirical analysis of the relation between level of detail in uml models and defect density. *Model Driven Engineering Languages and Systems*, pages 600–614, 2008.

[13] E. Tryggeseth. Report from an experiment: Impact of documentation on maintenance. *Empirical Software Engineering*, 2(2):201–207, 1997.

[14] C. Wohlin, P. Runeson, M. Host, C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in Software Engineering: an Introduction*. Kluver Academic Publishers, 2000.

[15] J. Wüst. SDMetrics: the Software Design Metrics tool for the UML, 2012.

Lessons Learned from Evaluating MDE Abstractions in an Industry Field Study

Adrian Kuhn
Software Practices Lab
Institute for Computer Science
University of British Columbia

Gail C. Murphy
Software Practices Lab
Institute for Computer Science
University of British Columbia

ABSTRACT

In a recent empirical study we found that evaluating abstractions of model-driven engineering (MDE) is not as straight forward as it might seem. In this paper, we report on the challenges that we as researchers faced when we conducted the aforementioned field study. In our study we found that modeling happens within a complex ecosystem of different people working in different roles. An empirical evaluation should thus mind the ecosystem, that is, focus on both technical and human factors. In the following, we present and discuss five lessons learnt from our recent work.

1. INTRODUCTION

The promise of abstractions such as model-driven engineering (MDE) is that the representations they provide to engineers are semantically more similar to the domain by hiding away implementation details. Yet, in a recent field study we found that identification and evaluation of these abstractions is not as straight forward as it might seem. Our work was an ethnographic study of MDE adoption from an engineer's perspective [7], we did in-depth interviews with 20 engineers from the automative industry to learn how they succeed and struggle with model-driven techniques. While we present a brief overview of our study in Section 2, in this paper we focus on the challenges that we—as researchers—faced.

When doing our interviews we found it quite hard to capture the essence of abstractions. In the wild abstractions do not come as text book figures. We faced fundamental challenges such as, when are we looking at an abstraction and when not? When is an abstraction higher, i.e., more abstract, than another? Does more abstraction offer more or less flexibility? What about abstractions that makes engineers more productive but are harder to understand? Or, an abstraction that is more understandable but, at times, leaky and thus leads to bursts of overhead work. Has it possibly been designed to solve another problem than what it is currently used for at this site of study? Or maybe even, are we looking at an organizational rather than a technical abstraction, i.e., shift of responsibility between roles rather than between representation levels? Or, both? Essentially, how to identify an abstraction and how to evaluate the quality of that abstraction?

Also we realized, engineers are not necessarily aware at which level of abstraction they are working. To them, what they do is their daily work. They might not even be aware of possible other abstraction layers below or above their work. So, asking them about abstraction levels often does not make sense in their language. For example, none of the engineers that we interviewed referred or thought of to their work as modeling, while in fact they were using model-driven engineering techniques.

As we conducted our field study we continuously improved our questionnaire as we gained more insight into the challenge of studying abstractions. In the following we summarize our lessons learned:

- Quantitative approaches might fail, often abstractions are not "more of the same" but "something different" and thus not covered by established metrics.
- While qualitative approaches are better suited, they might still miss parts of the picture unless the complete ecosystem is taken into account.
- Recent adopters make for great interview partners, to them abstractions are still fresh and not yet daily business. Caution with learning curves is advised tough.
- Asking for examples of most recent work items is paramount to avoid bias, or else a field study risks reporting prescribed processes rather than actual practices.
- Focus on communication patterns rather than artifact flow, as artifact flow is a subset of communication.

These lessons learned are obviously based on a single case study only, however we hope that they might serve as the base for a discussion at the EESMOD workshop of how to best evaluate model-driven abstractions in field studies.

The reminder of this paper is structured as follows: In Section 2 we discuss our choice of methodology, and provide a brief overview of our industry case study. In Section 3 we discuss the lessons learned that are listed above. In Section 4 we outline related work. Eventually, in Section 5 we conclude with a summary.

2. OUR CASE STUDY IN A NUTSHELL

Our research is aimed at understanding cognitive issues in model-driven engineering, with a focus on information and communication needs from the perspective of individual engineers. As we started our research we had little background in model-driven engineering and faced the choice of an appropriate research methodology.

Quantitative approaches are warranted for scenarios where the state of the art provides a deep understanding of the domain under study and can refer to established means of measuring the phenomena under study. In our case, a literature review confirm a gap in research, we thus decided to go with an approach which is both qualitative and exploratory.

To enable the gathering of detailed, rich and contextual information about model-driven engineering, we chose a qualitative study approach. We interviewed people working with model-driven engineering technology in semi-structured interviews, following an exploratory case-study approach where open ended questions are asked in order to refine the research hypothesis as the current study is ongoing.

Exploratory approaches are great when you, as a researcher, start with little understanding of the domain under study. Grounded theory has recently gained popularity in the empirical software engineering community as an exploratory approach [1]. However, we choose to do an exploratory case study, where we are not necessarily looking for "one theory" but are facing a multifaceted case with possibly many theories explaining the observations.

An advantage of exploratory user studies, both grounded theory [2] and exploratory case studies [8], is that, as a researcher, you are able to adapt your questionnaire as you learn from the participants answers. For example, as we started our user studies we had questions about "how many models" a person is responsible for. Though we quickly learned that, unlike to us academics, to the engineers "model" was not a countable term. The question did make as much as to them as asking how many "softwares" a software engineer in a traditional team is working on.

In our interviews, we found that the terms "model" and "modeling" were used ambiguously. Engineers generally did not refer to their work as "modeling" but used the terms "auto-coding" and "hand-coding." These terms were used to differentiate between working with tools which include a step of code generation versus writing C-level code manually. Engineers used the term "model" ambiguously to refer to software models, as well as the *plant models* used for the in-silico simulation of vehicles. Engineers also used the term "simulation" ambiguously to refer to running the in-silico simulation of the plant models, as well as to running software models from within the modeling tools as opposed to running the auto-generated sources.

We believe the terminology we observed is mixing model-based design (MBD, an approach in system engineering for disentangling the development of control software and corresponding vehicles, using in-silico modeling while vehicles are not yet available) and model-driven engineering (MDE). The ambiguous use of terminology can be explained if we look at model-driven engineering as a division of labour between a few specialized language designers and many modelers. After all, the software engineers do not have to understand the full complexity of modeling, this is up to the specialized code-generation engineers. However, we found that points of friction in modeling tools, in particular the insufficient support of model diffing, may break the abstraction and nevertheless expose engineers to these complexities.

In the industry case study we interviewed 20 people working with model-driven approaches at General Motors, a large automotive company that heavily relies on model-driven engineering for their software development. The study is going to appear in the proceedings of MODELS 2012, the hosting conference of this workshop[1].

We found that, in the context of a large organization, contextual forces dominate the cognitive issues of using model-driven technology. The four forces we identified that are likely independent of the particular abstractions chosen as the basis of software development are the need for diffing in software product lines, the needs for problem-specific languages and types, the need for live modeling in exploratory activities, and the need for point-to-point traceability between artifacts. We also identified triggers of accidental complexity, which we refer to as points of friction introduced by languages and tools. Examples of the friction points identified are insufficient support for model diffing, point-to-point traceability, and model changes at runtime.

In the following we are going to focus on the setup and setting of the case study as far as it is of interest for the present paper. The study consisted of interviews with 20 engineers and managers working in different roles. We visited the industry of interest (General Motors) on two separate occasions, collecting data constructed through semi-structured in-depth interviewing. We interviewed 12 engineers and 8 managers. Overall, the engineers we interviewed came from four different teams from different company departments. All teams were global, that is spread across sites in India and America, however we interviewed people from the American sites only. The 12 engineers selected for interviews were sampled from several roles however their profiles are similar, that is they all work with the same process and use the same modeling technology. Each interview was 90–120 minutes long, recorded on tape and transcribed for encoding by one of the authors of this paper.

In a first visit, we interviewed 10 participants from both management and technical roles to familiarize ourselves with the software process used in the automotive industry. Based on what we learned from the first interviews, in our second visit, we interviewed an additional 10 participants, all of them working with software models but in different roles. The interviews were semi-structured, following an exploratory case-study approach where open ended questions are asked in order to identify research hypothesis for future studies. We asked participants to describe their work, how their work fits into the process of the organization, with whom they interact on a weekly basis, and which artifacts are the input and which are the output of their work. We also asked to see

[1]Preprint available at `http://arxiv.org/abs/1207.0855`

current or recent examples of artifacts on which they were working.

We transcribed the 12 interviews with engineers (4 from the first visit and 8 from the second visit). We encoded the transcripts and from this encoding, we distilled the contextual forces and points of friction presented in this paper. We encoded the interviews by tagging sentences with hashtags as if they were tweets. We then used a series of tag clouds to identify patterns in the data, merging and splitting tags as we saw need. We did two passes over the tags, a first one to identify all forces and frictions that shape the work of the participants, and a second pass to identify forces and frictions that might provide the basis for general hypotheses on model-driven engineering, ruling out those that are specific to the organization under study.

3. LESSONS LEARNED

In this section we share our lessons learnt of assessing abstractions in the wild. The nature of abstractions is difficult to qualify. There is value but often it is faceted, the same tool might be both more and less abstract. Sometimes providing better abstractions to one role but worse abstractions to another role, sometime to the same role. We found that, in the context of a large organization, contextual forces dominate the cognitive issues of using model-driven technology. While some abstractions are technical and applied locally by individuals, such as information hiding in programing language, MDE happens within a complex ecosystem of different people working in different roles, or even different parts of an organizations.

In the following we present the main lessons learned from conducting our study.

Finding *Quantitative approaches might fail, often abstractions are not "more of the same" but "something different" and thus not covered by established metrics.*

Novel abstractions, by their definition of raising representations to be more semantic and of hiding away details, are disrupting established quantitative metrics. So, how to establish good metrics for a novel abstraction? Designing new metrics requires a understanding of the domain, both technical and semantic. An understanding that is typically best gained by first doing qualitative and exploratory studies.

It might seem obvious that KLOC (lines of code) are an inappropriate metric to compare the complexity of manually written and auto-generated code. Yet the difficulty of assessing abstractions using quantitative methods are much more subtle. While increasing the abstraction of one process step might positively impact that one step, work further downstream in the process might suffer negative benefits. For example, cost being gained by reduced development times might be outweigh by increased cost of certification, as has been found in a study by Hutchinson *et.al.* [4, 5].

In a similar way, time gained by spending less time in one tool might be lost again by time spent in a novel tool or even doing unexpected ad-hoc workarounds. In our case study we have found that engineers spend significant time manually putting together screenshots of model changes in Microsoft Powerpoint in order to email them as a "change set" to other engineers. A job which has been automated using text-based diffing tools before the introduction of MDE abstractions.

Finding *While qualitative approaches are better suited (than quantitative ones), they might still be miss parts of the picture unless the complete ecosystem is taken into account.*

Participant will not report *"see, part of my work is done by somebody else"* since to them it not their work anymore when somebody else does it. Yet, when looking at adoption of higher abstractions, such as MDE ,very often it is the case that part of somebody's work is now done by another role. Abstraction might move work from one role to another or even from one team to another. An empirical study of model-driven engineering should thus mind the ecosystem and look into how details are abstracted away across roles and organizational units rather than just across technical boundaries.

Sometimes, a comparison is being made between rhe shift from source to models in MDE and the shift from assembly to high-level languages. We found that this is not a useful analogy for designing empirical studies of MDE. The abstractions introduction when moving from assembly to high-level code are typically hiding away details which are taken care of by the machine. Consider for example garbage collection, where the machine takes care of allocating and freeing memory, which has previously been the engineer's burden. This is not the case of many of the abstractions introduced by MDE. Code generation is unlike code compilation. In order to provide engineers with domain-specific abstractions, the implementation details that are hidden from engineers must be taken care of by other, more specialized roles, rather than the machine. In our case study we found that a team of specialized code generation experts is responsible for defining and managing these abstractions.

Finding *Recent adopters make for great interview partners, to them abstractions are still fresh and not yet daily business. Caution with learning curves is advised tough.*

Recent adopters are make for great interviews partners. They are much more aware of abstraction levels, since to them the abstractions their working with are novel and thus they are still aware of what improved and what worsened with introducing these novel abstractions. They were still in the state of comparing the Now (model driven engineering with Simulink and Rhapsody) and the Then (hand coding in machine level languages). Yet, caution is advised as they might be in a learning curve and some of their observations are due to that learning curve. But that is exactly our job as ethnographic researchers, taking a step back and being able to do this kind of analysis.

The very definition of abstraction means that we might prefer to ask certain kinds of questions about abstractions introduced to people who can compare how a similar things was built before, i.e., they need to know what might be hidden now. More importantly, as we pointed out before, we need to ask how did people work before and what did the people

versus the machine do compared to after the abstractions are introduced.

On our second visit we had the chance the include a team in our study who had only recently adopted model-driven engineering. Interviewing this kind of recent adopters, offers a unique window of opportunity onto comparing the "before" and "after" of an abstraction's introduction. Other than engineers who had been using model-driven technology since their training, these people can compare how work has been done before and after the introduction of model-driven abstraction.

A danger is though that these people might report negatively about many novel aspects of the abstractions as they are still on a learning curve of adopting new ways of getting work done. We found that it helped to have both recent adopters as well as experienced users of model-driven engineering in the same sample.

Finding *Asking for examples of most recent work items is paramount to avoid bias, or else a field study risks reporting prescribed processes rather than actual practices.*

A quantum leap in our understanding of model-driven engineering happened as we started to ask participants for concrete examples. This is a well-known lessons learned for many empirical researchers. If you, as a researcher, ask people general questions about how they get their job done, answers by participants tend to remain equally general, typically describing an idealized account of how their job should be done rather how their job is actual done when the "rubber hits the road."[2]

Examples. Show us the most recent model (or other artifact in question) that you were working with. This focus on the most recent work is important to avoid the participant's filter bias. If not asked for the most recent artifact, or the most recent week, we risk of being presented with ideal-case examples rather than actual samples from their daily work. And they will look different than you as researcher expect! For example we asked engineers: do you have tests? Yes. Do you have repeatable tests? Yes. Are they automated? Yes. So we expected tests that are akin to unit testing practices known from software engineering. But when we asked them to show us their most recent test, they opened an Excel sheet with instructions to a human tester and with a fields to enter what they see on screen. Obviously repeated and automated, but not the way we expected!

For example, here is a lesson learned. After two visits of ten interviews each, we realized that we missed a whole population stakeholders, the code-generation experts. We interviewed product engineers and their leads but did reach out to that one specialized team of code-generation experts. We were neither aware that they exist nor which crucial role they play in the model-driven engineering process. Mainly because neither their job title gave any clue of their work, nor were they included in the process model that was presented to us. We only learned about them when we started to check in with wrap-up questions, e.g. "Were there more people to whom you talked last week that are not yet on this list?," after having gone step by step through all communications and meetings of a participant's most recent week. So even in our participant's heads these specializes were so much outside of their process that they did not think of them in the first place. Yet, they are key to understanding the MDE adoption and its benefits. We thus plan to interview these people in future work.

We started to ask questions like "tell us more of about the most recent change ticket your worked on" and then continued with "can you please show us the model that you changed for that ticket" and then "can you show us the exact changes you did to that model." This way we made sure to learn about the actual work being done, and not hearing twenty times about the same formally defined process that it has been presented to us on our first briefing.

It is very important to narrow down this kind of questions to the most specific examples possible, such as "most recent change ticket you worked on" or "people you talked to last week," and then following them up with questions such as "were there more people you talked to last week that had not been mentioned so far" in order to make catch missing data. It is typically this latter kind of catch-up questions that reveal the most interesting tidbits of information.

Of course it may happen the that the "most recent change ticket" or "last week" had been exceptional and do not to serve as good exemplars. We controlled for this using questions like "had last week been a typical week of your work." Often the reply had been one of "yes it had been typical expect for..." which again lead to very interesting insight into their typical work as we learned which kind of work participants consider untypical.

Finding *Focus on communication patterns rather than artifact flow, as artifact flow is a subset of communication.*

In the begin our questionnaire focused on artifact flow, because we figured that would be a great source of learn about all parts of the ecosystem that are involved in MDE. But the we realized, focusing on people and communication is a much better approach: we started going with them through all meetings and communication of the most recent week, asking for each communication about every participant's role and how they are connected to each other. It was this information which proved to provide a complete picture of the MDE ecosystem, including artifact flow. We thus conclude that artifacts flow is a subset of communication only, and thus communication a better means to learn about abstractions and their adoption.

On our second visit we started to include questions such as "whom did you talk to last week" in our interviews. We asked participants to walk us through their most recent week of interaction with other people. This lead to most interesting insight about the organizational perspective of model-driven engineering abstractions. For example we learned about teams and organization groups that had not been captured by the formalized process. For example, we learned that some engineers are part of task force groups that are

[2] *This metaphor is just one of many wonderful examples of a rich language of automotive metaphors used by our interview partners.*

evaluating novel model-driven engineering technology. And, we learned about the existence of a specialized team of code generation experts of which we had not been aware of previously. Which has led to a revised understanding of model-driven engineering not so much being a technical abstraction, where details are hidden away from humans and being taken care of by the machine (as e.g., in a compiler), but an organizational abstraction where details are hidden away from a large workforce of engineers and being take care of by a specialized team of code generation experts [7].

4. RELATED WORK

We are not the first to study MDE abstractions in the field, though to our best knowledge there is little work that shares lessons learnt about the research process used for conducting these studies.

Heijstek and Chaudron [3] studied an industrial MDE case over two years, where a team of 28 built a business application for the financial sector. Using grounded theory they found 14 factors which impact the architectural process. They found that MDE shifts responsibility from engineers to modelers, and that the domain-specific models facilitated easier communication across disciplines and even became a language of business experts. The setup of their case differs from our recent work [7] in that their case had a whole-system view on a closed ecosystem of 28 people, with premium access to both project lead and main architect of the system. In comparison, we had a peephole view on a much larger ecosystem of tens of thousands of people that are collaborating across the main company and it's subsidiaries. It will be interesting to compare the findings of these two studies with regard to these different perspectives.

Hutchinson *et.al.* presented their results of a qualitative user study, consisting of semi-structured interviews with 20 engineers in 20 different organizations [4, 5]. They identified lessons learned, in particular the importance of complex organizational, managerial and social factors, as opposed to simple technical factors, in the relative success, or failure, of MDE. As an example of organizational change management, the successful deployment of model driven engineering appears to require: a progressive and iterative approach; transparent organizational commitment and motivation; integration with existing organizational processes and a clear business focus.

The proceedings of RAO 2006 [6], a workshop on the role of abstractions, providing interesting insight into both the role and study of abstraction, both in the context of MDE and in the context of software engineering in general.

5. CONCLUSIONS

In this paper we presented lessons learned with regard to the to research methodology of a recent industry case study of ours. When we reviewed definitions of *abstraction* we found a common theme of "hiding away details" with the purpose of "reducing details so programmers can focus on a few concepts at a time" (quotes taken from Wikipedia). How can we, are we asking thus, evaluate what is hidden away? We found that the introduction of MDE abstractions is not happening at the mere technical level, but happens within a complex ecosystem of different people working in different roles. An empirical study of MDE should thus mind the ecosystem.

Given that abstraction is about hiding away details, often people working with these abstraction are not aware of what has been hidden themselves. We found that interviewing recent adopters of MDE technology provide a unique window of opportunity onto people who can compare how work has been done before and after an abstraction's introduction We also found it helpful to ask participant to focus on concrete examples, such as "most recent bug report" or "people you talked to last week," in order to learn about their actual work experience.

These lessons learned are obviously based on a single case study only, however we hope that they might serve as the base for a discussion at the EESMOD workshop of how to best study "that what is hidden away," i.e., abstractions.

Acknowledgments

This research has been funded by the Canadian Network on Engineering Complex Software Intensive Systems for Automotive Applications (NECSIS). Special thanks to Joe D'Ambrosia and Shige Wang from GM Research. We thank Deepak Azad, Neil Ernst, Manabu Kamimura for their feedback on an earlier draft of this paper.

6. REFERENCES

[1] Steve Adolph, Wendy Hall, and Philippe Kruchten. Using grounded theory to study the experience of software development. *Empirical Software Engineering*, 16(4):487–513, August 2011.

[2] Juliet M. Corbin and Anselm Strauss. Grounded theory research: Procedures, canons, and evaluative criteria. *Qualitative Sociology*, 13(1):3–21, March 1990.

[3] W. Heijstek and M. R. V. Chaudron. The impact of model driven development on the software architecture process. In *Software Engineering and Advanced Applications (SEAA), 2010 36th EUROMICRO Conference on*, pages 333–341, 2010.

[4] John Hutchinson, Mark Rouncefield, and Jon Whittle. Model-driven engineering practices in industry. In *Proceeding of ICSE'11*, ICSE '11, pages 633–642, New York, NY, USA, 2011.

[5] John Hutchinson, Jon Whittle, Mark Rouncefield, and Steinar Kristoffersen. Empirical assessment of MDE in industry. In *Proceeding of ICSE'11*, ICSE '11, pages 471–480, New York, NY, USA, 2011.

[6] Jeff Kramer and Orit Hazzan. The role of abstraction in software engineering. In *Proceedings ICSE'06*, ICSE '06, pages 1017–1018, New York, NY, USA, 2006.

[7] Adrian Kuhn, Albert Thompson, and Gail Murphy. An exploratory study of forces and frictions affecting large-scale model-driven development. In *MODELS'12*, 2012.

[8] Robert K. Yin. *Case Study Research: Design and Methods*. Sage Publications, 3rd edition, 2003.

Does the Combined use of Class and Sequence Diagrams Improve the Source Code Comprehension? Results from a Controlled Experiment

Giuseppe Scanniello
DMI, University of Basilicata
Potenza, Italy
giuseppe.scanniello@unibas.it

Carmine Gravino and Genoveffa Tortora
University of Salerno
Fisciano, Salerno, Italy
{gravino, tortora}@unisa.it

ABSTRACT

We present the results of a controlled experiment aimed to investigate whether the source code comprehension increases when participants are provided with UML class and sequence diagrams produced in the software design phase. The experiment has been conducted with Master students in Computer Science at the University of Salerno. The data analysis shows that the participants significantly better comprehend source code when it is added with class and sequence diagrams together.

Categories and Subject Descriptors: D.2.0 [Software Engineering]: General
General Terms: Experimentation, Measurement
Keywords: Comprehension, Controlled Experiment, UML

1. INTRODUCTION

Nowadays the documentation of object oriented software systems contains several UML (Unified Modeling Language) diagrams [16]. The wide diffusion of the UML is due to the number of visual notations it offers to write system blueprints, including conceptual (e.g., business processes and system functions) and concrete items (e.g., programming language statements, database schema, and reusable software components) [10].

The assessment of the benefits deriving from the use of the UML in all the phases of the software life cycle is relevant for the software engineering community as shown by the number of empirical studies in terms of controlled experiments and case studies available in the literature [4]. Although a number of studies have been conducted on the UML, only a few of them have been carried out to assess whether the use of UML models produced in the design phase improves source code comprehension [4].

In this paper, we present the results of a controlled experiment conducted with last year Master students in Computer Science at the University of Salerno. We analyze whether the participants achieved an improved comprehension of source code when it is added with UML class and sequence diagrams produced in the design phase. The control group is represented by the participants who performed comprehension tasks with source code alone, while the treatment group is constituted by the participants who performed the comprehension tasks class and sequence diagrams together.

The remainder of the paper is organized as follows. The design of the experiment is presented in Section 2, while we present and discuss the achieved results in Section 3. The threats that may affect the validity of the results are highlighted in Section 4. In Section 5, we discuss related work. Final remarks and future work conclude.

2. CONTROLLED EXPERIMENT

The planning and the design of the experiment is reported in this section. We followed the template for experimentation in software engineering suggested in [21]. For replication purposes, we made available on the web[1] an experimental package and the raw data.

2.1 Definition

Applying the Goal Question Metric (GQM) paradigm [2], the goal of the experiment can be defined as follows: ***Analyze*** *the use of UML class and sequence diagrams produced in the design phase* ***for the purpose of*** *evaluating them* ***with respect to their*** *support in the comprehension of source code* ***from the point of view of the*** *researcher,* ***in the context of*** *students in Computer Science, and* ***from the point of view of the*** *project manager,* ***in the context of*** *junior programmers.*

2.2 Planning

2.2.1 Context

Participants were last year students enrolled to a Master program in Computer Science at the University of Salerno. They can be considered not far from novice software engineers or junior programmers [5].

The participants to the experiment (UniSa from here on) were all graduate students with basic software engineering knowledge. They had knowledge of requirements engineering, high and low level design of object oriented software

[1] www.scienzemfn.unisa.it/scanniello/SC_UML_Exp1/

systems based on the UML, software development, and software maintenance. The students were asked to accomplish the experiment as an optional activity of an Advanced Software Engineering course of the Master program in Computer Science at the University of Salerno.

There were 16 participants, who were not graded on the comprehension results achieved in the experiment. We asked the participants to perform the tasks based on the experimental objects individually and in a professional way to get one point more to the final mark.

We used two systems for managing: *(i)* the sales of a music shop (i.e., *Music Shop*) and *(ii)* the reservation of theater tickets (i.e., *Theater Ticket Reservation*). The systems Music Shop and Theater Ticket Reservation were developed by students of the Master program in Computer Science at the University of Basilicata as the laboratory activity of an Advanced Object Oriented Programming course. These systems never underwent maintenance operations.

The experimental objects used were selected within these systems. The selection was guided keeping in mind a trade-off between the complexity and the relevance of the implemented functionality. The effort to perform comprehension tasks (about 1 hour) on these objects was another criterion used. For Music Shop, we selected a chunk of 463 LOCs (S1 from here on). For the second system, we selected a chunk of 378 LOCs (S2 from here on). As far as S1 is concerned, the class diagram contained: 6 classes, 27 attributes, and 45 methods; while the sequence diagram: 1 actor, 5 objects, and 11 messages. For S2, the class diagram contained: 5 classes, 22 attributes, and 36 methods; while the sequence diagram: 1 actor, 5 objects, and 9 messages. S1 and S2 were small enough to fit the time constraints of the experiment though realistic for small maintenance operations that a novice software engineers or junior programmer perform within a software company [20].

2.2.2 Hypothesis Formulation and Selected Variables

We defined and tested the following null hypothesis:

Hn0 The presence of UML class and sequence diagrams *does not significantly improve* the comprehension of source code.

The alternative hypothesis (i.e., Ha0) can be easily derived because it admits a positive effect of UML class and sequence diagrams on source code comprehension.

To test the hypothesis above, we considered the independent variable (also named main factor) *Method*. It is a nominal variable that can assume two values: *NO_Mo*(source code alone) and *Mo* (source code added with UML diagrams).

To quantitatively evaluate the comprehension achieved by the participants on the source code, we asked them to fill out a comprehension questionnaire for each experimental object. These questionnaires consisted of 15 multiple choice questions, each admitting the same number of possible answers with only one correct. Then, we defined and used the following dependent variable:

Table 1: Experiment design

	Group A	Group B	Group C	Group D
Run 1	S1, Mo	S1, NO_Mo	S2, Mo	S2, NO_Mo
Run 2	S2, NO_Mo	S2, Mo	S1, NO_Mo	S1, Mo

Comprehension: the number of correct answers provided by the participant.

2.2.3 Experiment Design

We adopted the within-participant counterbalanced experimental design. It is shown in Table 1: each participant worked on S1 and S2 using either Mo or NO_Mo. The students were randomly assigned to each group: A, B, C, and D. Each group contained 4 students.

We also analyzed the effect of:

System. Differences in the experimental objects (e.g., complexity and domain) could affect the comprehension of the participants.

Order of Method. The order in which the participants performed the laboratory runs may bias the results.

2.2.4 Execution of the Experiment

Pilot. A pilot experiment was conducted to evaluate possible issues related to the experimental material. The participants to the pilot were 2 Bachelor students in Computer Science at the University of Basilicata. The results indicated that 2 hours were sufficient to accomplish the experiment. The pilot results allow identifying minor issues in the experimental material. These issues were properly fixed before the experiment.

Experiment Execution. The experiments were organized in three phases. In the first phase the participants attended an introductory lesson on how to execute the comprehension tasks. The participants were informed of the pedagogical purpose of the experiment. Details on the experimental hypotheses were not given.

The second and third phases were sequentially performed in the same day. In the second phase, the participants were asked to accomplish the comprehension tasks according to the adopted design in the two subsequent laboratory runs. Each participant was provided with the following material:

- Handouts of the introductory presentation.
- For each object, we provided the participants with the source code. Depending on the task, we also furnished the UML diagrams. A printout of the associated comprehension questionnaire was also given. The material for the second laboratory run was given to the participants only when they accomplished the first run.
- A post-experiment survey questionnaire used to gain enough insight to strengthen and explain the results of each experiment. The post-experiment survey questionnaire was filled out at the end of the second task and collected by the supervisors together with the material of the second task.

- Some sheets of paper and a pencil.

The post-experiment survey questionnaire was defined to get indications about the overall quality of the provided material, the perceived usefulness of the models, the clarity of the experimental objects, and the goals of the experiment. The used post-experiment survey questionnaire is shown in Table 2.

2.3 Analysis Procedure

We used the non-parametric Wilcoxon test to assess the effect of method (i.e., accept the Hn0). This test was also used in all the cases paired analyses were required (e.g., to study the effect of System). In case of unpaired analysis, we chose the Mann-Whitney U exact test. We employed these non parametric tests due to the sample size and (in some cases) the non-normality of the distributions.

Statistical tests check the presence of statistical significant differences, but they do not provide any information about the magnitude of such a difference. We then used the point-biserial correlation r because is the best way to compute the magnitude of the difference when a Wilcoxon test is used [14]. The r value is computed a $\sqrt{Z^2/N}$, where Z is returned by the Wilcoxon test and N is the sample size. In the empirical software engineering fields [18], the magnitude of the effect sizes measured using the point-biserial correlation is classified as follows: small (0 to 0.193), medium (0.193 to 0.456), and large (0.456 to 0.868).

Further, to analyze the probability that a statistical test will reject a null hypothesis when it is actually false, we analyzed the statistical power of the tests performed. A value close to 0.80 is considered as a standard for the adequacy [13].

To test the effect of Order of Method, we used a method similar to the one suggested by Briand *et al.* in [3]. Let be: $Diff(NO\_Mo)$ the differences for the Comprehension values achieved by the participants, who (according to the experimental design) performed the tasks with NO_Mo first and then with Mo; $Diff(Mo)$ the differences for the Comprehension values achieved by the participants, who performed the tasks with Mo first and then with NO_Mo. We applied the non-parametric Mann-Whitney U exact test to verify *Had:* $Diff(NO\_Mo) > Diff(Mo)$.

Table 2: Post-experiment Survey Questionnaire

Id	Question	Answers
Q1	I had enough time to perform the tasks	(1-5)
Q2	The task objectives were perfectly clear to me	
Q3	The tasks I performed were perfectly clear to me	
Q4	Judge the difficulty of the task concerning the system Music Shop (i.e., S1)	(A-E)
Q5	Judge the difficulty of the task concerning the system Theatre Ticket Reservation (i.e., S2)	
Q6	Using the UML class and sequence diagrams the comprehension of a software system is enhanced	(1-5)
1 = strongly agree; 2 = agree; 3 neutral; 4 = disagree; 5 = strongly disagree		
A = very high; B = high; C = medium; D = low; E = very low		

For all the performed statistical tests, we decided to accept a probability of 5% of committing a Type-I-Error [21].

To investigate the effect of System, we applied the Bonferroni correction [8], [9]. In particular, we used two tests to analyze whether the effect of System is statistically significant. Then, the p-values have to be less than $\alpha_{cor} = \frac{0.050}{2} = 0.025$.

To graphically show the answers of the post-experiment survey questionnaire, we adopted boxplots. These are widely employed since they provide a quick visual representation to summarize data [9].

3. ANALYSIS AND RESULTS

Some descriptive statistics (i.e., median, mean, and standard deviation) on the Comprehension dependent variable are shown in Table 3. The statistics are grouped by Method and System. These descriptive statistics show that the participants achieved a better comprehension when they accomplished the task with the UML diagrams. Furthermore, the participants achieved nearly the same comprehension on S1 and S2 both with Mo and NO_Mo.

3.1 Influence of Method

The results of the Wilcoxon test revealed that Hn0 can be rejected (p-value < 0.01).The effect size is large (i.e., 0.62) and the statistical power is 0.91. An average improvement of 14% was achieved when the participants accomplished the comprehension task with the class and sequence diagrams.

The results of a paired analysis is reported in Table 4. The results show that the number of participants that benefited from the UML diagrams in the source code comprehension (Mo > NO_Mo) were 13 out of 16, while 2 were those who obtained a better comprehension using source code alone (Mo < NO_Mo). One participant out of 16 achieved the same comprehension using Mo and NO_Mo (Mo = NO_Mo), respectively. For each participant, we also computed the difference between the comprehension values he/she achieved with Mo and that achieved with NO_Mo (i.e., Mo − NO_Mo). This value is positive in case the participant achieved a better comprehension with Mo, otherwise negative. The difference is zero, when the same comprehension values was achieved by the participant using Mo and NO_Mo. Some descriptive statistics on Mo − NO_Mo are reported in Table 4. In particular, the average value is 1.31, while the median is 1. The minimum and maximum are -3.00 and 4.00, respectively. The standard deviation is 1.66.

3.2 Effect of co-factors

System - The Mann-Whitney test revealed that there was no significant effect of System. When using Mo and NO_Mo, the p-values are 1 and 0.87, respectively.

Order of Method - the Mann-Whitney test revealed that the order in which the participants performed the comprehension tasks was not statistically significant. These results indicated that the participants did not get a significantly

Table 3: Descriptive Statistics

System	Observation	Mo					NO_Mo				
		Min	Max	Med	Mean	Std. Dev.	Min	Max	Med	Mean	Std. Dev.
All	32	10	15	14	13.06	1.53	9	14	12	11.75	1.29
S1	16	11	14	14	13.12	1.25	9	13	11.5	11.62	1.41
S2	16	10	15	13.5	13	1.85	10	14	12	11.88	1.25

Table 4: Analysis of the differences

Mo > NO_Mo	Mo < NO_Mo	Mo = NO_Mo	Descriptive statistics of Mo − NO_Mo				
			Min	Max	Med	Mean	Std. Dev.
13/16	2/16	1/16	-3.00	3.00	1.00	1.31	1.66

better comprehension of the source code when passing from the first task to the second one.

3.3 Further analysis

We also collected information on the time the participants spent to accomplish a comprehension task on source code. We then performed a further analysis on that information to complete the assessment on the affect of the UML class and sequence diagrams on the source code comprehension.

The Wilcoxon test revealed that there was no statistically significant difference in the completion time (i.e., the dependent variable) when using Mo and NO_Mo (p-value = 0.12). The hypothesis tested with this non parametric test was two-sided because we could not make any postulation on the effect of Method on completion time. Therefore, the combined use of UML class and sequence diagrams does not significantly affect the time the participants spent to comprehend source code. However, the descriptive statistics indicate that the participants on average spent more time when using Mo (mean = 30.06) with respect to NO_Mo (mean = 23). The median values are 27.5 for Mo and 20 for NO_Mo. Furthermore, the standard deviation for Mo is almost twice that of NO_Mo (i.e., 13.13 and 7.61, respectively).

The Mann-Whitney test revealed that there was not a significant effect of System on the observed results. The p-values are 0.53 and 0.79 when using Mo and NO_Mo, respectively. Furthermore, the effect of Order of Method was not statistically significant as well. The returned p-value was 0.37.

It is worth mentioning that the investigation on the completion time was not our primary goal here. This is because we confined the discussion on that dependent variable in this subsection.

3.4 The Results of the Post-experiment Survey Questionnaire

The boxplots reported in Figure 1 summarize the answers to the post-experiment survey questionnaire of UniSa. As the box of Q1 shows, the participants to experiment considered the time to conduct the experiment appropriate (the median is 1 and corresponds to strongly agree). They also clearly understood both the objectives and the tasks. The boxes of Q2 and Q3 show that the medians are 1 (strongly agree) and 2 (agree), respectively. A neutral judgment on the complexity of the experimental objects was given as the boxes of Q4 and Q5 show: both the medians are 3. As the box of Q6 shows, all the participants found the use of the UML effective for the comprehension of source code (the median is 2 and corresponds to agree).

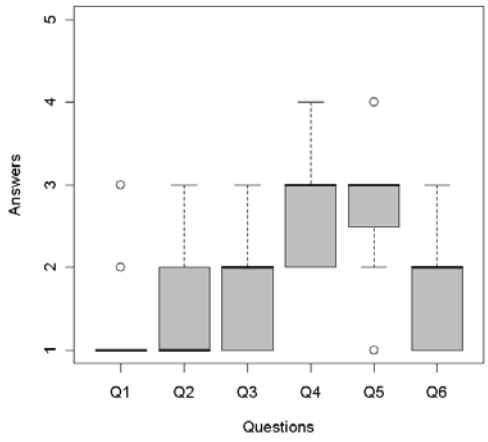


Figure 1: Boxplots of the answers

4. THREATS TO VALIDITY

In the following subsections the threats that could affect the validity of the controlled experiments are presented. We use the schema proposed in [21].

4.1 Internal Validity

Internal validity concerns the degree to which conclusions can be drawn about the causal effect of the independent variables on the dependent variables.

Interaction with selection. This threat has been mitigated because each group of participants worked on different objects with Mo or NO_Mo on two comprehension tasks. Further, the participants had similar experience with UML, software system modeling, and computer programming.

Maturation. Participants might have learned how to improve their comprehension performances passing from the first task to the second one. The data analysis revealed that the order in which the participants performed the tasks did not significantly affect source code comprehension.

Diffusion or imitation of treatments. This threat concerns the information exchanged among the participants. The participants were monitored by the experiment supervisors, who did not allowed the participants to communicate each other.

4.2 External Validity

The main issue of the external validity refers to the possibility of generalizing the results.

Interaction of selection and treatment. The use of students may affect the external validity [5], [6], [17]. Threats are related to the representativeness of the participants as compared with professionals. Replications with this kind of participants are needed to confirm or contradict the achieved results.

Interaction of setting and treatment. It concerns the experimental objects used. The software systems on which the experimental objects were selected have never undergone maintenance operations. This may affect the source code comprehension. Also, the size and complexity of the objects may affect the validity of the results. However, larger and more complex tasks could excessively overload the participants, thus biasing the results.
Using source code printout could have negatively affected the performance of the participants. It is worth noting that the use of a IDE should equally affect the comprehension of the participants both using and not using the diagrams.

4.3 Construct Validity

Construct validity concerns generalizing the results to the concepts behind the experiment. Some threats are related to the design of the experiments and to social factors.

Interaction of different treatments. The adopted design partially mitigated these threats.

Mono-method bias. Using one single dependent variable could bias the results. The adopted measure to quantitatively assess the comprehension is well known and widely used (e.g., [11], [15]).

Confounding constructs and level of construct. More levels than High and Low could be used in the classification of participants' ability. We are also aware that the use of a different measure to assess participants' ability could lead to different results. Also, the way to group the students into high and low experienced participants could represent another threat.

Evaluation apprehension. We mitigated this threat because the participants were not evaluated on the comprehension they achieved on the source code used in the experiment. The participants were not aware of the experimental hypotheses investigated.

Experimenters' expectations. We mitigated this threat formulating the questions of the comprehension questionnaires so conditioning their answers in favor of neither Mo nor NO_Mo. The post-experiment survey questionnaire was designed to capture the participants' perception on the experiment and the used material. The survey was intended to support and explain the quantitative results and was designed using standard approaches and scales [19].

4.4 Conclusion Validity

Conclusion validity concerns issues that may affect the ability of drawing a correct conclusion.

Reliability of measures. The used measure allowed us to assess in an objective way the comprehension level achieved by the participants on source code.

Random heterogeneity of participants. We drew fair samples and conducted our experiment with participants belonging to these samples. Another threat is related to the number of observations. Replications on a larger number of participants are required to confirm or contradict the results.

Fishing and the error rate. For UniSa the null hypothesis has been rejected with p-value less than 0.01 and 0.91 as the statistical power value.

5. RELATED WORK

We discuss the related literature concerning controlled experiments aimed at assessing the effect of using the UML in software maintenance and program comprehension. To get a deeper understanding on empirical evaluations on the models and forms used in the UML, a systematic literature review is available in [4].

Arisholm *et al.* [1] observe that the availability of documentation based on the UML may significantly improve the functional correctness of changes as well as the design quality when complex tasks have to be accomplished. Although this study and the one presented here have the same research goal (i.e., impact of UML documentation on software maintenance), a number of differences are present. The main difference is that we specifically focus on comprehension tasks, while the authors on modification tasks performed both on UML diagrams and source code.

Dzidek *et al.* [12] investigate on the costs and benefits in using the UML to maintain and evolve software systems. The authors conduct a controlled experiment with professional programmers. The results reveal that the use of the UML significantly impacts the functional correctness of the maintenance operations. Conversely, the UML use does not significantly affect the time to perform maintenance operations. The main difference with respect to our investigation is that the focus of the controlled experiment is not based on the comprehension of source code and the UML diagrams are different.

Gravino *et al.* [15] present a controlled experiment to assess whether UML models produced in the early phases of the development process (i.e., requirements elicitation and analysis phases) improve the comprehension of source code. The results reveal that the use of models does not significantly improve the comprehension of source code with respect to the use of source code alone. In some way, the study presented here and that highlighted in [15] go in the same direction: investigating whether or not the comprehensibility of the source code might improve when it is added with UML diagrams. The most remarkable difference between these papers is that here we consider UML diagrams produced in the design phase.

6. CONCLUSION AND FUTURE WORK

In this paper, we have presented the results of a controlled experiment to assess whether the comprehension of source code is affected when it is added with UML class and sequence diagrams produced in the design phase. The data analysis revealed that the participants benefited from the use of the UML diagrams. We used a controlled experiment because a number of confounding and uncontrollable factors could be present in real project settings. Controlled experiments are often conducted in the early steps of empirical investigations that take place over the years (e.g., [1], [7]).

Due to the results of the experiment, we plan to conduct industrial case studies in the future. This part of our research is still in progress and is the most challenging and complex. The results presented in the paper suggest the following future research directions:

1. It is possible that participants, who were provided with the UML diagrams, did not deeply analyzed the code because they believed that these diagrams provided adequate information to accomplish a comprehension task on source code. This point is interesting from the researcher perspective and then worthwhile being investigated.

2. It could be interesting to investigate the motivation guiding the software engineers in trusting software models and how they select them to accomplish their tasks. It will be then worth analyzing how software engineers choose the information to perform comprehension tasks.

Acknowledgments

We wish to thank the Ilaria Bilancia and the Michela Continanza for their precious help in preparing the experimental material and in executing the experiments. We also thank the participants to the experiment.

7. REFERENCES

[1] E. Arisholm, L. C. Briand, S. E. Hove, and Y. Labiche. The impact of UML documentation on software maintenance: An experimental evaluation. *IEEE Transactions on Software Engineering*, 32:365–381, 2006.

[2] V. Basili, G. Caldiera, and D. H. Rombach. *The Goal Question Metric Paradigm, Encyclopedia of Software Engineering*. John Wiley and Sons, 1994.

[3] L. C. Briand, Y. Labiche, M. Di Penta, and H. Yan-Bondoc. An experimental investigation of formality in UML-based development. *IEEE Transactions on Software Engineering*, 31(10):833–849, 2005.

[4] 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, 2011.

[5] J. Carver, L. Jaccheri, S. Morasca, and F. Shull. Issues in using students in empirical studies in software engineering education. In *Proceedings of the 9th International Symposium on Software Metrics*, pages 239–, Washington, DC, USA, 2003. IEEE Computer Society.

[6] M. Ciolkowski, D. Muthig, and J. Rech. Using academic courses for empirical validation of software development processes. *EUROMICRO Conference*, 0:354–361, 2004.

[7] M. Colosimo, A. De Lucia, G. Scanniello, and G. Tortora. Evaluating legacy system migration technologies through empirical studies. *International Journal on Information and Software Technology*, 51(12):433–447, 2009.

[8] W. J. Conover. *Practical Nonparametric Statistics*. Wiley, 3rd edition edition, 1998.

[9] J. L. Devore and N. Farnum. *Applied Statistics for Engineers and Scientists*. Duxbury, 1999.

[10] P. J. Dobing B. How uml is used. *Communications of the ACM*, 49(5):109–113, 2006.

[11] J. Dolado, M. Harman, M. Otero, and L. Hu. An empirical investigation of the influence of a type of side effects on program comprehension. *IEEE Transactions on Software Engineering*, 29:665–670, 2003.

[12] W. J. Dzidek, E. Arisholm, and L. C. Briand. A realistic empirical evaluation of the costs and benefits of UML in software maintenance. *IEEE Transactions on Software Engineering*, 34:407–432, May 2008.

[13] P. Ellis. *The Essential Guide to Effect Sizes: Statistical Power, Meta-Analysis, and the Interpretation of Research Results*. Cambridge University Press, 2010.

[14] A. Field and G. Hole. *How to Design and Report Experiments*. Sage publications Limited, 2003.

[15] C. Gravino, G. Tortora, and G. Scanniello. An empirical investigation on the relation between analysis models and source code comprehension. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, pages 2365–2366, New York, NY, USA, 2010. ACM.

[16] O. M. Group. OMG Unified Modeling Language (OMG UML), Infrastructure, V2.1.2. Technical report, Nov. 2007.

[17] J. Hannay and M. Jørgensen. The role of deliberate artificial design elements in software engineering experiments. *IEEE Transactions on Software Engineering*, 34:242–259, March 2008.

[18] V. B. Kampenes, T. Dybå, J. E. Hannay, and D. I. K. Sjøberg. A systematic review of effect size in software engineering experiments. *Information & Software Technology*, 49(11-12):1073–1086, 2007.

[19] A. N. Oppenheim. *Questionnaire Design, Interviewing and Attitude Measurement*. Pinter, London, 1992.

[20] G. Scanniello, C. Gravino, and G. Tortora. Investigating the role of UML in the software modeling and maintenance - a preliminary industrial survey. In *Proceedings of the 12th International Conference on Enterprise Information Systems*, pages 141–148, 2010.

[21] C. Wohlin, P. Runeson, M. Höst, M. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in Software Engineering - An Introduction*. Kluwer, 2000.

UML Class Diagram Simplification: What is in the developer's mind?

Hafeez Osman
hosman@liacs.nl

Dave R. Stikkolorum
drstikko@liacs.nl

Arjan van Zadelhoff
avzadelh@liacs.nl

Michel R.V. Chaudron
chaudron@liacs.nl

Leiden Institute of Advanced Computer Science, Leiden University
Niels Bohrweg 1, Leiden, the Netherlands

ABSTRACT

Class diagrams play an important role in software development. However, in some cases, these diagrams contain a lot of information. This makes it hard for software maintainers to use them to understand a system. In this paper, we aim to discover how to simplify class diagrams in a such way that they make systems easier to understand. To this end, we performed a survey to analyze what type of information software developers find important to include or exclude in order to simplify a class diagram. This survey involved 32 software developers with 75% of the participants having more than 5 years of experience with class diagrams. As the result, we found that the important elements in a class diagram are class relationship, meaningful class names and class properties. We also found that information that should be excluded in a simplified class diagram is GUI related information, private and protected operations, helper classes and library classes. In this survey we also tried to discover what types of features are needed for class diagram simplification tools.

Keywords

Reverse engineering, UML, Class Diagram, Simplification

1. INTRODUCTION

The UML class diagram is one of the valuable artefacts in software development and software maintenance that describes the static structure of programs at a higher level of abstraction than source code [9]. The UML models, which are usually created during the design phase, are often poorly kept up to date during the realization and maintenance phase. As the implementation evolves, correspondence between design and implementation degrades from its initial design [10]. Reverse engineering is one of the techniques used to uncover a software design after the implementation phase. Reverse engineering is the process of analyzing the source code of a system to identify its components and their interrelationships and create design representations of the system at a higher level of abstraction [4]. However, (the) class diagrams resulting from reverse engineering sometimes suffer from too much details. In particular, reverse engineered class diagrams are typically a detailed representation of the underlying source code, which makes it hard for the software engineer to understand what the key elements in the software structure are [11]. Although several Computer Aided Software Engineering (CASE) tools have options to leave out several properties in a class diagram, they are unable to automatically identify classes and information that are not useful or less important. As part of a recent study, Fernández-Sáez et al. [8] found that developers experience more difficulties in finding information in reverse engineered diagrams than in forward designed diagrams and also find the level of detail in forward designed diagrams more appropriate than in reverse engineered diagrams. A good representation of a class diagram by showing the crucial information of a system is needed, especially when new programmers want to join a development group; they need a starting point in order to understand the whole project. Tools support during maintenance, re-engineering or re-architecting activities have become important to decrease the time software personnel spend on manual source code analysis and help to focus attention on important program understanding issues [2].

This paper aims to find out how to simplify a UML class diagram by leaving out unnecessary information without affecting the developer's understanding of the entire system. To this end, we have conducted a survey to gather information from software developers about what type of information they focus on. We have prepared a questionnaire that consists of 15 questions which are divided into 3 parts in order to discover this information. In total, 32 software developers from the Netherlands that participated in this survey. The paper is structured as follows. Section 2 discusses related works and followed by Section 3 that describes the survey methodology. Section 4 describes the result and findings. We discuss our findings in Section 5 and Section 6 suggests future work. This is followed by our conclusion in Section 7.

2. RELATED WORKS

In this section, we discuss several studies that are slightly related. Yusuf et al. [15] did a study about assessing the comprehension of UML Class diagrams via eye tracking. Their goal was to obtain an understanding of how human subjects use different types of information in UML class diagrams in performing their tasks. They found that experts tend to use such things as stereotype information, coloring, and layout to facilitate more efficient exploration and navigation of class diagrams. Also, experts tend to navigate from the center of the diagram to the edges whereas novices tend to navigate from top-to-bottom and left-to-right.

Egyed [6] described an approach for automated abstraction that allows designers to 'zoom out' on class diagrams to investigate and reason about their bigger picture. This approach was based on a large number of abstraction rules and, when used together, it can abstract complex class structures quickly. A part of the abstraction rules are semantic rules that look at the semantic properties of classes and relationships which makes it possible to eliminate a helper class and derive a (slightly) more abstract class diagram. In total, the article provides 121 rules to abstract a class diagram. They validated their abstraction technique and its rules on numerous third-party applications and models with up to several hundred model elements.

Bassil et al. [1] conducted a survey study about software visualization (SV) tools. This study addresses various functional, practical, cognitive and code analysis aspects that users may be looking for in SV tools. They found that functional aspects such as searching and browsing, use of colors, and easy access from the symbol list to the corresponding source code were rated as the most essential aspects. Also hierarchical representations and navigation across hierarchies were strongly desired.

3. SURVEY METHODOLOGY

In this section, we describe i) The questionnaire design; and ii) The experiment description.

3.1 Questionnaire Design

The questionnaire was organized into three parts i.e. Part A, B and C. In total, there were 15 questions. For this survey, we organized the questionnaire by dividing this questionnaire into two different sets of questions. Both sets of questions had the same questions for Part A and C. However, we differentiated the questions in Part B. The questionnaire can be found at [12].

3.1.1 Part A: Personal Questions

Part A consisted of six questions. Questions one to four were intended to access respondents background information. In questions five and six, we wanted to discover the respondents' preferences of software artefacts (i.e. UML, Source code) for understanding a system.

3.1.2 Part B: Practical Problems

This part contained three questions. Each of these questions included of a class diagram. In this part the respondents' were required to mark information that can be left out of the provided class diagram without affecting their understanding of the system. The class diagrams that were used in this survey are the following:

1. **Automated Teller Machine (ATM) simulation system [3] :** This fully functional system has a class design and complete implementation source code. The complete software documents based on UML that were provided consists of 22 design classes. We reverse engineered the design of this system for this study.

2. **Pacman Game [5] :** Pacman's Perilous Predicament is a turn based implementation on the classic Pacman arcade game. The amount of classes in the source code in this system is 17 while only 15 classes are stated in the class diagram design. Both forward and reverse engineered designs were used in this survey.

3. **Library System [7]:** Library System is a system that enables a user to borrow a book from a library. This complete system consists of 24 classes in the source code. The reverse engineered design was used for this survey.

Enterprise Architect version 7.5 were used as a tool to reverse engineered the source codes to produce class diagrams. The questionnaire included diagrams with different Level of Detail (LoD). In set A, ATM system in MLoD and Library System in HLoD were used and in set B, ATM system in HLoD and Library System in MLoD were used. Different LoD were used to simulate different types of detail that are used in a class diagrams. We also used different sources of class diagram by setting Forward Design Class Diagram and Reverse Engineered Class Diagram to simulate the different flavours of class diagrams that exist in the software industry. In HLoD class diagrams, all class diagram elements i.e.classes, attributes, types of attributes, operations, operations' return types, parameters in operations and relationship are presented. In MLoD, all elements in the class diagram are shown except Types of attributes and Parameters in operations.

3.1.3 Part C: (Class Diagram) Indicators for Class Inclusion

This part consisted of six open-ended questions. The aim of these questions was to discover which information is needed in a class diagram and which type of information may be left out.

3.2 Experiment Description

The experiment was conducted on 6th of June 2012 at Leiden Institute of Advanced Computer Science (LIACS), Leiden. The participants of this survey were software developers from all over the Netherlands. In total, there were 32 respondents and all of them were members of a software developer community called Devnology. The participants had to answer every question and were free to ask any question during the questionnaire session. The time given to answer those questions was 60 minutes and all participants answered the questions in one session.

4. RESULTS AND FINDINGS

In this section, we present our results and analysis of the answers given by the respondents. The complete responses of this questionnaire can be found at [13] and the full analysis can be found at [14]. This section is divided into three parts which presents our analysis for this questionnaire.

4.1 Part A: Personal Questions

This part presents the results and analysis of the respondents' background and their preference of software artefacts (UML or Source Code) for understanding a system.

4.1.1 Background of the Respondents

The questions to access the respondents' background are the following:

- Question A1: What is your role at the moment?
- Question A2: How many year(s) of experience do you have in working with class diagrams?
- Question A3: Where did you learn about UML?
- Question A4: How do you rate your own skills in creating, modifying and understanding a class diagram?

In terms of the respondent's role, 81% of the respondents are programmers. 50% of the respondents are software architects and 28% of the respondents are software designers. This shows that the majority of the respondents are involved in the design and implementation phases in software development. 14 out of 26 programmers (54%) are also software architects or software designers. For the respondents' experience in class diagrams, we found that 50% of the respondents are experienced with class diagrams for more than 10 years. The results also show that 75% of the respondents have experience with class diagrams for more than 5 years. In term of UML knowledge, 47% of the respondents had learned about UML in Polytechnic or University and 25% have taken professional training to learn UML. Meanwhile, 38% of the respondents learned UML by themselves and 19% learned from their colleague(s) or from industrial practice. There were no participants that answered 'No'. This shows that all participants of this survey have knowledge of UML. For the respondent's skills in class diagrams, most of the respondents (88%) have average and good skills on creating, modifying, and understanding class diagrams and only 3% have excellent skills related to class diagrams. This indicates that over 90% of the respondents have average skills or above related to class diagrams.

4.1.2 Repondents' Preference of Software Artefacts

The questions to discover the respondent's preference of software artefacts are the following:

- Question A5: Indicate whether you (dis)like to look at source code for understanding a system?
- Question A6: Indicate whether you (dis)like to look at UML models for understanding a system?

The results shown in Figure 1 indicate that there are not much differences between Like or Dislike of Source code versus UML design. We further investigated this result by separating this and present it according to the role of the respondents, specifically programmer, software architect, and software programmer. The results show that the programmers are a bit more positive about source code than UML but the difference is not significant. It was quite a surprise to see that a lot of software designers like using source code more than UML to understand a system. The same for the software architects, they like using source code more than UML to understand a system.

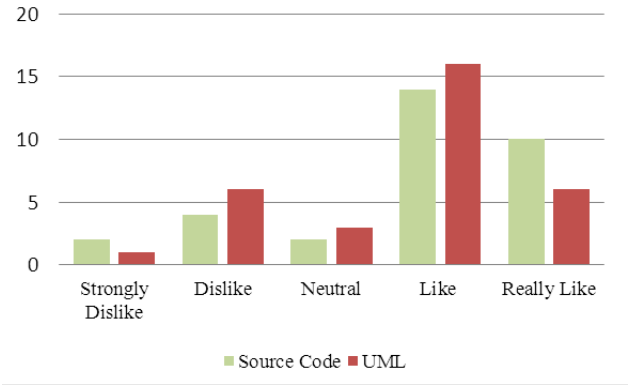


Figure 1: Respondents Like or Dislike Source Code vs UML

4.2 Part B: Practical Problems

The results of this part were analysed by combining the answers based on the following categories: attribute, operation, class, relationship, inheritance and package.

4.2.1 Category 1: Attribute

In the attribute category we divided this category into two subcategories: Properties and Type of Attribute. We divided the properties subcategory in three elements: Protected, Public and Private. We also divided the type of attribute subcategory into three elements: No primitive type, GUI related, and Constant. The results show that 25% of the respondents chose not to include the GUI related attributes. 19% of the respondents like to leave out Private and Constant types of attributes. 13% of respondents proposed to leave out protected attributes.

4.2.2 Category 2 : Operation

The operation category is divided into two subcategories namely Properties and Type. The Properties subcategory consists of four elements and these are Private, Protected, Public and Return Type. The Type subcategory also consists of four elements and these are Event Handler, General Function, Getters/Setters and Constructor. For the constructor element, we divided this element into 2 groups and these are With Parameter and Without Parameter because the respondents seem to differentiate this information. The results show that 25% of the respondents chose to exclude constructors without parameters. Nevertheless, 16% of the respondents suggested that all constructors should be left out in a class diagram. For getters and setters, 19% of the respondents suggested that these operations should be excluded. 9% of the respondents mentioned that General

No	Group	Subgroup	No	Group	Subgroup
1	**Relationship/ Connectivity/ Interaction**	*Association*	3	**Class structure/ properties**	*Abstraction*
		Inheritance			*Method/Operation*
		Direction			*Attribute*
		Dependency			*Public Interface*
		Multiplicity			*Class Entities*
2	**Class Role and Responsibility**	*Classname (meaningful)*			*Size Large/Small*
		Class Behaviour			*Public Properties*
		Business Entities			*Class Hierarchy*
		Main Classes/ Object/Purpose			*Object related*
		Class functionality and responsibility	4	**High level**	*Concept*
		Domain			*Design Pattern*
		properties name and methods name			*Overview*
		Reasoning	5	**Others**	*Data*
		"starting" point			*All Generic Classes*

Table 1: Keywords on Types of Information to Understand a System

Functions should not be included in a class diagram because these functions are commonly used and well-known to programmers. Event Handlers were chosen to be excluded from a class diagram by 6% of the respondents.

4.2.3 Category 3: Class

The class category is divided into two subcategories which are Type of Class and Role. The type of class subcategory consists of Interface, Enumeration, and Abstract elements while the role subcategory consists of five elements which are Console, Listener, Input/Support Classes, Log, and GUI Related. In Type of Class, 38% of the respondents chose not to include Enumeration classes. This is followed by interface classes with 19% and 13% suggested that Abstract classes should not be included in simplified class diagrams. The Role subcategory results show that half of the respondents suggested that GUI related classes and classes for logging tasks should be left out in order to simplify a class diagram. The respondents suggested eliminating these classes because without these classes you can still understand the system. The input function is a class that is used to take the input from the interface or device. 21% of the respondents indicated that this type of class should not to be included.

4.2.4 Category 4: Package

In the package category, there is only one subcategory which is Separation of Class Diagram. In the Library System class diagram, there were 4 respondents that drew several lines to separate the GUI related classes. They suggested that the class diagram should be separated into two different diagrams. This means that they wanted to keep the GUI related classes and classes created for the system separated.

4.3 Part C: Class Diagram Indicators for Class Inclusion/Exclusion

The analysis for this part was done by observing the answers from the respondents and creating several keywords to categorize these answers.

4.3.1 Type of Information in Class Diagrams for Understanding a Software System

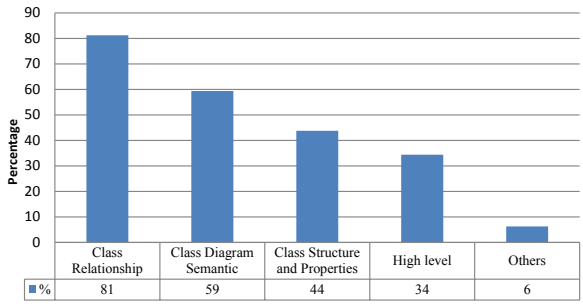


Figure 2: Types of Information the Respondents Look For in a Class Diagram

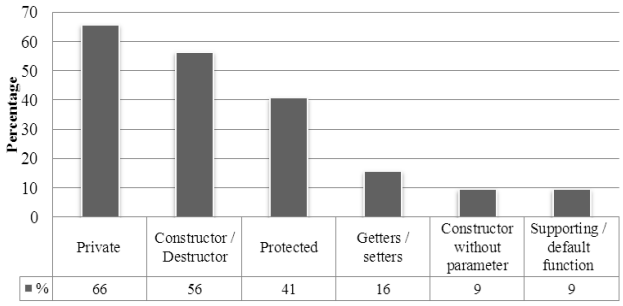


Figure 3: Information of Operations that Should be left out

The respondents were asked the following question: 'In software documentation, particularly in class diagrams, what type of information do you look for to understand a software system?'. In this question, we were expecting to get the type of information that the respondents look for to understand a system. Based on the answers, we created several keywords and categories as shown in Table 1. In Figure 2, the results show that class relationship is the most important information in a class diagram that the respondents searched for understanding a class diagram. 81% of the respondents mentioned this. 59% of the respondents searched for class Role and Responsibility(RnR) such as meaningful class names, class functionality and behaviour, class properties and so on. About 44% of the respondents were looking at class properties such as attributes, operations, class interfaces and so on. This follows with 34% of the respondents that were looking at the high level abstraction of the class diagram for example design concepts, design patterns and class overviews.

4.3.2 Type of Information that Can be Left out

For type of information that can be left out, the respondents need to answer the following question: 'In a class diagram, what type of information do you think can be left out without affecting your understanding of a system?'. This question is divided into four sections which are: Classes, Operations, Relationships, and Others. In the section of classes, almost half of the respondents (44%) suggested that helper classes should not be included in a class diagram. A quarter of the respondents (25%) did not want library classes to appear in a class diagram. These library classes could make

a class diagram more complex. 22% of the respondents suggested that the interface class type should not be included in a class diagram.

In the section of Operations, the results(Figure 3) show that 66% of the respondents chose to exclude private operations in a class diagram. 56% of the respondents mentioned that constructors and destructors are not needed in a class diagram in order to understand a system while only 9% of the respondents mentioned that they do not need constructors without parameters. 41% of the respondents mentioned that protected operations should be left out from a class diagram. In section Relationship, multiplicity is what most respondents mentioned that is not needed in a class diagram. However, only 6% of the respondents mentioned this, which is a quite low percentage. 3% of the respondents do not need any Labels (or roles of the relationships), Self Relations and References in a class diagram. In section Other(s), 9% of the respondents said that private fields should not be included in a class diagram. Only 3% of the respondents suggested technical, duplicates and UI information not to be included in a class diagram.

4.3.3 The Criteria to Indicate Important Classes in a Class Diagram

The respondents were asked about 'What criteria do you think indicate that a class (in a class diagram) is important for understanding a system?'. This question tried to discover the criteria to indicate important classes in a class diagram. As the results, 38% of respondents think that the relationship is the most important criterion in a class diagram which also align with our results in question C1. 16% of the respondents think that meaningful class names, Business or domain value, and the position of class are the important criteria in a class diagram. This result shows that RnR and the meaning of a class play a role in understanding a class diagram. Some respondents preferred to search for the position of the class and most of them mentioned that the middle of a class diagram should contain the important classes.

4.3.4 Type of Relationships that the Respondents Look at First

The respondents were asked the following question: 'If you try to understand a class diagram, which relationships do you look at first?'. In this question, we aimed to find out what type of relationship the respondents look at first and three types of relationships were provided as example answers. We found that it is quite biased because most of the answers only mentioned about these types of relationships. None of the respondents answered other types of relationships such as composition, aggregation, and realization. The results shows that 41% of the respondents like to search for association relationships first while 19% search for Dependency relationships. Only 9% searched for inheritance relationships.

4.3.5 Features/functions Expected in a Class Diagram Simplification Tool

The respondents were asked the following question: 'If there is a tool for simplifying class diagrams (e.g. obtained from reverse engineering), what features/functions would you expect from such a tool?'. In this question, we tried to discover what kind of features the respondents are looking for if there is a tool which could simplify a class diagram. The results show that the respondents mainly want a tool that can hide/unhide information. The other feature that relates to this is the drill up/down feature because when you are drilling up, the amount of information of a class diagram will be less and vice versa. 16% of the respondents wanted to see more information about a class by hovering over a class in a class diagram for example. Another feature that many respondents wanted (13% of the respondents) is the changeable layout of the class diagram in which the navigation can be improved.

5. DISCUSSION

In this section we discuss the results and findings presented in the previous section.

5.1 Respondents Background

In Part A, we have accessed the information about the respondents' skills and experiences in UML particularly in class diagrams. All of the respondents have knowledge in UML with 75% of the respondents have more than 5 years of experience in class diagrams where 50% of the respondents have experience for more than 10 years. In terms of respondent skills in class diagrams, we found that over 90% of the respondents have at least average skills in creating, modifying and understanding class diagrams. For the respondents preference of software artefacts, the results show that there is no significant difference between the usage of source code and class diagrams in order to understand a system. However, we found that most of the software architects and software designers prefer source code over class diagrams to understand a system. A reason for this result could be that they have a good knowledge of programming or they have other techniques other than UML.

5.2 Class Properties

Relationships in a class diagram are considered the important elements to show the structure of classes in a class diagram. Most of the respondents in this survey looked at the association relationship first. This shows that the association relationship is important in class diagrams. However, we found this result not really accurate since the respondents only gave the answer within the examples given in the question. In this survey, we found that most of the respondents suggested leaving out or separating the GUI related information from the class diagrams. The respondents focus more on class diagram information that is created by the programmer or software designer. The GUI related information exists in source code when a developer used GUI libraries provided by Rapid Application Development (RAD) tools such as Visual Basic. In terms of class operations, most of the respondents suggested to leave out the private and protected type of operations. These types of operations are only used for internal classes and member classes for protected operations. We also discovered that constructor/destructor operations should not appear in simplified class diagrams. Particularly in Part B, we found that most of the respondents suggested that constructors without parameters should be excluded.

5.3 Class Role and Responsibility(RnR)

One of our useful discoveries in this study is the importance of the class RnR in a class diagram. Class RnR based on classes', operations' and attributes' name; are important because from our observation the respondents seemed to try to understand a system based on class RnR. By using this information, they can get an overall idea on how a system works and get some hints of the functionalities of classes in a class diagram. In this survey we also discovered that classes that should be left out in a class diagram are helper classes, library classes and interfaces classes. Most of the respondents suggested leaving out helper classes. Nevertheless, it is not easy to automatically identify helper classes based on the class name or other information because it only can be identified manually and the results are different based on the software developer's experience.

5.4 Class Diagram Simplification Tool Features

From this survey, we found out that most of the respondents need tools for simplifying a class diagram that can Hide/Unhide information and Drill up and down a class diagram. With this feature they can use the tools to understand the system in general by leaving out the details and they can get more information when they want to modify the system.

6. FUTURE WORK

This study was an early experiment on how to simplify a class diagram and we see a number of ways to extend this work. In this study, we have discovered information that should be left out to simplify a class diagram. By using this information, a simplified class diagram could be produced. We propose to validate the resulting class diagram by using an industrial case study and discover the suitability of the simplified class diagram for the practical usage. Also from this result, we found that class role and responsibility are one of the important indicators in a class diagram. The role and responsibility of a class are detected by using the class names, operations names and attributes names. We would like to suggest a study on names (class, operation and attribute) that the software developers find important or meaningful in order to understand a system.

7. CONCLUSION

This study presented a survey on how to simplify a class diagram without affecting their understanding of a system. In particular, the questions in this survey were about the information that should be left out from a class diagram and also what kind of important information should remain. 32 software developers from the Netherlands participated in this survey. The most important element in a class diagram is the class relationship. In this survey we discovered that most of the respondents search for class roles and responsibility in order to get high-level understanding how a system works. This means, meaningful class names, operation names and attribute names are important to show the functionality or responsibility of a system. To simplify a class diagram, most of the respondents chose to exclude GUI related information and also library classes. This shows that most of the software developers only need the information about the classes that are created or designed. Most of the respondents also mention that helper classes should be excluded to simplify a diagram. However, it is not easy to automatically identify a helper class. Private, Protected and Constructor (without parameter) are types of operations that may be left out in order to simplify a class diagram. Although we are aware research on validation of our approach needs to be done, we found several useful indicators that could be used in the future for class diagram simplification.

8. ACKNOWLEDGEMENTS

We would like to thank all the participants from the Devnology community.

9. REFERENCES

[1] S. Bassil and R. K. Keller. *Software visualization tools: Survey and analysis*, volume 67, pages 7–17. IEEE, 2001.

[2] B. Bellay and H. Gall. *A Comparison of Four Reverse Engineering Tools*, pages 2–11. IEEE Computer Society Press, 1997.

[3] R. C. Bjork. Atm system. `http://www.math-cs.gordon.edu/courses/cs211/ATMExample/`.

[4] E. J. Chikofsky and J. H. Cross. *Reverse Engineering and Design Recovery: A Taxonomy*, volume 7, pages 13–17. IEEE Computer Society Press, 1990.

[5] A. Craig, A. Dinardo, and R. Gillespie. Pacman game. `http://code.google.com/p/tb-pacman/`.

[6] A. Egyed. *Automated abstraction of class diagrams*, volume 11, pages 449–491. ACM Transaction Software Engineering Methodology, 2002.

[7] H. Eriksson, M. Penker, B. Lyons, and D. Fado. *UML 2 Toolkit*. Wiley Publishing Inc, 2004.

[8] A. M. Fernández-Sáez, M. Genero, M. R. V. Chaudron, and I. Ramos. *A Controlled Experiment on the Impact of UML Diagram Origin on Maintenance Performance*. Unpublished.

[9] Y.-G. Guéhéneuc. *A Systematic Study of UML Class Diagram Constituents for their Abstract and Precise Recovery*, pages 265–274. IEEE, 2004.

[10] A. Nugroho and M. R. V. Chaudron. *A Survey of the Practice of Design - Code Correspondence amongst Professional Software Engineers*, pages 467–469. Proceedings of the First International Symposium on Empirical Software Engineering and Measurement, September 20-21, 2007.

[11] H. Osman and M. R. V. Chaudron. *An Assessment of Reverse Engineering Capabilities of UML CASE Tools*, pages 7–12. 2nd Annual International Conference Proceedings on Software Engineering Application, September 12-13, 2011.

[12] H. Osman and A. van Zadelhoff. Original questionnaire. `http://www.liacs.nl/~hosman/Questionnaire.rar`.

[13] H. Osman and A. van Zadelhoff. Survey data. `http://www.liacs.nl/~hosman/SurveyData.rar`.

[14] H. Osman, A. van Zadelhoff, D. Stikkolorum, and M. Chaudron. Technical report. `http://www.liacs.nl/~hosman/Technical_Report_2012.pdf`.

[15] S. Yusuf, H. Kagdi, and J. I. Maletic. *Assessing the Comprehension of UML Class Diagrams via Eye Tracking*, pages 113–122. IEEE, 2007.

Making the Case for Measuring Mental Effort*

Stefan Zugal
University of Innsbruck
Technikerstraße 21a
6020 Innsbruck, Austria
stefan.zugal@uibk.ac.at

Jakob Pinggera
University of Innsbruck
Technikerstraße 21a
6020 Innsbruck, Austria
jakob.pinggera@uibk.ac.at

Hajo Reijers
Eindhoven University of Technology
PO Box 513
NL-5600 MB Eindhoven, The Netherlands
h.a.reijers@tue.nl

Manfred Reichert
Universität Ulm
Building O27,
James-Franck-Ring
89069 Ulm, Germany
manfred.reichert@uni-ulm.de

Barbara Weber
University of Innsbruck
Technikerstraße 21a
6020 Innsbruck, Austria
barbara.weber@uibk.ac.at

ABSTRACT

To empirically investigate conceptual modeling languages, subjects are typically confronted with experimental tasks, such as the creation, modification or understanding of conceptual models. Thereby, accuracy, i.e., the amount of correctly performed tasks divided by the number of total tasks, is usually used to assess performance. Even though accuracy is widely adopted, it is connected to two often overlooked problems. First, accuracy is a rather insensitive measure. Second, for tasks of low complexity, the measurement of accuracy may be distorted by peculiarities of the human mind. In order to tackle these problems, we propose to additionally assess the subject's mental effort, i.e., the mental resources required to perform a task. In particular, we show how aforementioned problems connected to accuracy can be resolved, that mental effort is a valid measure of performance and how mental effort can easily be assessed in empirical research.

Categories and Subject Descriptors

G.3 [**Probability and Statistics**]: Experimental design

General Terms

Experimentation, Human Factors, Measurement

1. INTRODUCTION

Over the years, numerous conceptual modeling languages and associated modeling tools have been proposed [15]. In order to allow for an objective discrimination whether new proposals come along with improvements, the adoption of empirical software engineering has been advocated [4, 26]. Certainly, empirical research has been shown to be suitable for putting discussions on an objective basis. Still, in order to contribute to truly objective results, a valid experimental setup, as well as valid measurement methods are indispensable—slightest changes in the design might lead to fundamentally different outcomes [12].

In this work, we focus on empirical research that involves human activities, such as the creation, modification and understanding of conceptual models. Therein, various methods have been applied to identify differences. In particular, researchers have used modeling tasks [20], modification tasks [7] and sets of questions [5] to assess performance of conceptual modeling languages. In order to *measure* the outcome of tasks, typically *accuracy* and *duration* are analyzed (cf. [5, 7, 11, 20, 28]). Accuracy thereby refers to the percentage of correctly performed tasks, whereas duration refers to how long a subject required to perform a task. Even though accuracy is well-established and widely adopted, it is connected to two often overlooked problems. First, in order to identify differences with respect to accuracy, subjects need to commit errors. Hence, subtle differences that may be relevant, but do not lead to errors, cannot be identified (e.g., slight improvement of comprehensibility). Second, it has been shown that for tasks that are easy, humans tend to make mistakes that are actually not caused by the modeling notation, but are rather the result of peculiarities of the human mind [10]. In order to overcome these problems and to improve validity of the collected data, we propose to additionally assess the subject's *mental effort*, i.e., the mental resources required for performing the task. We would like to stress that we *do not* propose replacing accuracy and duration, but rather using mental effort as an additional perspective that potentially provides further insights. The contribution of this paper is twofold. First, we argue for measuring mental effort on the basis of literature. Second, we will substantiate our claims with experiences drawn from own experiments.

*This research is supported by Austrian Science Fund (FWF): P23699-N23

The remainder of this paper is structured as follows. Section 2 discusses problems related to accuracy and how to address them using mental effort. Insights from experiments making use of mental effort are presented in Section 3 and afterwards discussed in Section 4. Section 5 presents related work and Section 6 concludes with a summary.

2. MEASURING MENTAL EFFORT

In the following we start by discussing the previously described problems in more detail. Then, we introduce *mental effort* to address the aforementioned problems.

Problems Concerning Accuracy. In the introduction, we briefly mentioned that accuracy is of rather low sensitivity and potentially incorrect for tasks of low complexity. Issues regarding the sensitivity become clear when looking at the definition of accuracy. Usually, accuracy is defined to be the ratio of correctly performed tasks (e.g., correct answers) divided by the number of all performed tasks (e.g., total amount of questions). In other words, subjects have to commit mistakes in order to obtain a lower accuracy. If a task performed in the course of an experiment is not difficult enough to provoke errors, no differences can be observed with respect to accuracy, also known as *ceiling effect* [25]. Likewise, if differences between experimental tasks are not large enough, no differences can be observed.

In addition, for tasks of low complexity a further problem arises—it has been recognized that for such tasks subjects tend to commit more careless mistakes. In [10], this phenomenon is explained by *Dual-Process Theory* [22]. Roughly speaking, this theory postulates that the human brain consists of two systems, *S1* and *S2*. S1 processes are characterized as being fast, unconscious and effortless. S2 processes, in contrast, are slow, conscious and effortful. In addition, S2 serves as monitor of the fast automatic responses of S1. Apparently, in certain situations, S1 comes up with a fast response and S2 does not intervene—leading to answers that are fast but incorrect. Hence, for tasks of low complexity, it may be the case that accuracy does not reflect the task's difficulty, but rather this peculiarity of the human mind.

Up to now we have discussed problems associated with measuring accuracy, i.e., low sensitivity and potential problems when assessing accuracy for tasks of low complexity. In the following, we introduce the concept of *mental effort* and illustrate how it can be used to overcome these problems.

Measuring Mental Effort. In general, the human brain can be seen as a *"truly generic problem solver"* [24]. Within the human brain, cognitive psychology differentiates between working memory that contains information currently being processed, as well as long-term memory in which information can be stored for a long period of time [17]. Most severe, and thus of high interest, are limitations of the working memory. As reported in [2], working memory cannot hold more than about seven items at the same time. The amount of working memory currently used is thereby referred to as *mental effort*. The importance of the working memory has been recognized and led to the development and establishment of Cognitive Load Theory, meanwhile widespread and empirically validated in numerous studies [3].

To measure mental effort, various techniques, such as rating scales, pupillary responses or heart-rate variability are available [17]. Especially rating scales, i.e., self-rating mental effort, has been shown to reliably measure mental effort and is thus widely adopted [9, 17]. Furthermore, this kind of measurement can easily be applied, e.g., by using 7-point rating scales. For instance, in [13] mental effort was assessed using a 7-point rating scale, ranging from *(1) very easy* to *(7) very hard* for the question *"How difficult was it for you to learn about lightning from the presentation you just saw?"*.

In the context of conceptual models, mental effort is of interest as it appears to be connected to performance, e.g., properly answering questions about a model. In general, it is known that errors are more likely to occur when the working memory's limits are exceeded [23]. Similarly, in [14] it is argued that higher mental effort is in general associated with lower understanding of models.

Based on these insights, we argue that mental effort is connected to performance, i.e., accuracy and duration. In contrast to accuracy, however, subtle differences can presumably be observed. In particular, for cases where mental effort is well within the working memory's limits and thus does not provoke a significant amount of errors, still a different mental effort could be observed. In addition, for tasks of low complexity, careless mistakes may distort the measurement of accuracy. For mental effort, however, it can be expected that careless mistakes will not distort the measurement.

3. MENTAL EFFORT IN EMPIRICAL RESEARCH

So far, our arguments for measuring mental effort are based on insights from literature. In the following, we will complement the discussion with findings we gained in own experiments. For each experiment, we will shortly sketch the setup on a very abstract level and point out relevant findings related to the measurement of mental effort.

3.1 Experiment 1: Test Cases in Declarative Business Process Modeling

Background. In this experiment, we investigated whether the presence of test cases supports the maintenance of declarative business process models, as argued in [32]. In the context of this work, the relevant information is that subjects were asked to adapt conceptual models with different types of operational support.

Setup. We employed a randomized, balanced single-factor experiment with repeated measurements (cf. [27]). The factor was *adoption of test cases*, having factor levels *test cases* as well as *absence of test cases*. The experiment's objects were change assignments for two declarative process models. Measured response variables relevant for this work were *mental effort* and *accuracy*, i.e., the amount of errors conducted (details are provided in [31]). To assess mental effort, we employed a 7-point rating scale, ranging from *Extremely low mental effort (1)* to *Extremely high mental effort (7)* for the question *"How would you assess the mental effort for completing the change tasks (with tests)?"*. For assessing the impact of factor level *absence of test cases*, the phrase *"with tests"* was replaced by *"without tests"*.

Execution of Experiment. The experiment was conducted in December 2010 in a course on business process management at the University of Innsbruck; in total 12 students participated. Operational support for collecting demographic data was provided by Cheetah Experimental Platform (CEP) [21], modeling assignments were conducted using Test Driven Modeling Suite (TDMS) [30].

Findings Relevant to Mental Effort. The collected data indicated that subjects, who had test cases at hand, conducted fewer errors, however, the differences were not significant (Wilcoxon Signed-Rank Test, Z = -0.857, p = 0.391). Interestingly, the data indicated a lower mental effort for subjects who had test cases at hand. However, in this case the differences could be found to be significant (Wilcoxon Signed-Rank Test, Z = -2.565, p = 0.010). A detailed analysis showed that the provided models were too simple to provoke the desired effects, i.e., differences with respect to the amount of errors committed. In fact, 96% of the tasks were performed correctly, leaving almost no room for improvement. Still, the models were difficult enough to achieve significant results with respect to mental effort. Knowing that errors are more likely to occur when the working memory's limits are exceeded [23], these results seem plausible. Even though the tasks were not difficult enough to go beyond the limit of the subjects' working memory and thereby provoking errors, different levels of mental effort were required. Put differently, it appears as if in this case measuring mental effort provided a more sensitive method than accuracy.

3.2 Experiment 2: Test Cases in Declarative Business Process Modeling (Replication)

Background. In this experiment, we further explored this research direction, i.e., the background is identical to Experiment 1.

Setup. Since this experiment is a replication of Experiment 1, the setup is identical, except for more complex models[1].

Execution of Experiment. The experiment was conducted in December 2011 in a course on business process management at the University of Ulm; in total 31 students participated. Again, CEP [21] and TDMS [30] were used as operational support.

Findings Relevant to Mental Effort. Data analysis showed that subjects who had test cases at hand conducted significantly less errors (Wilcoxon Signed-Rank Test, Z = -3.165, p = 0.002) and required significantly less mental effort (Wilcoxon Signed-Rank Test, Z = -3.867, p = 0.000). Interestingly, the total amount of correctly performed tasks dropped from 96% in Experiment 1 to 80% in this experiment. Hence, the two key observations are, as follows. First, apparently a certain level of complexity was required to provoke enough errors and to make differences with respect to accuracy significant. Second, mental effort consistently showed significant differences in both experiments. In other words, as discussed in Section 2, mental effort and accuracy seem connected, however, a certain level of complexity is required for accuracy in order to show significant differences.

[1]Material can be downloaded from:
http://bpm.q-e.at/experiment/TDMReplication

3.3 Experiment 3: Hierarchy in Business Process Models

Background. In this experiment we investigated the impact of hierarchy on the understandability of BPMN models. In the context of this work, the essential part is that we elaborated pairs of information-equivalent models, one of them making use of hierarchy. Then, we asked subjects to answer questions about those models, measuring accuracy of answers, duration and mental effort.

Setup. We employed a randomized, balanced single-factor experiment with repeated measurements (cf. [27]). The factor was *hierarchy* with factor levels *flat* as well as *hierarchical*. The experiment's objects were two BPMN-based business processes. Measured response variables relevant for this work were *accuracy of answers*, *duration* and *mental effort*[2]. In contrast to Experiment 1 and Experiment 2, where mental effort was assessed once for each subject, in this experiment we measured the expended mental effort after each question. To assess mental effort, we used a 7-point rating scale ranging from *Extremely low mental effort (1)* to *Extremely high mental effort (7)*. The question for measuring mental effort was: *"Please indicate your mental effort for answering this question (question X)"*.

Execution of Experiment. The experiment was conducted in a course on business process management at the University of Eindhoven in January 2012; in total 114 students participated. Again, CEP [21] was used for presenting the models to subjects and collecting answers.

Findings Relevant to Mental Effort. The assessment of accuracy, duration and mental effort *per question*, as opposed to Experiment 1 and Experiment 2, where mental effort was assessed once for the entire experiment, allowed for a much more fine grained analysis. In the course of this experiment, 2 BPMN-based business process models were presented to each subject. For each model, 8 questions were asked, leading a total of 16 questions per subject. Since we expected different mental effort, accuracy and duration, depending on whether a question was posed for a hierarchical model or a flat model, responses were analyzed separately, leading to a total of 32 data points. In the following, we will discuss this data from two perspectives. First, we will present a case in which accuracy did not reflect the difficulty of a task, but rather peculiarities of the human mind. Second, we will take a closer look into the relation between mental effort, accuracy and duration.

Accuracy for Tasks of Low Complexity. In Section 2 we argued that measurement of accuracy might lead to unexpected results—in the following, we provide further empirical evidence. In particular, the third question in this experiment yielded an average mental effort of 3.14, accuracy of 0.79 and duration of 40 seconds when asked for the hierarchical model. If the same question was posed for the information-equivalent model without hierarchy, mental effort increased to 3.75, duration increased to 51 seconds, but also the accuracy increased to 0.91. Statistically speaking, a Mann-Whitney U test showed that mental ef-

[2]Material can be downloaded from:
http://bpm.q-e.at/experiment/Hierarchy

fort increased significantly (z = -3.271, p = 0.001), also the duration increased significantly (z = -4.468, p = 0.000). Apparently inconsistently, also the average accuracy increased, even though not significantly (z = -1.717, p = 0.086)—according to previous findings, *higher* mental effort should have been associated with *lower* accuracy.

In order to resolve this contradiction, we investigated the aforementioned question in detail. The analysis showed that it should have been harder to answer the question for the non-hierarchical model, i.e., lower accuracy could be expected. In particular, for answering the question in the hierarchical model, 13 BPMN nodes had to be taken into account—for the non-hierarchical model, however, 92 nodes had to be considered[3]. Knowing that this amount of nodes required the subjects to scroll considerably to see all relevant model elements, it seems surprising that actually a *higher* accuracy could be observed. However, in the light of Dual-Process Theory [22], these results can be explained as follows. For the hierarchical model, the question could be answered easily, as indicated by the average mental effort of 3.14 (approximately *Low mental effort*). Hence, it seems plausible that system S1 provided a quick, but incorrect answer that was not overridden by S2. In the non-hierarchical model, subjects were forced to scroll to determine the answer, i.e., involving conscious activities, hence activating S2. The activation of S2, in turn, ensured that the question was answered in a controlled way, instead by a quick response of S1. Summarizing, it seems as if relying on accuracy would have been misleading in this case, while mental effort and duration provided more reliable results.

Validity of Mental Effort. So far we have provided empirical evidence that mental effort is more sensitive than accuracy and can be measured properly for tasks of low complexity. In the following, we will provide empirical evidence that mental effort is tightly connected to accuracy and duration, i.e., is a valid measure of performance. To visualize the interplay between mental effort and accuracy, we used a scatter plot (cf. Figure 1). Therein, the x-axis represents the average mental effort required for answering a question. Values from 1 to 7 represent the rating scale used for assessing mental effort, ranging from *Extremely low mental effort (1)* to *Extremely high mental effort (7)*. Likewise, the y-axis reflects a question's average accuracy, i.e., the ratio of correct answers to total answers given for a question. As discussed in Section 2, higher mental effort should be associated with lower performance. Hence, in this particular case, higher mental effort should be associated with lower accuracy. In fact, in Figure 1, a tendency toward lower accuracy with increased mental effort can be observed. In particular, the dashed line represents the regression line as obtained through simple linear regression (R^2 = 0.41, F(1,30) = 21.16, p = 0.000). Please note that this regression does not contradict the case when mental effort and accuracy do not perfectly correlate, as demonstrated in the example above. Rather, the regression is not perfect, hence leaving room for such idiosyncrasies.

Akin to Figure 1, in Figure 2, the interplay between mental effort and duration is illustrated. Likewise, the x-axis

[3]The models and question can be accessed through: http://bpm.q-e.at/misc/HierarchyQuestion3

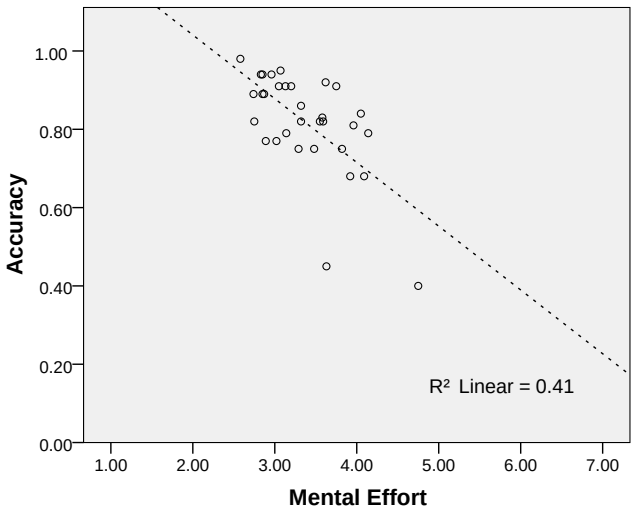


Figure 1: Mental effort versus accuracy

represents mental effort. On the y-axis, the average amount of seconds required for answering a question can be found. The dashed line is the result of simple linear regression (R^2 = 0.55, F(1,30) = 36.70, p = 0.000). Interestingly, in this case higher mental effort is associated with higher duration. In the light of the background presented in Section 2, also this result seems plausible. The more difficult a questions is to answer, the longer the answering process will take and the higher the mental effort will be.

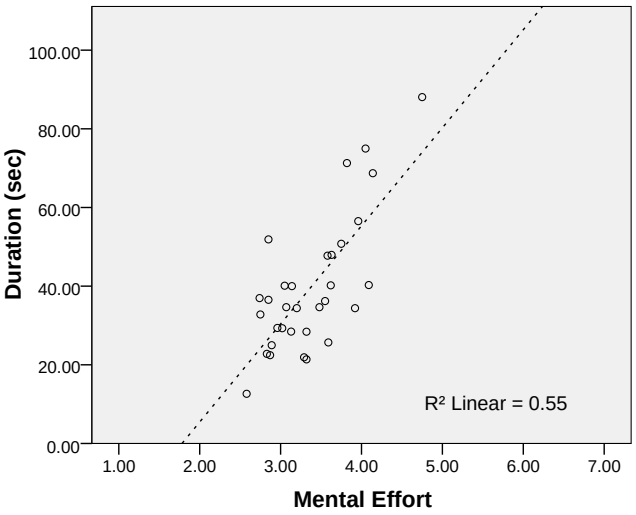


Figure 2: Mental effort versus duration (sec)

To substantiate these observations, we computed Pearson Correlation coefficient for correlations between mental effort, accuracy and duration. As shown in Table 1, the findings coincide with the observations made in Figures 1 and 2. In particular, the results confirm that mental effort and accuracy are correlated negatively (r(30) = -0.643, p = 0.000); mental effort and duration are correlated positively (r(30) = 0.742, p = 0.000). Finally, accuracy and duration are

correlated negatively (r(30) = -0.459, p = 0.008).

		Mental Eff.	Duration
Accuracy	Pearson Corr.	-0.643**	-0.459**
	Sig. (2-tailed)	0.000	0.008
	N	32	32
Mental Eff.	Pearson Corr.		0.742**
	Sig. (2-tailed)		0.000
	N		32

**. Correlation is significant at the 0.01 level (2-tailed).

Table 1: Correlations

4. DISCUSSION

Up to know we argued that accuracy is presumably rather insensitive and may be distorted for tasks of low complexity. In order to tackle these problems, the measurement of mental effort was proposed. In the following, key insights as well as limitations of this approach are discussed.

Regarding sensitivity, Experiment 1 and Experiment 2 provided empirical evidence that mental effort is more sensitive than accuracy. In Experiment 1 tasks of rather low complexity were provided to the subjects. Even though differences with respect to accuracy and mental effort could be observed, only mental effort differed significantly [31]. In Experiment 2 the task complexity was increased, consequently more errors were committed. Knowing that errors are more likely to be committed when the working memory is overloaded [23], these observations seem plausible. In Experiment 1, different levels of mental effort could be observed. However, the working memory was not overloaded, resulting in a low error rate and hence marginally fluctuations in accuracy. In Experiment 2, increased complexity lead to an overload of working memory, accordingly the error rate increased and accuracy dropped. In other words, it seems likely that differences with respect to mental effort can be observed before differences with respect to accuracy occur, i.e., mental effort appears to be more sensitive.

Regarding tasks of low complexity, Experiment 3 provided further insights. In particular, we could observe an increase of mental effort and duration that was connected to increased accuracy—actually a decrease of accuracy could be expected, as far more model elements had to be taken into account. As indicated in [10], it seems as if this result can be traced back to peculiarities of the human mind, which tends to commit more careless mistakes for tasks of low complexity. Hence, in such a case it seems as if the measurement of mental effort provides a more accurate picture. Please note that this finding does not contradict the correlation between mental effort and accuracy, as found in Experiment 3. Rather, the correlation is valid in general, while this peculiar interplay could be found for one specific question.

Apparently, several limitations apply to this work. First, as shown in Figure 2, a *linear relationship* between mental effort and duration could be found. Even though this seems plausible for short-lasting tasks (the maximum average duration was about 90 seconds), it seems questionable in how far this holds for longer tasks. For instance, a long-lasting repetitive task, such as finding all elements named *"A"* within a model, will most likely lead to a low mental effort, *but* a long duration. Second, mental effort is a subjective measure. Even though it has been shown that people are able to give a numerical indication of their mental burden [16], it is questionable whether mental effort of different subjects can be compared directly. Hence, it seems advisable to ensure a reasonable sample size when conducting between-subject experiments or to focus on within-subject experiments. Third, we reported from consistent results across three experiments. Still, our findings may be peculiarities of these experiments. To improve the generalization, more experiments adopting different modeling languages are required.

5. RELATED WORK

In the domain of cognitive psychology, the work of Paas et al., in which mental effort is discussed broadly, is directly connected [17]. In contrast to this work, however, mental effort is not linked to conceptual modeling. In the domain of conceptual modeling, related work can be found where model understandability is of concern. For instance, Aranda et al. provide guidelines for assessing model understandability [1]. Besides accuracy and duration, *perceived difficulty* is proposed to be measured. However, in contrast to this work, no indications on how perceived difficulty can be measured, are given. Likewise, [11] assesses in a survey how understandability of models is measured. The most prominent measure is accuracy, followed by duration and perceived ease of understanding. However, these studies rather rely on the ease-of-use scale from *Technology Acceptance Model* [6] than on rating scales for measuring mental effort. Recently, mental effort has also been used as a tool for discussing model understandability. For instance, in [29] the role of mental effort in Business Process Modeling is discussed. Likewise, in [28, 33] mental effort is employed for assessing the impact of hierarchy on model understandability. In contrast to this work, however, mental effort is rather used as a tool for discussion; the measurement of mental effort is not of concern. Apparently, measuring mental effort is only meaningful if the validity of the experimental design can be ensured. In this respect [8, 18] provide guidelines for the proper operationalization of measurements.

6. SUMMARY AND OUTLOOK

In this work, we showed how measuring mental effort allows to compensate for shortcomings when measuring accuracy. In particular, we argued that mental effort is more sensitive than accuracy and that the measurement is not distorted for tasks of low complexity. Hence, it allows to identify subtle differences between conceptual models or conceptual modeling languages. Likewise, when data regarding accuracy is affected by ceiling effects, mental effort can still provide valuable insights. In addition, we showed that the measurement of mental effort can be implemented easily through the adoption of rating scales. Thereby, we recommend to measure mental effort after *each* task in order to provide a fine-grained measurement. With this contribution we hope to help in paving avenues for even more comprehensive empirical investigations.

Future work will imply the collection of further data for a deeper investigation of the interplay between mental effort, accuracy and duration. In particular, we plan to adopt eye

movement analysis [19] to constantly monitor mental effort while subjects perform a task.

7. REFERENCES

[1] J. Aranda, N. Ernst, J. Horkoff, and S. Easterbrook. A Framework for Empirical Evaluation of Model Comprehensibility. In *Proc. MISE '07*, pages 7–12, 2007.

[2] A. Baddeley. Working Memory: Theories, Models, and Controversies. *Annu. Rev. Psychol.*, 63(1):1–29, 2012.

[3] M. Bannert. Managing cognitive load—recent trends in cognitive load theory. *Learning and Instruction*, 12(1):139–146, 2002.

[4] J. C. Carver, E. Syriani, and J. Gray. Assessing the frequency of empirical evaluation in software modeling research. In *Proc. EESSMod '11*, pages 28–37, 2011.

[5] J. A. Cruz-Lemus, M. Genero, M. E. Manso, S. Morasca, and M. Piattini. Assessing the understandability of UML statechart diagrams with composite states—A family of empirical studies. *Empirical Software Engineering*, 25(6):685–719, 2009.

[6] F. Davies. *A Technology Acceptance Model for Empirically Testing New End-User Information Systems: Theory and Results*. PhD thesis, Sloan School of Management, 1986.

[7] A. M. Férnandez-Sáez, M. Genero, and M. R. V. Chaudron. Does the level of detail of uml models affect the maintainability of source code? In *Proc. EESSMod '11*, pages 3–17, 2011.

[8] A. Gemino and Y. Wand. A framework for empirical evaluation of conceptual modeling techniques. *Requir. Eng.*, 9(4):248–260, 2004.

[9] D. Gopher and R. Brown. On the psychophysics of workload: Why bother with subjective measure? *Human Factors: The Journal of the Human Factors and Ergonomics Society*, 26(5):519–532, 1984.

[10] I. Hadar and U. Leron. How intuitive is object-oriented design? *Communications of the ACM*, 51(5):41–46, 2008.

[11] C. Houy, P. Fettke, and P. Loos. Understanding understandability of conceptual models: What are we actually talking about? In *Proc. ER '12*, pages 64–77, 2012.

[12] R. Laue and A. Gadatsch. Measuring the Understandability of Business Process Models - Are We Asking the Right Questions? In *Proc. BPD '10*, pages 37–48, 2011.

[13] R. Mayer and P. Chandler. When learning is just a click away: Does simple user interaction foster deeper understanding of multimedia messages. *Journal of Educational Psychology*, 93(2):390–397, 2001.

[14] D. L. Moody. Cognitive Load Effects on End User Understanding of Conceptual Models: An Experimental Analysis. In *Proc. ADBIS '04*, pages 129–143, 2004.

[15] J. Mylopoulos. Information modeling in the time of the revolution. *Information Systems*, 23(3/4):127–155, 1998.

[16] F. Paas. Training strategies for attaining transfer of problem-solving skill in statistics: A cognitive-load approach. *Journal of Educational Psychology*, 84(4):429–434, 1992.

[17] F. Paas, A. Renkl, and J. Sweller. Cognitive Load Theory and Instructional Design: Recent Developments. *Educational Psychologist*, 38(1):1–4, 2003.

[18] J. Parsons and L. Cole. What do the pictures mean? Guidelines for experimental evaluation of representation fidelity in diagrammatical conceptual modeling techniques. *DKE*, 55(3):327–342, 2005.

[19] J. Pinggera, M. Furtner, M. Martini, P. Sachse, K. Reiter, S. Zugal, and B. Weber. Investigating the Process of Process Modeling with Eye Movement Analysis. In *Proc. ER-BPM '12*, to appear.

[20] J. Pinggera, P. Soffer, S. Zugal, B. Weber, M. Weidlich, D. Fahland, H. Reijers, and J. Mendling. Modeling Styles in Business Process Modeling. In *Proc. BPMDS '12*, pages 151–166, 2012.

[21] J. Pinggera, S. Zugal, and B. Weber. Investigating the process of process modeling with cheetah experimental platform. In *Proc. ER-POIS '10*, pages 13–18, 2010.

[22] K. E. Stanovich and R. West. Individual differences in reasoning: implications for the rationality debate? *Behavioural and Brain Sciences*, 23(5):665–726, 2000.

[23] J. Sweller. Cognitive load during problem solving: Effects on learning. *Cognitive Science*, 12(2):257–285, 1988.

[24] W. J. Tracz. Computer programming and the human thought process. *Software: Practice and Experience*, 9(2):127–137, 1979.

[25] W. P. Vogt. *Dictionary of Statistics & Methodology: A Nontechnical Guide for the Social Sciences (Fourth Edition)*. SAGE Publications, 2011.

[26] C. Wohlin, M. Höst, and K. Henningsson. Empirical research methods in software engineering. In *Empirical Methods and Studies in Software Engineering*, volume 2765 of *LNCS*, pages 7–23. Springer, 2003.

[27] C. Wohlin, R. Runeson, M. Halst, M. Ohlsson, B. Regnell, and A. Wesslen. *Experimentation in Software Engineering: an Introduction*. Kluwer, 2000.

[28] S. Zugal, J. Pinggera, J. Mendling, H. Reijers, and B. Weber. Assessing the Impact of Hierarchy on Model Understandability—A Cognitive Perspective. In *Proc. EESSMod '11*, pages 123–133, 2011.

[29] S. Zugal, J. Pinggera, and B. Weber. Assessing process models with cognitive psychology. In *Proc. EMISA '11*, pages 177–182, 2011.

[30] S. Zugal, J. Pinggera, and B. Weber. Creating Declarative Process Models Using Test Driven Modeling Suite. In *Proc. CAiSE Forum '11*, pages 16–32, 2011.

[31] S. Zugal, J. Pinggera, and B. Weber. The impact of testcases on the maintainability of declarative process models. In *Proc. BPMDS '11*, pages 163–177, 2011.

[32] S. Zugal, J. Pinggera, and B. Weber. Toward Enhanced Life-Cycle Support for Declarative Processes. *Journal of Software: Evolution and Process*, 24(3):285–302, 2012.

[33] S. Zugal, P. Soffer, J. Pinggera, and B. Weber. Expressiveness and Understandability Considerations of Hierarchy in Declarative Business Process Models. In *Proc. BPMDS '12*, pages 167–181, 2012.

Micro-business Behavior Patterns associated with Components in a Requirements Approach

RJ Macasaet[1,2]
[1]Pentathlon Systems Resources Incorporated, Department of Software Research and Development
Cityland Pioneer, Pioneer Street, Mandaluyong City, 1550 Metro Manila, Philippines
[2]University of Granada, Departamento de Lenguajes y Sistemas Informáticos
Calle Periodista Daniel Saucedo Aranda S/N, 18071 Granada, España
rjmacasaet@pentathlonsystems.com, rjmacasaet@ugr.es

Manuel Noguera
University of Granada, Dpt. de Lenguajes y Sistemas Informáticos
C/ Pdta. Saucedo Aranda S/N, 18071 Granada, España
mnoguera@ugr.es

María Luisa Rodríguez
University of Granada, Dpt. de Lenguajes y Sistemas Informáticos
C/ Pdta. Saucedo Aranda S/N, 18071 Granada, España
mlra@ugr.es

José Luis Garrido
University of Granada, Dpt. de Lenguajes y Sistemas Informáticos
C/ Pdta. Saucedo Aranda S/N, 18071 Granada, España
jgarrido@ugr.es

Sam Supakkul
University of Texas at Dallas,
Department of Computer Science,
Richardson, Texas 75083, U.S.A.
ssupakkul@ieee.org

Lawrence Chung
University of Texas at Dallas,
Department of Computer Science,
Richardson, Texas 75083, U.S.A.
chung@utdallas.edu

ABSTRACT
Micro-businesses are the smallest enterprises and since they come in large numbers and are greatly diversified, they become difficult to define and classify. Micro-businesses also have several resource restrictions. These ambiguities and constraints make software research and development difficult in the micro-business domain. Component-based development offers advantages for the software of micro-businesses. The reuse of components for common requirements minimizes resource consumption in their software projects. This paper provides a working definition for micro-businesses, observations of their behavior, working micro-business behavior patterns, and examples of real world applications on how the patterns help in software development through requirements. The micro-business behavior patterns are associated with components that will be used later on in the development of micro-business software systems.

Categories and Subject Descriptors
D.2.13 [**Software Engineering**]: Reusable Software – Reuse Models

General Terms
Documentation

Keywords
Micro-businesses, SMEs, Business Process Modeling, Behavior Patterns, Components, CBD, Non-Functional Requirements, Requirements Engineering

1. INTRODUCTION
Micro-businesses are the smallest kind of small-to-medium sized enterprise (SME) and are referred to in several ways. Since many of these references are conflicting, coming to a consensus regarding its definition is difficult [1]. Micro-businesses can be classified as an enterprise with less than 10 people [2] although headcount may not be considered the best metric [3]. The age of a business [4] or the length of their (software) projects [5][3] could be used as alternative metrics. Classifying a business based on their degree of collaboration on software projects [6] or on their degree of adaptability to software systems [7] may also be suitable.

In order to avoid confusion in this paper, micro-businesses are defined as businesses which require only a few (less than 10) components for their software. In this definition, components are referred to as independent software modules which encapsulate a certain set of functions. These components can be replaced with other components or newer versions and modified without affecting other components.

The goal of component-based development (CBD) is to develop software systems by reusing pre-existing software components rather than rewriting them from scratch. In this

context, there is a general understanding of components as reusable building blocks in a software project [8]. CBD offers great opportunities for micro-businesses (and for SMEs in general) which normally are unable to purchase and maintain entire software systems autonomously [9]. Micro-businesses have tight budgets, limited manpower, lack technical exposure and maturity [7][10], and usually have no requirements processes suitable for their size [11][12].

Micro-businesses must define their requirements properly even with their size [13] because if software requirements are not expressed properly, there will eventually be problems during design, implementation [14], and acceptance, which all in all threaten the success of the software project [15]. Misunderstood requirements are a major cause of development inefficiency and project failure [16]. Hence, software process improvement efforts for small and medium firms (including micro) are continuously being made particularly in the requirements phase [17].

In order to improve the chances of success of micro-business software projects, attempts to improve their requirements process must be made. Such processes must not be costly, must not demand too much manpower [18], must not be too technical [13], must have simple and comprehensible models [19], must not consume too much time, must be "lightweight but effective" [20], and it must be "good enough" [21] for the stakeholders involved. Although trade-offs are made for micro-business requirements processes (in comparison to requirements processes for larger businesses with more resources), all stakeholders [22] end up "satisficed" [23] upon project completion, i.e., satisfying at some level a variety of needs, without necessarily optimizing results.

This paper presents micro-business behaviour patterns within a requirements approach [24] and explains how they are observed, specified, structured, and applied in real world cases. These micro-business behaviour patterns contribute to the requirements process and help improve the reusability of software components, thus increasing their quality.

This paper is structured as follows. Section 2 discusses related work. Section 3 discusses the micro-business behaviour pattern in detail wherein the subsections discuss how they are modeled, observed, and applied in real world cases. Finally, section 4 discusses conclusions and future work.

2. RELATED WORK

CBD and the use of patterns can contribute to the improvement and feasibility of requirements processes for micro-businesses. Recent CBD work [8] suggests that patterns are understood as valuable architectural design aids applicable to recurring problems in specific design contexts. Although it is difficult to classify patterns, it is generally accepted that architectural patterns [25], design patterns [26], and workflow patterns [27] exist and that these behavioral patterns could provide valuable guidance to stakeholders in a (micro-business) software project, especially during the requirements phase. Furthermore, initial studies have been made regarding requirements patterns specifically pertinent to the domain of commercial off-the-shelf (COTS) systems [28].

Micro-business behavior patterns which are part of a requirements approach [24] are presented in this paper. The patterns represent repeatedly occurring micro-business behavior in the real world and are associated with software components. The patterns aid CBD by improving the reusability of the software components in similar micro-businesses. In this field of research, the patterns contribute to the literature by sharing current real-world experiences and proposing behavioral models pertinent specifically to the domain of micro-businesses. The patterns are discussed in further detail in the following section.

3. THE MICRO-BUSINESS BEHAVIOR PATTERN

This section is organized as follows. The first subsection presents a modeling approach for a micro-business behavior pattern. The second subsection discusses how micro-business behavior patterns are observed in the real world. Finally, the last subsection illustrates real world applications of micro-business behavior patterns.

3.1 A Modeling Approach for a Micro-business Behavior Pattern

The concepts, artifacts, and relationships of the micro-business behavior pattern are modeled below in Fig. 1.

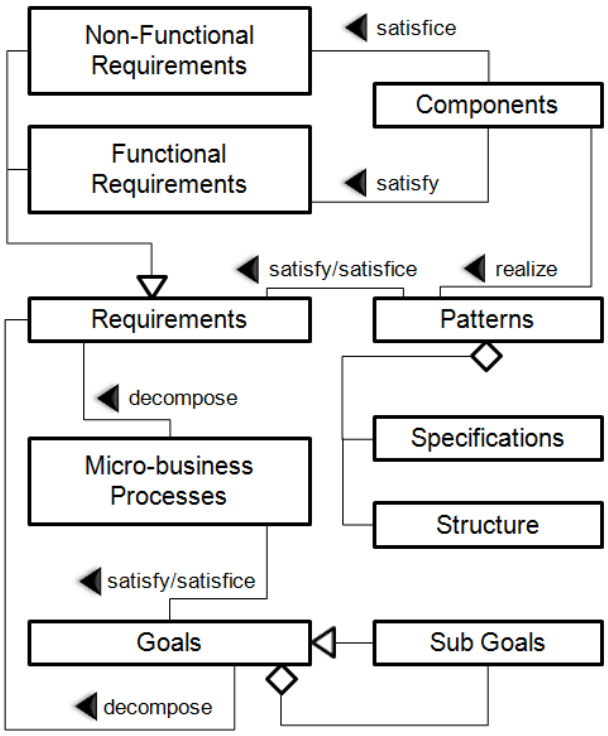


Figure 1. The Micro-business Behavior Pattern

The goals of the micro-business owner are satisfied or satisficed with micro-business processes. The information from the micro-business processes, goals, and sub-goals are decomposed into requirements. Methods which decompose goals into requirements can be applied here. Some examples of these methods can be found on p.198 of Requirements Engineering: Processes and Techniques [29] and in the goal modeling section of KAOS [11].

The requirements are further broken down into functional and non-functional requirements. This paper defines a functional requirement as what the system can do [30] whilst a non-functional requirement as the ability of the system to do it (the

functional requirement) [18]. A framework with step-by-step instructions which links goals to functional and non-functional requirements is presented on p. 198 of Requirements Engineering: Processes and Techniques [29].

The software components are those which satisfy the functional requirements and satisfice the non-functional requirements. These components are also those which realize or "make real" the micro-business behavior pattern. One or several components may realize a said micro-business behavior pattern.

The micro-business behavior pattern has specifications and structure (which will be illustrated in detail in subsection 3.3). The specifications include text and keywords which describe the pattern, a spreadsheet or list of the functional requirements it satisfies, and development-related information such as the authors and the development time spent on the components (expressed in man days).

The structure of the micro-business behavior pattern includes an image (or diagram) of the pattern which could also show other related micro-business behavior patterns (as shown in Figs. 2 and 3), a micro-business profile sample preset which guides software developers on component reuse, and a non-functional requirements priority profile list which helps define the priorities of the micro-business owner, their customers, and those of the software developer.

3.2 Observing Micro-business Behavior in the Real World

The first micro-business under observation is a clothes retail store. The clothes retail store owner is expanding and opening another store nearby. The owner would like to consolidate all his sales data at any point in time so that he could be more responsive in making important daily business decisions such as knowing when to order more of which kind of product, when to pay more attention to which store, and the like. The responsiveness of the software is very important for him.

The second micro-business under observation is a fruit store. The fruit store owner wants to know his sales totals at the end of each day. The fruits in his store spoil very quickly so he only wants to order specific amounts of fruit on certain days in order to avoid wastage. He wants the software deployed within a week.

The third micro-business under observation is a wine bar. The wine bar owner wants to know his daily sales totals and wants to know if the number of wine glasses served are proportionate to the number of wine bottles consumed. His customers come in and out of the bar quickly especially during peak hours and these customers demand to pay and obtain a receipt instantly.

Upon observing these three micro-businesses, common behaviors can be identified. First, all of these micro-business owners would like to know their sales figures. Second, all the micro-business owners would like to know their inventory levels. Following the goals of CBD, there is no need to develop the same software for each micro-business again and again if certain components can be reused.

3.3 Applying Micro-business Behavior Patterns in the Real World

This subsection will explain micro-business behavior patterns in detail (the structure and specifications) and how they are applied in the real world cases mentioned in the previous subsection (3.2) – the clothes retail store, the fruit stand, and the wine bar.

The first common need of the micro-business owners is that they would all want to know their sales figures. In this case, the micro-business behavior pattern which can be applied to these three micro-businesses is a Point-of-Sale "POS" Behavior Pattern. The specifications and structure of this behavior pattern are presented below. Its diagram is illustrated in Fig. 2. Special notes are marked with an "*" in bold.

Point-of-Sale "POS" Behavior Pattern Specifications

Text Description:
The point of sale "POS" process involves recording sales, being able to display totals, and providing a customer with a receipt (or confirmation of sale).
***Take note that this text description fits the common needs of the three micro-businesses mentioned in subsection 3.2.**

Keywords:
POS, Sale, Sales, Point of Sale, Receipt

List of Functional Requirements:
1. Record Sale(s)
2. Display Sales Total(s)
3. Provide Receipt(s)

Development-related Information:
1. Authors and their last edit date (chronological) – Jerrick Lim 053010, Philip Santos 060710
2. Total development time spent on software component(s) – 42 man days
3. Average customization time of software component(s) during deployments – 10.5 man days

Point-of-Sale "POS" Behavior Pattern Structure

Non-Functional Requirements Priority Profiles (check top priority, priorities can also be ranked, 1 as top):
Micro-business Perspective
- Low-cost
- Quick deployment ***Checked by Fruit Stand Owner** – There could be discussions on increasing the number of software developers during deployment in order to meet the deadline.
- Quick (responsive) software ***Checked by Clothes Retail Store Owner** – Since two sites are involved, there will be discussions on the reliability of the internet provider(s).
- Easy-to-use, User-friendly software
- Ease of Maintenance
- Secure Database
- Portability of software

Developer Perspective
- High Margin
- Quick deployment
- Ease of Implementation
- Ease of Maintenance

Customer Perspective
- Quick (responsive) software ***Checked by Wine Bar Owner** - There could be discussions on considering the purchase of more powerful hardware in order to make the software more responsive during operations and to print receipts faster.
- Security

Micro-business Profile Sample Presets (check one) – There are specific software components which pertain to each selection. This grouping of components helps in re-using them, eventually improving their quality:

Supermarket
- Stand-alone (only one physical branch)
- Multiple (several physical locations)

Retailer
- Stand-alone (only one physical branch) ***Checked by Fruit Stand Owner**
- Multiple (several physical locations) ***Checked by Clothes Retail Store Owner**
- Online (only virtual sales)
- Total (physical and virtual sales)

Wholesaler
- Micro-distributor
- Regional
- National
- Continental
- International

Restaurant
- Fast-food
- Dine-in
- Refreshments and Bar ***Checked by Wine Bar Owner**
- Delivery
- Fine-dining
- Multi-service

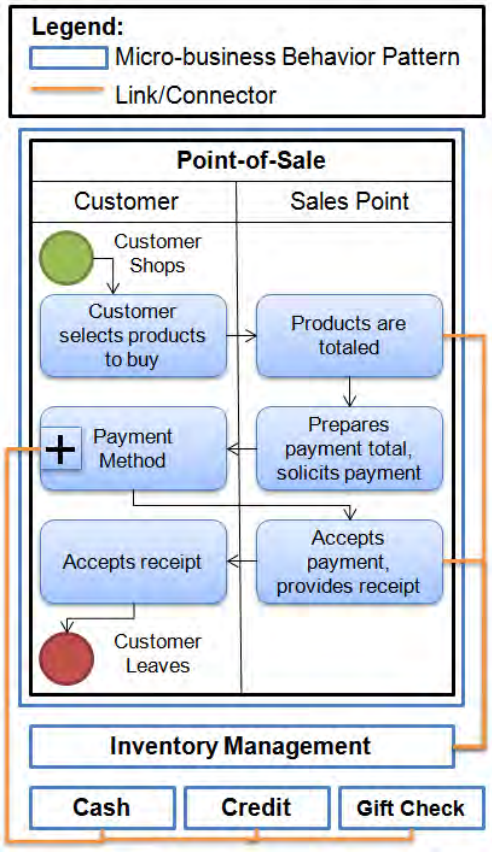


Figure 2. The Point-of-Sale Behavior Pattern

In this case, the diagram has been expressed in business process modeling notation (BPMN) [31]. BPMN has been gaining wide acceptance [32] and is meant to be "readily understandable by all business users" [31]. Such notation is ideal for use in a requirements approach [24] which aims to be "not too technical" [13], comprehensible, and simple enough [19] for micro-business owners who are not too technically exposed [7][10] and who normally view these diagrams during their day-to-day project meetings.

Take note that the links/connectors which are represented as orange lines could eventually be some kind of pattern in their own right [8]. Other micro-business behavior patterns which are shown in Fig. 2 and are connected by the orange lines are the Cash Payment Behavior Pattern, the Credit Card Payment Behavior Pattern, the Gift Check Payment Behavior Pattern, and the Inventory Management Behavior Pattern. The payment behavior patterns represent repeatedly occurring payment behaviors of customers (which are also considered micro-business behavior).

The Inventory Management Behavior Pattern which is shown as a related pattern in Fig. 2 addresses the second common need of the micro-business owners (as discussed in subsection 3.2) which is that they would all want to monitor their inventory. The specifications and structure of the Inventory Management Behavior Pattern are presented below. Its diagram is illustrated in Fig. 3. Special notes are marked with an "*" in bold.

Inventory Management Behavior Pattern Specifications

Text Description:
Inventory management involves monitoring and recording the duration of the storage of items.
***Take note that this text description fits the common needs of the three micro-businesses mentioned in subsection 3.2.**

Keywords:
Inventory, Warehouse, Item, Storage

List of Functional Requirements:
1. Record the storage of an item or several items.
2. Record the removal of an item or several items.
3. Display current inventory.
4. Record the storage time of an item or several items.

Development-related Information:
1. Authors and their last edit date (chronological) – Kerwin Velasco 042810, Rudolph Lavarias 051910
2. Total development time spent on software component – 35 man days
3. Average customization time of software component during deployments – 8 man days

Inventory Management Behavior Pattern Structure

Non-Functional Requirements Priority Profiles (check top priority, priorities can also be ranked, 1 as top):
Micro-business Perspective
- Low-cost
- Quick deployment ***Checked by Fruit Stand Owner** – There could be discussions on increasing the number of software developers during deployment in order to meet the deadline.
- Quick (responsive) software ***Checked by Clothes Retail Store Owner** – Since two sites are involved, there will be discussions on the reliability of the internet provider(s).

- Easy-to-use, User-friendly software
- Ease of Maintenance
- Secure Database
- Portability of software

Developer Perspective

- High Margin
- Quick deployment
- Ease of Implementation
- Ease of Maintenance

Customer Perspective

- Quick (responsive) software ***Checked by Wine Bar Owner*** - There could be discussions on considering the purchase of more powerful hardware in order to make the software more responsive during operations and to print receipts faster.
- Security

Micro-business Profile Sample Presets (check one) – There are specific software components which pertain to each selection. This grouping of components helps in re-using them, eventually improving their quality*:*

Time-related

- Space rental
- Expiration date/Spoilage control ***Checked by Fruit Stand Owner***
- Ingredients inventory (Food-related)

Non-time related

- Raw materials inventory (Manufacturing)
- Junk Yard Storage

Generic POS attachment

- Raw ***Checked by Clothes Retail Store Owner and Wine Bar Owner***
- Custom

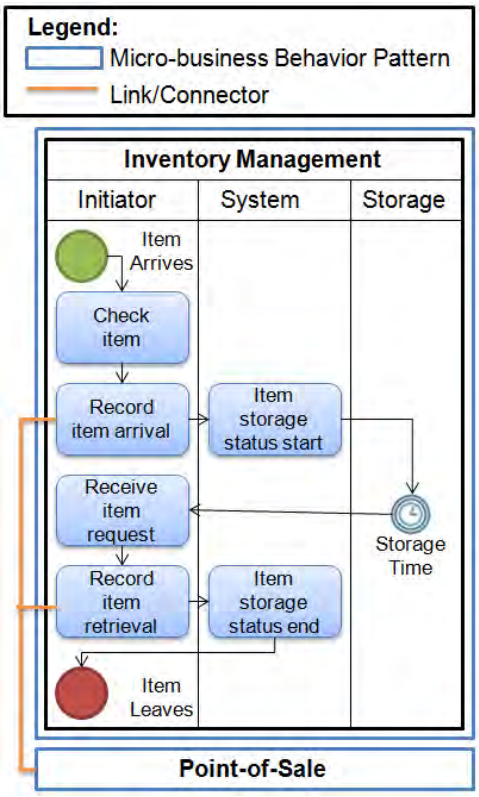


Figure 3. The Inventory Management Behavior Pattern

4. CONCLUSIONS AND FUTURE WORK

The large number and diversity of micro-businesses make them difficult to define and classify. Further compounded with several resource restrictions, such ambiguities and constraints make software research and development in the micro-business domain an extra challenge. A requirements approach [24] using micro-business behavior patterns and CBD techniques could help improve the success of software projects for micro-businesses. The reuse of components for common requirements minimizes resource consumption in similar software projects.

This paper provides a working definition for micro-businesses, a modeling approach for micro-business behavior patterns, and examples of how such patterns are applied in real world cases. The micro-business behavior patterns are associated with components that will be used later on in the development of micro-business software systems. These micro-business behavior patterns are being used in a requirements approach [24] in a real world software development company, Pentathlon Systems Resources Incorporated. The authors of this paper and the developers at the software company are continuously striving to improve the structure and specification of the patterns for better use.

Future research work involves the creation of a systematic search method for the micro-business behavior patterns. Currently, keyword-based methods are being used. Ontological and case-based methods are currently being studied for feasibility and practicality.

5. ACKNOWLEDGEMENTS

This research has been funded by the Innovation Office of the Andalusian Government through project TIN-6600, the Spanish Ministry of Economy and Competitiveness through project TIN2012-38600, CEI BioTIC Granada under project 20F2/36, and Pentathlon Systems Resources Incorporated.

6. REFERENCES

[1] Merten, T., Lauenroth, K., and Bürsner, S. 2011. Towards a New Understanding of Small and Medium Sized Enterprises in Requirements Engineering Research. In *Proceedings of the 17th International Working Conference on Requirements Engineering: Foundation for Software Quality REFSQ*, Essen, Germany, 60-65.

[2] European Commission. 2008. The New SME Definition User Guide and Model Declaration. URL = http://www.ec.europa.eu/enterprise/enterprise_policy/sme_definition/sme_user_guide.pdf (Last accessed on December 2, 2011)

[3] Aranda, J. 2010. Playing to the Strengths of Small Organizations. In *Proceedings of the 1st Workshop on RE in Small Companies RESC, 141-144.*

[4] Nikula, U., Sajeniemi, J., and Kalvianen, H. 2000. A state-of-the-practice survey on requirements engineering in small-and-medium-sized enterprises. In *Telecom Business Research Center Lappeenranta Research Report 1, Lappeenrata University of Technology.*

[5] Aranda, J., Easterbrook, S.M., and Wilson, G. 2007. Requirements in the Wild: How Small Companies do it. In *Proceedings of the 15th IEEE International Requirements Engineering Conference RE*, Delhi, India.

[6] Jantunen, S. 2010. The Benefit of Being Small: Exploring Market-Driven Requirements engineering Practices in Five Organizations. In *Proceedings of the 1st Workshop on RE in Small Companies RESC*, pp. 131-140.

[7] Kamsties, E., Hormann, K., and Schlich, M. 1998. Requirements Engineering in Small and Medium Enterprises: State-of-the-Practice, Problems, Solutions, and Technology Transfer. In *Conference on European Industrial Requirements Engineering CEIRE*, London, United Kingdom.

[8] Elizondo, P. and Lau, K. 2010. A Catalogue of Component Connectors to Support Development with Reuse. *Journal of Systems and Software 83*, 2010, 1165-1178. DOI = http://dx.doi.org/10.1016/j.jss.2010.01.008

[9] Turowski, K. 2000. Establishing Standards for Business Components. In *K. Jacobs (Eds.): Information Technology Standards and Standardisation: A Global Perspective*. Hershey, 131-151.

[10] Kauppinen, M., Tapani, A., Kujula, S., and Laura, L. 2001. Introducing Requirements Engineering: How to Make a Cultural Change Happen in Practice: Helsinki University of Technology. In *Software Business and Engineering Institute*. DOI = http://doi.ieeecomputersociety.org/10.1109/ICRE.2002.1048504

[11] Respect-IT. 2007. KAOS Tutorial Version 1.0. URL = http://www.objectiver.com/fileadmin/download/documents/KaosTutorial.pdf (Last accessed on March 10, 2011)

[12] Lamsweerde, A. 2001. Goal Oriented Requirements Engineering: A Guided Tour. *Invited mini-tutorial paper which appeared in Requirements Engineering, International Symposium on Toronto, August 2001, 249-263. Proceedings RE'01 5th IEEE*. DOI = http://doi.ieeecomputersociety.org/10.1109/ISRE.2001.948567

[13] Young, R. 2004. *Requirements Engineering Handbook*. Artech House, Incorporated, Norwood, Massachusetts.

[14] Kauppinen, M., Vartiainen, M., Kontio, J., Kujula, S., and Sulonen, R. 2004. Implementing Requirements Engineering Processes Throughout Organizations: Success Factors and Challenges. In *Information and Software Technology 46*, pp. 937-953.

[15] Davis, C.J., Fuller, R.M., Tremblay, M.C., and Berndt, D.J. 2006. Communication Challenges in Requirements Engineering and the Use of the Repertory Grid Technique. In *Journal of Computer Information Systems, 46, (5), 78*.

[16] Maalej W., Happel, H.J., and Seedorf, S. Applications of Ontologies in Collaborative Software Development. 2010. In *I. Mistrik, J Grundy, A. van der Hoek, and J. Whitehead (Eds.), Collaborative Software Engineering*, Berlin, Heidelberg.

[17] Pino, F., García, F., and Piattini, M. 2008. Software Process Improvement in Small and Medium Software Companies: A Systematic Review. In *Software Quality Control, Volume 16, Issue 2, June 2008*. 237-261. DOI = 10.1007/s11219-007-9038-z.

[18] Holcombe, M. 2008. *Running an Agile Software Development Project*. Wiley, Hoboken, New Jersey, USA.

[19] Kruchten, P. 2003. *The Rational Unified Process*. Addison-Wesley Professional.

[20] Ambler, S. 2002. *Agile Modeling*. John Wiley and Sons.

[21] Bach, J. 1997. Good Enough Quality: Beyond the Buzzword. In *IEEE Computer Society, vol. 30, no. 8, August 1997*, pp. 96-98.

[22] Niazi, M.K. 2002. Improving the Requirements Engineering Process through the Application of a Key Process Areas Approach. In *Australia Workshop on Requirements Engineering*.

[23] Simon, H. 1992. *The Sciences of the Artificial*. MIT Press.

[24] Macasaet, R., Chung, L., Garrido, J., Rodriguez, M., and Noguera, M. 2011. An Agile Requirements Elicitation Approach based on NFRs and Business Process Models for Micro-businesses. In *Proceedings of the 12th International Conference on Product Focused Software Development and Process Improvement PROFES* (Torre Canne, Italy, June 20-22, 2011). ACM, New York, NY, 50-56. DOI = http://doi.acm.org/10.1145/2181101.2181114.

[25] Kuchana, P. 2004. *Software Architecture Design Patterns in Java*. Auerbach Publications, Boston, MA, USA.

[26] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. 1995. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley.

[27] Russell, N., ter Hofstede, A.H.M., van der Aalst, W.M.P., Mulyar, N., 2006. *Workflow control-flow patterns: A revised view*. Technical Report 34, BPM Center, BPM-06-22.

[28] Méndez, O., Franch, X., Quer, C. 2008. Requirements Patterns for COTS Systems. In *Proceedings of the 7th International Conference on Composition-Based Software Systems, Madrid, Spain on February 25-29, 2008*. DOI = 10.1109/ICCBSS.2008.34

[29] Kotonya, G. and Sommerville, I. 2003. *Requirements Engineering: Processes and Techniques*. John Wiley and Sons Limited, England.

[30] Sommerville, I. 2004. *Software Engineering, Seventh Edition*. Pearson Education.

[31] Object Management Group, Inc. 2008. Business Process Modeling Notation Version 1.1. URL = http://www.omg.org/spec/BPMN/1.1/PDF (Last accessed on March 10, 2011)

[32] Koskela, M. and Haajanen, J. 2007. Business Process Modeling and Execution: Tools and Technologies Report for the SOAMeS Project. In *VTT Research Notes 2407, VTT Technical Research Centre of Finland*.

Business Process Modelling: Five Styles and a Method to Choose the Most Suitable One

Gianna Reggio, Maurizio Leotta, Filippo Ricca, Egidio Astesiano
DIBRIS, Università di Genova, Italy
{ gianna.reggio | maurizio.leotta | filippo.ricca | egidio.astesiano }@unige.it

ABSTRACT

A software developer facing a modelling task may follow different styles at different levels of abstraction and precision, to better cope with the aims and the potential users of the model. We address the problem of modelling the business processes by means of UML activity diagrams, and present five styles differing in the precision level, from the Ultra-Light style, where the nodes and the edges of the activity diagram are decorated by freely-formed text, to precise styles where instead OCL and UML actions are used. Then, we propose a practical empirical method for choosing the most suitable style depending on the context in which the models will be used (why, when, where, how long, by whom).

Categories and Subject Descriptors:
D.2.2 [**Software Engineering**]: Design Tools and Techniques

General Terms: Design, Documentation.

Keywords: UML, Business Process Modelling, Styles.

1. INTRODUCTION

The Unified Modeling Language (UML) is used to model many disparate aspects of different software and systems, in all the phases of the development process, with different aims, and by different kinds of stakeholders (e.g., business analysts and developers). This is possible because the UML offers a large number of constructs and allows to leave out any detail. Thus, a modeller when facing with a modelling task may follow different styles[1]. This has been recognized since long time. For example, Fowler proposes three different ways to build UML models: *UML as Sketch*, *UML as Blueprint*, and *UML as Programming Language* [4]. Within the UML, the activity diagrams are used to model and visualize the flows of control (and of data) in different entities, such as systems, objects, use cases, and operations.

[1]A particular manner or technique by which something is done, created, or performed (Merriam Webster's Dictionary)

Activity diagrams have been also used to model business processes [3, 5]. Modellers may follow different styles when modelling the business processes using the activity diagrams, e.g., the few guidelines suggested in [1]. An examination of the publicly available activity diagrams shows that the large majority of the modellers are completely undisciplined and produce activity diagrams without following any style. Moreover, except [1], no other relevant proposals are available. Also prompted by the participation in various research projects conducted in cooperation with the industry, we have then defined various styles for modelling the business processes with the UML activity diagrams that differ for the degree of precision of the produced models. These styles range from the Ultra-Light, where no guidelines drive the modellers, to styles for producing precise[2] models, where OCL and UML actions are exploited to decorate arcs and nodes, respectively. Each style is motivated by some specific modelling activities made in some specific context by specific persons.

Then, we have tried to evaluate these five styles. For example, for what concerns the comprehension [7] we got some empirical evidence that a precise model is easier to comprehend. By another empirical investigation we have found that producing models following the Ultra-Light style may results in making many mistakes and errors, that can be detected and corrected when revising such models following a precise style [6]. The precise models seem to be "better"; however, they have also problematic aspects, e.g., their production requires more effort and work of people with a good UML knowledge. Thus, in this paper we address the problem of finding the most suitable style when modelling the business processes using the UML. We propose a method, based on our experience in the context of business process modelling, that, on the basis of the context in which the modelling will be done, evaluates the five styles by giving a value to their suitability in such context.

The main contributions of this paper are:

– a detailed presentation of the five styles (described in this paper exhaustively for the first time) ranging from the lighter to the more precise by means of a running example;
– a method able to help the decision makers to choose the most suitable style depending on the context in which the models will be used.

We present the five styles for the business modelling with the UML in Sect. 2. In Sect. 3 we propose our method to evaluate the five styles for choosing the most suitable one. Finally, in Sect. 4 we draw some conclusions.

[2]Exactly or sharply defined or stated (Merriam Webster's Dictionary)

2. THE FIVE STYLES

In this paper we assume the common intuitive meaning of *business process*, i.e.: "*A progression of tasks (activities, interactions, ...) that involve two or more entities, and create or add value to the organization's activities. In a sequential process, each step is dependent on occurrence of the previous step; in a parallel process, two or more steps can occur concurrently*"; and we will use the following terminology:

– the *(business process) participants* are the active entities performing the various tasks. We distinguish the participants that are human beings (and thus capable of autonomous activities) from those corresponding to software and hardware systems.
– the *(business process) objects* are the entities over which the activities of the process are performed, obviously these entities are passive, i.e., they are not able to do any activity by themselves.
– the *(business process) data* are the data used in the various tasks.

To present the five styles for Business Process modelling (shortly *BP modelling*), we will use as a running example the business process corresponding to ordering in an e-commerce site (EC), briefly described as follows:

A client sends an order. If the client is not already registered, (s)he will be asked to register to the site, if (s)he refuses the order will be cancelled. Then the order will be sent to the warehouse, which will prepare the package, and in the meantime either the handler of the credit cards or Paypal will be contacted (depending on the preferences of the client, expressed at the registration time) to get the payment; after the package will be sent; finally the carrier will inform that the package has been delivered, and the order will be archived.

In this section, we introduce five styles for BP modelling using the UML: Ultra-Light, Light, Disciplined, Precise Conceptual and Precise Operational, where the degree of precision is minimum in the first style and maximum in the last two. Fig. 1 presents the structure of the models of the business processes produced following the five styles.

2.1 The Ultra-Light Style

The simplest models are those produced following the *Ultra-Light style*: they just consist of an activity diagram produced without following any guideline, using[3] action nodes, control nodes, edges, and time and accept events, where the nodes and the edges are decorated by natural language text fragments freely formed; we can better name it *No-Style*, since the modeller is completely free to produce the activity diagram as s(he) likes. The Ultra-Light style is the most commonly used, and obviously it is the easiest and fastest to follow, but the Ultra-Light models are also easily full of mistakes, see, e.g., [6], and cannot be used for any kind of post elaboration.

[3]Swimlanes and object nodes are not treated here for space reasons.

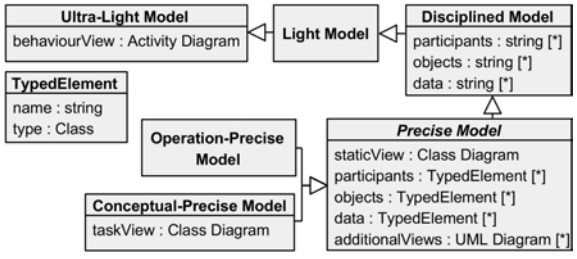


Figure 1: BP model structure for the five styles

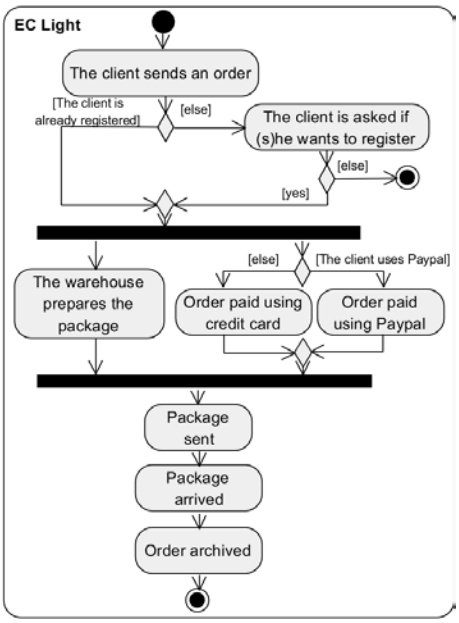


Figure 2: EC: Light Model

2.2 The Light Style

The *Light style* only imposes few restrictions (listed in the following) on the use of the visual constructs of the activity diagrams and on their layout, whereas the decorations of nodes and edges are still completely unconstrained: they are just natural language text fragments.

– For each decision node there must be a matching merge node and similarly for any fork node there must be a matching join node (exception can be made whenever a flow leaving the decision/fork node ends with a final node).
– One outgoing edge from a decision node must be labelled with the "else" guard.
– The flowing of the tokens should be depicted vertically, and the edges leaving a decision node should be depicted as follows: the edge corresponding to the regular/correct course of the events should be vertical, whereas the alternative corresponding to an error or an exceptional case should be depicted horizontally.

Notice that it may happen that the sentences defining the activities may be either in active or passive form (e.g., "Clerk fills the form" and "Form is filled"), and that the entity executing the activity may be precisely determined or left undefined (e.g., "Form becomes filled"); in other cases it is possible that nominal sentences are used instead of verbal phrases ("Filling the form"). Also the objects over which the business process activities are performed may be described in different ways, for example by a substantive (e.g., "Form", "The form") or by a qualificative sentence (e.g., "Client form", "Filled form").

Fig. 2 presents the Light model of the EC business process. As required, it is an activity diagram satisfying the constraints just introduced[4]. The various tasks are denoted by

[4]For space reasons the two edges leaving the second decision node are horizontal instead of almost vertical.

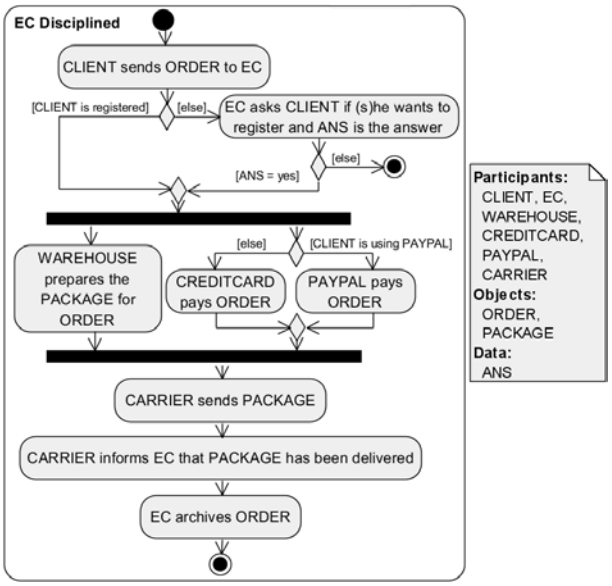


Figure 3: EC: Disciplined Model

natural language sentences having different structure; some are active and makes explicit the subject (e.g., The client sends an order), some others are in the passive form (Order archived) and so no information is given on who will perform the task. Notice how the relationship between the task The client is asked if (s)he wants to register and the subsequent decision node with an edge with guard yes is completely based on the reader understanding of the meaning of an English sentence.

2.3 The Disciplined Style

A business process model that follows the *Disciplined style* consists of (see Fig. 1): the participant/object/data lists (written using CAPITAL LETTERS), and an activity diagram describing the process behaviour, where:

– the action nodes are decorated by tasks described by simple natural language sentences having the form: "*subjects + present tense verb + object complements + other complements*" or "*subjects + present tense passive form + other complements*", where the *subjects* and the *object complements* are either participants or business objects of the process, and the data may appear in *other complements*.
– the guards on the edges leaving the decision nodes must be qualificative sentences about some of the participants/ objects/data of the business process, e.g., "*X is ...*", "*X has ...*", "$X \geq 10$".

Notice that the participants/objects are roles for the entities taking part in the business process and not specific individuals.

To determine the tasks of a business process it is important to keep in mind that they are assumed to be atomic in the context of the model of such process (i.e., it is not important/relevant to detail them further in term of actions of the various participants), but it is not always true that they correspond to elementary actions of the participants. For example it is ok to have a task of the form "The SELLER and the CLIENT exchanges the CONTRACT", as well as to give a more detailed model where the tasks are instead "The SELLER sends the CONTRACT" and "The CLIENT receives the CONTRACT".

The passive form must be used whenever who will execute the task is either not relevant or not known. "CLIENT pays INVOICE" is an example of active sentence, whereas "ORDER is archived" is a case of passive sentence; both follow the above constraints. The use of the passive style should be quite careful. If we do not want to describe or we do not know who are the participants of the business process, we can represent several tasks using the passive form. The resulting business process model will be quite abstract, and it may be then refined by transforming the passive sentences into active ones, after having determined the subjects.

Note that the guards cannot be actions of someone, for example "X answered yes" or "X accepts" cannot be a guard; in this case there should be a task corresponding to give an answer and then the decision will be about the answer, thus the guard will have the form "answer = Yes".

Fig. 3 shows the Disciplined model of the EC business process. Notice that in this model all the sentences are in active form, e.g., EC archives ORDER and EC registers CLIENT, because the style requests to find the participants lead us to detect the presence of EC, the system supporting the e-commerce. To represent the answer of the client to the registration proposal we have used a process data ANS; in this way the relationships between the guards and the previous task is clear, whereas in the Light model it was completely left to the reader's intuition.

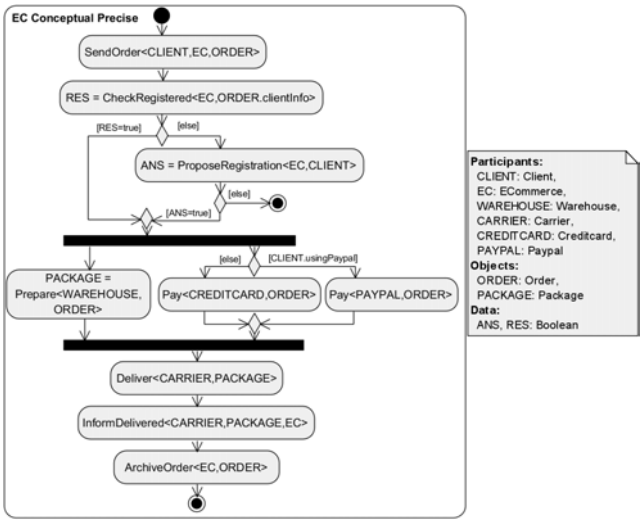
2.4 The Precise Style with Conceptual Tasks

The *Precise style with Conceptual Tasks style* (shortly *Precise Conceptual style)* for BP modelling requires to describe the participants, the objects and the data precisely by means of a class diagram, named *static view*, and to use an activity diagram to model the behaviour of the process, whereas the conditions on the edges leaving the decision nodes will be OCL expressions, and the action nodes will be decorated by elements of special classes stereotyped by ≪task≫, introduced by a class diagram named *task view*. Thus, a Precise Conceptual model consists of (see Fig. 1): *i)* a static view, i.e., a class diagram introducing the classes needed to type its participants, objects and data, *ii)* the participant/object/ data lists, *iii)* a task view, i.e., a class diagram introducing the task classes, *iv)* and an activity diagram representing its behaviour, satisfying the following constraints.

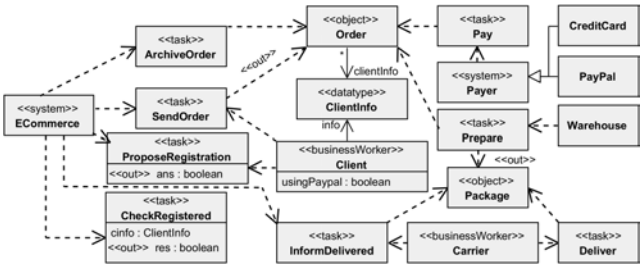
a) The classes in the static view must be stereotyped by ≪object≫ (business process objects), ≪businessWorker≫ and ≪system≫ (business process participants distinguished in autonomous entities, human beings and hardware/software systems); datatypes may be also included in this class diagram. The elements of those classes may be described using the many tools offered by the UML, for example constraints and behavioural diagrams.

b) The participants will have a name and will be typed by a class with stereotype either ≪businessWorker≫ or ≪system≫, the objects also will have a name and will be typed by a class stereotyped by ≪object≫, and the data will be typed by UML datatypes, either predefined or user defined in the static view. It is possible to constrain the possible participants, objects and data of a business process.

c) The fact that some participants and objects of the business process take part in a task is modelled by means of dependency relations linking the task class with the partici-



(a) Behaviour View (Activity Diagram)



(b) Static and Task Views (Class Diagram)

Figure 4: EC: Precise Conceptual Model

pant/object classes. The task classes should depend on the object classes (to depict that the tasks will act over them), and the participant classes should depend on the task class (to depict that they will take part in the tasks). Furthermore, the dependency between a task and an object class may be stereotyped by ≪out≫ if such business object is created during the task. If a task is characterized by some data, then those data are represented by attributes of the task class typed by datatypes. An attribute may be stereotyped by ≪out≫, in the case of a data produced by the task itself. Some constraints of kind invariant/pre/post may be attached to a ≪task≫ class expressing relationships holding always during/before/after the task execution among the involved entities. The behavioural aspects of a task instead may be modelled using some of the many constructs offered by the UML, e.g., sequence and again activity diagrams. The action nodes of the activity diagram modelling a business process will be labelled by instances of task classes, presented in the following way:

TaskName<x_1,...,x_n>

where TaskName is a task class, and x_1,...,x_n are the participants/objects/data involved in the task.

Fig. 4 shows the model of EC following the Precise Conceptual style. In this case we have put together the static and the task view and thus the model consists of a class diagram, an activity diagram and the participant/object/data lists.

The class diagram introduces the classes defining the participants and the objects, together with some datatypes used to describe them (for example ClientInfo). EC, PAYPAL and CREDITCARD are participants of kind system (they correspond respectively to the software system running the e-commerce site, the Paypal payment service and the credit card handling system), whereas CLIENT is a human participant and CARRIER and WAREHOUSE are respectively an external transport company and a department of the e-commerce company; they are not classified as systems since they cannot be fully automatized. Notice that the client is not involved in the task for delivering the package because it is assumed that the delivery will be made at a certain address and does not require an active participation of the client itself. Instead, the task informDelivered involves two participants, the CARRIER and the EC system.

2.5 The Precise Style "Operation Based"

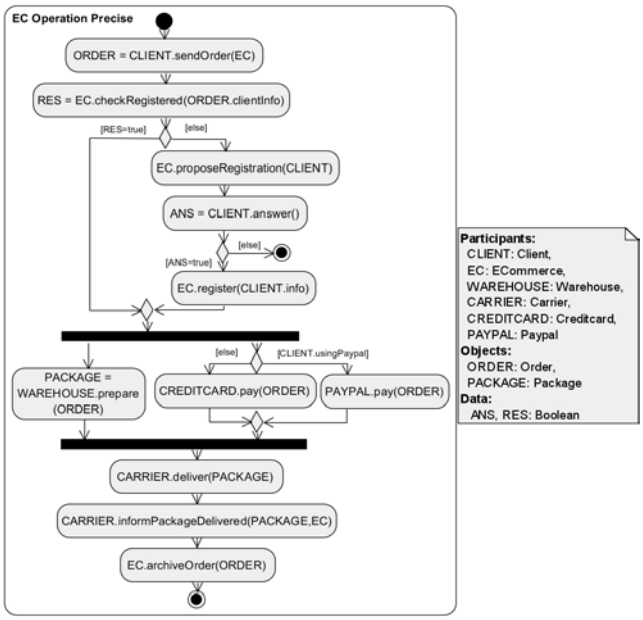
The *Precise style "Operation Based"* (shortly *Precise Operational style*) for BP modelling requires to describe the participants, the objects and the data precisely by means of a static view, as for the Precise Conceptual style of Sect. 2.4, and similarly the activity diagram modelling the behaviour of the process is presented in a precise way. The only difference concerns the way the UML is used to model the tasks: now the tasks are modelled by means of calls of operations of the participant or object classes. Thus, the Precise Operational model of a business process consists of (see Fig. 1): *i)* a static view, *ii)* the participant/object/data lists, *iii)* and an activity diagram representing its behaviour satisfying the following constraints.

The tasks involving the participants and the objects will be modelled by operations of the various participant/object classes stereotyped by ≪task≫ (whenever all the operations of a class have this stereotype it may be omitted to make the visual presentation simpler). When defining the ≪task≫ operations, it is important to keep in mind that *i)* an operation corresponding to a task part of a class C stereotyped by ≪businessWorker≫ or ≪system≫ describes a task that a participant of type C is responsible to perform and it should be named using a verb in the infinitive form without the "to"; *ii)* an operation corresponding to a task part of a class C stereotyped by ≪object≫ describes a tasks that will be done over an object of type C and it should be named by the past participle of a verb.

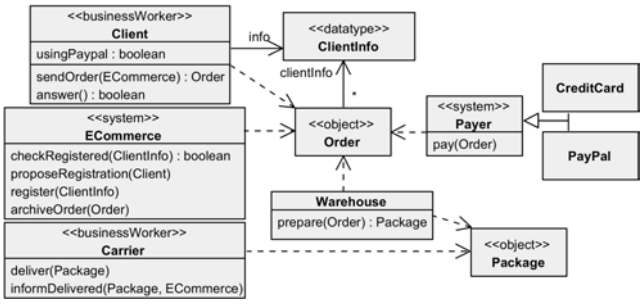
The action nodes of the activity diagram will be decorated either by calls of task operations on either participants or objects (and the participants, the objects and the data will freely appear as arguments of them) or by UML actions, i.e., assignment, creation and destruction of class instances; whereas the conditions on the edges leaving the decision nodes will be OCL expressions in which the participants, the objects and the data will freely appear.

Notice how the guidelines asking to define task operation either corresponding to operations over objects or to something that a participant is responsible to execute leads to have fine-grained tasks more or less of the same size, and finer than those of the Precise Conceptual style.

Fig. 5 presents the Precise Operational model of the EC business process consisting of a class diagram, an activity diagram, and the typed participant/object/data lists. The class diagram introduces the classes defining the participants/objects/data as for the Precise Conceptual model of Sect. 2.4. Since all the operations are stereotyped by ≪task≫ we have omitted to depict it visually in the class diagram.



(a) Behaviour View (Activity Diagram)



(b) Static View (Class Diagram)

Figure 5: EC: Precise Operational Model

Notice that in this model the request for registration to the client is implemented by three tasks: the request by EC, the client answer and then the registration of the same by EC, in case of positive answer; and thus differently than in the Precise Conceptual and in the Disciplined models. The reason is the style guideline asking to attach to each participant the tasks which are under his responsibility, and so we had to single-out the answering of the client.

In this example there are no task operations on the classes modelling the business process objects, because we have modelled explicitly who is performing the various tasks; instead, if we would prefer avoiding to express who is delivering the package, we can delete the class Carrier and add two operations to the class Package (requiredDelivering and delivered).

3. CHOOSE THE MOST SUITABLE STYLE

Even if the precise styles are better for what concerns the expressiveness and the quality of models, as shown in [6, 7], we think that the context in which the business process models appear (the why, when, where, how long, by whom the models are produced/used) and the wanted quality of the same models may influence the choice of the most suitable style. Hence, we have devised a method for choosing the most suitable style trying to balance precision and freedom, similarly to the proposal of [2] for choosing between agile and prescriptive software development methods.

Factors		Weights		Influence						Modelling Context	
				Ultra-Light	Light	Disciplined	Conceptual	Precise	Precise Operational		
F1	Modeller has a good knowledge of UML	W1	0.8	I1	-2	-1	0	1	1	MC1	[-1,0,1]
F2	Model reader has a good knowledge of UML	W2	0.8	I2	-2	-2	-2	1	1	MC2	[-1,0,1]
F3	Model used as a communication media	W3	0.8	I3	0	1	2	1	-1	MC3	[0,1]
F4	Model used in a MD Development as source of (semi-)automatic transformations	W4	0.8	I4	-2	-2	-1	1	2	MC4	[0,1]
F5	Model must be produced in a short time	W5	0.8	I5	2	1	0	-1	-2	MC5	[0,1]
F6	Model will have a long life span (and perhaps evolve)	W6	0.5	I6	-2	-1	1	1	1	MC6	[0,1]
F7	Modelled business is critical	W7	1.0	I7	-2	-1	0	2	2	MC7	[0,1]
F8	Model quality should be checkable	W8	0.5	I8	-2	-1	0	2	2	MC8	[0,1]

Figure 6: Factors and weights for choosing the most suitable style

To choose the most suitable style, we have created an algorithmic *suitability* estimation method. Our method allows the user to obtain a *suitability value* in a given context, for each style proposed in this paper, simply assessing the presence, in such context, of some factors, listed in Fig. 6. There is a group of factors concerning the environment in which the model will be developed and used (F1,..., F7), and a factor concerning the possibility to check the quality of the model itself (F8). These factors have been selected empirically[5]: the authors have independently proposed a list of factors, prompted by their (long for some of them) modelling experience in many different contexts and producing models of different quality; then such lists have been compared, discussed and then merged in a "not too long" common list (the considered factors were present in all the author lists).
A user of our method has just to decide which factors are present in her/his Modelling Context (MC) and return the numbers MC_1,...,MC_8 in the following way:

– for i = 1,2: if the factor F_i is present, then $MC_i = 1$;
if F_i is not present, then $MC_i = -1$;
if F_i is not relevant/known $MC_i = 0$;
– for i = 3,...,8: if the factor F_i is present, then $MC_i = 1$,
else $MC_i = 0$;

Then, the formula (**) will assign a suitability value to each style S and the style with the higher value will be considered the most suitable.

(**) Suitability of Style S: $ST_s = \sum_{i=1,...,8} W_i * I_{i,s} * MC_i$

where $S \in$ {Ultra-Light, Light, Discip., Prec. Conc., Prec. Oper.}

– $I_{i,s}$ represents the influence of factor F_i on the suitability of the style S: –2 when it is heavily deterring it and 2 when it is heavily favouring it: $-2 \leq I_{i,s} \leq 2$;
– W_i represents the importance of factor F_i: near 0 when the importance is negligible and 1 when it is of paramount importance: $0 < W_i \leq 1$.

To guarantee a sensible selection of the used weights (W_i, $I_{i,s}$, $i = 1, ..., 8$), the authors have empirically[5] estimated them. More in detail, all the authors have independently proposed some values for them, and then such values have been compared and discussed. In the cases of disagreement (few and where the difference was not excessive) a common value has been decided. The values of the various weights (W_i, $I_{i,s}$, $i = 1, ..., 8$) can be seen in Fig. 6.

[5]originating in or based on observation or experience (Merriam Webster's Dictionary)

	F1 (MC1)	F2 (MC2)	F3 (MC3)	F4 (MC4)	F5 (MC5)	F6 (MC6)	F7 (MC7)	F8 (MC8)	Ultra-Light	Light	Discip.	Prec. Conc.	Prec. Oper.
A)	-1	-1	1	0	1	0	0	0	**+4,8**	+4,0	+3,2	−1,6	−4,0
B)	1	1	0	1	0	1	1	1	−8,8	−6,0	−1,9	+5,9	**+6,7**
C)	1	-1	1	0	0	1	1	0	−3,0	+0,1	**+3,7**	+3,3	+1,7

Figure 7: Applications of the style suitability evaluation method

An Excel spread sheet freely available[6] allows to easily insert the values characterizing the presence of the various factors (MC_i, $i = 1, ..., 8$) and will do all the needed calculations, presenting the suitability values for the five styles (so the weights W_i, $I_{i,s}$, are reported here to give a complete explanation of how the style selection method works; the user have only to assess the presence of the various factors just filling the gray column in Fig. 6).

Let us discuss the various factors and their impact on the choice of the most suitable style, as formalized by the chosen weights. F1 and F2 take into account the knowledge of the UML by who produces and reads the model (notice how the three light styles are equally suitable for readers not fluent in the UML notation: since all of them require just the knowledge of the visual constructs of the activity diagrams). F3 and F4 concern the role of the produced model, and we consider two dimensions: whether it is relevant that the model is easy to read, because it is a piece of documentation, penalizing the choice of the precise-styles and heavily-favouring the Disciplined style, and whether it has to be used as the starting point of some (semi-)automatic transformations, favouring the precise styles (mainly the Precise Operational). Then, we consider whether it is important to produce the model in a short time (F5), since the time obviously increases as the style becomes more precise. Factor F6 is related to the span life of the model and thus to the possibility to undergo some evolution, which equally favours the Disciplined and precise styles (indeed, the structuring of the textual decoration of the Disciplined activity diagram is a great support to evolution, think, e.g., to replace a participant with another one, in this case it will be easy to find all the tasks in which s(he) was taking part). F7 is the most influential factor for the choice of a precise style, and again heavily favours the precise styles (in this case in the same way, since they differ only in the level of abstraction, not in rigor). F8 concerns the possibility to check the quality of the model, i.e.: are you interested in checking if (1) is the model complete and minimal and without UML errors? (2) are rigorous inspections on the model doable? (3) are complexity metrics on the model definable?

We have then validated these weights by applying the evaluation method to many cases out of the possible $576 = 3^2 * 2^6$ (several cases are reported in the Excel spreadsheet). Here we report three typical modelling contexts: **A)** a draft model of a business process to be discussed with the stakeholders; **B)** a model to be used as the starting point of the (semi-)automatic generation of a BPEL implementation of a system supporting a business process; and **C)** a model of the process for getting the approval of a new building project by the local administrations in Italy. In Fig. 7, we present the values characterizing the three cases and the evaluation results. We got that for case **A)** the Ultra-Light style is the most suitable (+4,8), for case **B)** the winner style is the Precise Operational (+6,7), and, finally, for case **C)** the winner is the Disciplined style (+3,7). These results are quite reasonable: **A)** the Ultra-Light model is acceptable since it will be just used to discuss on the process with stakeholder, and will be heavily modified before to reach a stable form; **B)**: the Precise Operational is the most suitable style to get a model to transform into a running system using BPEL; **C)** the Disciplined style allows a quite precise description of the things to be done and the item to be handled but at a quite abstract level and in a way easy to read for non-technical persons, not cluttering the model with too many details, but at the same time helping to avoid mistakes that may have very serious consequences (e.g., to name in the same way two slightly different documents or tasks).

[6] `http://softeng.disi.unige.it/TR/Styles.xls`

4. CONCLUSION AND FUTURE WORK

In this paper, we have first presented five styles, differing in the precision level, for modelling the business processes by means of UML activity diagrams. Then, we have devised a practical empirical approach to choose a style among the proposed ones, keeping in consideration the context in which they will be used and for what.

In this work, we focused on business process modelling and chose UML activity diagrams for their representation. However, we think that our work (styles and our method for the choice of the style) can be generalized for other UML diagrams, notations, purposes and in other contexts/settings with some rework. For example, a similar proposal could be put forward for UML state machines used to describe the behaviour of an entity or for business processes expressed by means of BPMN.

As future work, we would like to test more systematically and deeply our practical approach. In particular, we intend to better validate our weights with additional cases (i.e., modelling contexts) and try our approach in an industrial context. Another interesting direction could be re-thinking the approach using machine learning techniques.

5. REFERENCES

[1] S. W. Ambler. *The Elements of UML 2.0 Style*. Cambridge University Press, 2005.

[2] B.W. Boehm and R. Turner. *Balancing agility and discipline: a guide for the perplexed*. Addison-Wesley.

[3] E. Di Nitto, L. Lavazza, M. Schiavoni, E. Tracanella, and M. Trombetta. Deriving executable process descriptions from UML. In *Proceedings of ICSE 2002*, pages 155–165.

[4] M. Fowler and S. Kendall. *UML Distilled: A Brief Guide to the Standard Object Modeling Language (4th Edition)*. Addison-Wesley Professional, 2010.

[5] S. Jurack, L. Lambers, K. Mehner, G. Taentzer, and G. Wierse. Object flow definition for refined activity diagrams. In *Proceedings of FASE 2009*, pages 49–63.

[6] G. Reggio, M. Leotta, and F. Ricca. "Precise is better than Light" A Document Analysis Study about Quality of Business Process Models. In *Proceedings of EmpiRE 2011*, pages 61–68. IEEE, 2011.

[7] G. Reggio, F. Ricca, G. Scanniello, F. Di Cerbo, and G. Dodero. A precise style for business process modelling: Results from two controlled experiments. In *Proceedings of MODELS 2011*, volume 6981 of *LNCS*, pages 138–152. Springer.

Modelling and Managing Variability with Feature Assembly – An Experience Report

Lamia Abo Zaid
Vrije Universiteit Brussel
Pleinlaan 2
1050 Brussels, Belgium
Lamia.Abo.Zaid@vub.ac.be

Olga De Troyer
Vrije Universiteit Brussel
Pleinlaan 2
1050 Brussels, Belgium
Olga.DeTroyer@vub.ac.be

ABSTRACT
Feature models have been commonly used to model the variability and commonality in software product lines. We have defined the Feature Assembly Modelling, a feature modelling technique that allows to model variability in software adopting a multi perspective approach. Furthermore, the approach allows modelling software by combining both variability and reusability, i.e. we have developed an approach to take reusability into account while defining new software. To support the approach, we have also developed an information retrieval framework that provides an interactive visualization of the feature models. The visualization allows users to explore and query the existing models. In this paper, we report on our experience in introducing this variability modelling approach into a small-scale software company. This experience was very useful for both parties. The company was able to uncover the structure of their software and the modelling exercise provided them better insight in their products. For us, it has helped to better understand the needs of companies, to evaluate the usability of our Feature Assembly approach and the associated learning curve, as well as revealing its current limitations. Moreover, as we are aware of the fact that classical feature modelling is not yet a practice adopted by companies, it was interesting to see that our approach was well accepted and appreciated by the company.

Categories and Subject Descriptors
D.2.13 [**Software Engineering**]: Reusable Software – ***Domain Engineering.***

D.2.10 [**Software Engineering**]: Design – ***Methodologies, and Representation.***

General Terms
Design, Management.

Keywords
Feature Models, Feature Assembly, Software Product Lines, Variability, Variability Management, Experience Report

1. INTRODUCTION
Companies developing multiple related software products or products having different variants (i.e. software product lines) are faced with many challenges as the complexity of the software increases considerably by introducing variability. For instance, how can they keep an overview on the many different variants of the products produced and installed for different customers; how do the different features of the variant products relate to each other; which dependencies exist between the different features; what is the impact of making some changes to a certain feature; etc. We claim that in order to avoid problems in later stages of the software life cycle, it is necessary to perform a thorough modelling phase of the variability in a product (or family of products) in order to reveal the complexity introduced by common and variable features, feature relations, and feature dependencies.

Different feature oriented modelling techniques (e.g., [1] [2]) exist that allow to model variability and commonality in software. The term "feature" is used to denote an abstraction that different stakeholders can understand [1]. Despite the relevance of feature modelling for industry, a recent study [3] reveals that there are very few reports in the feature modelling literature on the application of feature models in practice. Furthermore, a few of the detected papers discuss successful and unsuccessful applications of feature modelling in practice. Results show that companies often had doubts when applying the feature modelling technique and its usefulness was not fully anticipated beforehand [3].

Furthermore, it is agreed upon both in academia and industry, that analysis and design are often underestimated when developing new products [4] [5]. The impact of good design is obvious, yet good practice remains a challenge. Furthermore, it was found that in small and medium scale companies variability is not planned beforehand [4] [6] but actually evolves with time due to the expansion of the software to serve more customers or due to the need to customize some features to meet the different needs of different customers [7]. In these situations, a poor product design may create problematic situations as the software becomes difficult to extend, becomes extremely complex and unstable, and most of the company's development time will be spent in bug fixing, maintenance, and testing.

This paper presents our experience of introducing our own Feature Assembly Modelling approach to a software company. We aimed to validate the approach using a real case. Moreover, as we are aware of the fact that classical feature modelling is not yet a practice adopted by companies, we were interested in understanding the reasons for this and check if our Feature Assembly Approach could provide a solution for this. The paper is organized as follows: in section 2, we provide some background on variability modelling and feature modelling. Section 3 presents our Feature Assembly approach. Next, in section 4 we describe the case study and introduce a set of research questions that this case study aimed to answer. Next, in section 5, we present the accomplishment of the case study and section 6 presents the results. Section 7 provides a discussion of the results and section 8 presents the threats to validity. Finally, section 9 provides the conclusions and an overview of our future work

2. BACKGROUND
Companies developing related software products or products having different variants (i.e. a software product line) are faced

with many challenges as the complexity of developing, maintaining, and managing the software increases considerably due to introducing variability. Software variability is defined as "***the ability of a software system or artefact to be efficiently extended, changed, customized or configured for use in a particular context"*** [8]. In order to gain the merits of variability, there is a need for expressing this variability through variability analysis and variability modelling. Feature oriented domain analysis techniques have been commonly used to analyse the variability and commonality of variable products [2]. The resulted variability model is referred to as "feature model" and has been commonly used to model the variability and commonality [1] [2] [9]. Feature models are visual representations (graphs) of the features which the software is composed of in addition to their feature relations, feature dependencies, and their contribution to the variably of the system.

Despite the value of feature modelling it has not found its way to industry. This could be due to the complexity of the modelling technique [10], missing support/training on how to apply feature modelling [3], the different tools with different functionality and usability support [11], or simply because variability was not anticipated from the start but developed overtime as the product matured. A variability modelling technique should be expressive and intuitive enough to capture and represent information about features composing the software product line in addition to how these features contribute to the variability of the software product line. In our previous works, we have identified limitations of mainstream feature modelling techniques [10]. In order to overcome these limitations, we have proposed the Feature Assembly Modelling technique [10].

In addition to modelling variability there is a need for managing this variability. Feature models act as a medium for communicating product capabilities between different stakeholders. As the number of features grows, along with the increasing number of relations between features, finding information manually becomes difficult. Therefore there is a need to allow stakeholders to browse these feature models for information [12]. In addition, there is a need for efficient retrieval of information (e.g. search) from feature models [13] [14].

3. FEATURE ASSEMBLY

The main goal behind ***Feature Assembly*** [10] [15] is to be able to specify variable software products (e.g., a software product line) by combining and reusing (existing) software features. In doing so, reuse is promoted and supported from the initial software conception phase through the complete software development life cycle.

The Feature Assembly approach should help companies define their products better, by first conceiving them in terms of "features". In addition, in the Feature Assembly modelling technique, one needs to distinguish between features that represent variability (i.e. variation points) and those that do not. In order to deal with the size and the complexity of the models, the concept of "***perspective***" was introduced. A perspective allows considering the modelling from one particular point of view at a time, e.g., the system perspective, the task perspective, or the user interface perspective (other perspectives may exist as well). The modelling technique separates features from how they contribute to variability. This makes it possible to reuse features in different variability contexts. Furthermore, this could encourage companies to consider reuse as early as the design phase, not only by allowing to reuse existing features but also by forcing them to design for reuse. Considering reuse at the design level will enable reuse at the implementation level, increasing the overall productivity and reducing cost.

Feature Assembly does not only allow modelling of variability, it also offers continuous management of the information contained in these models. This functionality is provided by our Feature Assembly Framework [15]. The framework is based on a repository for storing features, their relations and dependencies, called the ***Feature Pool***. The Feature Pool Manager allows exploring and searching the pool for reusable features. The Feature Pool may also be used to store complete Feature Assembly Models. In this case, and using the Feature Pool, users can both browse and search the information contained in Feature Assembly models. The results are presented in a visual way, providing better insight in the information and models retrieved. This actually allows Feature Assembly models to act as an interactive documentation source, where users can readily find information. This should allow for improved understanding, management, and reuse of existing software features, as existing features can efficiently be identified as well as their dependencies. Additionally, it could support decision-making by unlocking the knowledge about the software features already developed in the company.

4. CASE STUDY - PLANNING

The case study presented in this paper was part of the living labs initiative of the VariBru[1] project. The idea was to conduct a pilot study in which our Feature Assembly approach was assessed by a company that encounters variability in their products and wants to explicitly represent this variability. The company did not apply the concepts of variability analysis and modelling before. The case study aimed applying the Feature Assembly approach for analysing and modelling variability from the domain analysis and design perspective. The objective was to evaluate the Feature Assembly approach and improve our understanding of the variability modelling needs of companies.

4.1 METHODOLOGY

We introduced the Feature Assembly approach to a small-scale software company, Antidot (located in Brussels), and applied it to a (variable) product of the company. Antidot is working in the domain of web-based IT solutions and services for corporations, companies and associations. To provide these services, they have developed their own Content Management System (CMS) which can be customized (i.e. configured) in different ways to serve the needs of their different customers. Antidot was interested in the approach as they wanted to increase the configurability of their product; they were looking for a way to help them keep track of the different variants of their features (in order to make more accurate customer offers) and the different configurations installed for different customers. Two employees of the company participated in the semi structured and flexible meetings we set up for conducting this case study; the first held the roles of CEO and Senior Project Manager; the second held the role of senior developer and designer (there is no dedicated team for the design). From our research team the authors were the participants, the case study was managed by the first author.

4.2 MOTIVATION

Antidot's CMS product has experienced an increase in features as well as feature variation (i.e. new variants of features were

[1] www.varibru.be

introduced) over time. This has led to a situation in which there was a need to track the available features, the features that hold variations or represent variation points, and the dependencies between these features.

As previous research [4] has showed that small and medium scale software companies are confronted with variability issues, it was interesting to validate the power of Feature Assembly in bringing variability modelling into practice for these companies in order to help solve their variability problems. We have formulated a set of questions that are oriented to measure the relevance of our Feature Assembly approach for the company. These questions can be applied to companies in a similar situation, i.e. that have some form of variability in their products, and which did not yet apply a feature modelling technique.

RQ1. Does the Feature Assembly approach bring variability modelling and management one step closer to industry, i.e. does the company Antidot see added value in adopting this variability modelling approach?

RQ2. Does the company have a problem of concealed information (i.e. information hidden in code, paper documents, or in the heads of the developers)?

RQ3. Can we promote reuse early in the development cycle? Will that make a difference for the development cost?

RQ4. Is our Feature Assembly approach effective and usable in practice?

The case study aimed at finding answers to these questions. Answering these questions should help us gain better understanding of the approach's feasibility as well as its limitations.

5. CASE STUDY - EXECUTION

We had several meetings with members of Antidot. In the first meeting, we introduced the Feature Assembly Modelling approach, i.e. the Feature Assembly Modelling Language and the concept of modelling by reusing features from an existing Feature Pool. We illustrated the concepts using an example. The company also explained how they currently manage their features. In order to help them understand the modelling technique, we made some models for their (existing) CMS and presented them in the next meeting; this quickly initiated a discussion as they saw mistakes in our models (which was not surprising as we didn't know all the details of their software), a sample of their corrections is shown in figure 1. We then asked them to do the modelling process on their own (as homework) and provided them with some basic documentation material about the modelling technique. During our meeting we used a pen-and-paper approach for creating the models (or rather modifying the created models). To help speed up the modelling process we have defined a Visio Stencil that draws the notations of the Feature Assembly Modelling Language, we provided Antidot with this stencil[2]. In the following meetings, we discussed their models, answered their modelling questions, and collected their comments on the ease of use and intuition of the modelling approach. We also introduced the Feature Assembly Framework prototype[3] that we created for

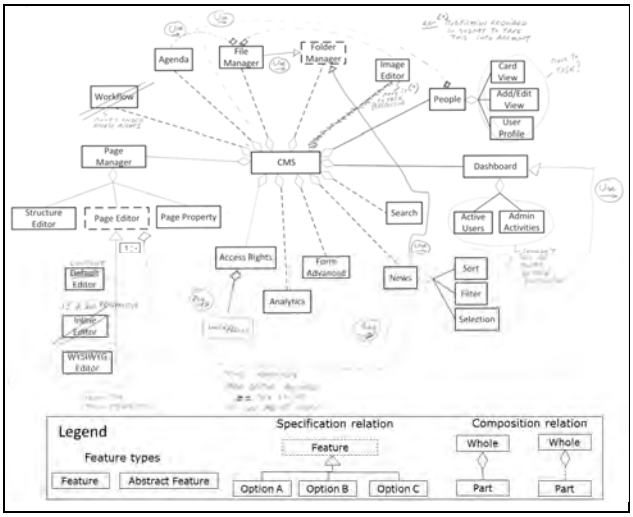


Figure 1. Excerpt showing early modelling of CMS to demonstrate modelling with Feature Assembly

testing the approach and asked them to try it out. In a next meeting, we collected their comments concerning the functionality and the usability of this prototype. By asking them to try out the prototype of the Feature Assembly Framework we wanted to investigate and better understand how companies want to be able to search for information about their designs, and what types of information they consider useful or essential. The available prototype visualizes the Feature Assembly models and allows users to navigate visually through the models in order to find features and information about features.

Figure 2. Screenshot showing how information can be found in Feature Assembly Models using form-based querying - Applied to the models of Antidot.

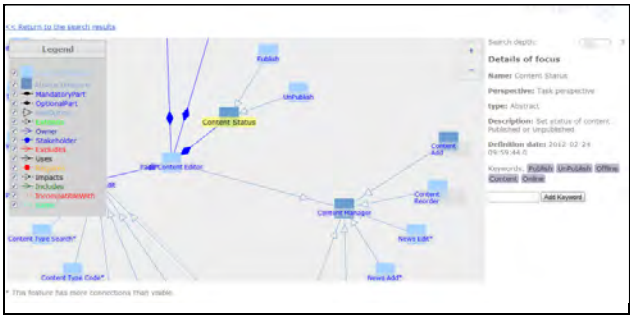


Figure 3. Screenshot showing how Feature Assembly Models are visualized allowing users to browse the information contained in the models - Applied to the models of Antidot.

Furthermore, the prototype allows users to search for information based on predefined metadata such as feature name, feature

[2] Note that at that time, a dedicated tool to create the Feature Assembly models was not yet available.

[3] The Feature Assembly prototype is a web-based application to visualize and explore feature assembly models (also used for Feature Pool contents). The examples can also be found on https://wise.vub.ac.be/feature_assembly/examples.html.

description, feature type, feature definition date (as interval), perspective name, perspective description. Additionally, the features belonging to a specific perspective can be shown. Also a tag cloud is used to enhance the searching via tags that can be assigned to features. The tag cloud also indicates the frequency of tags used to label features.

6. RESULTS

In this section, we elaborate on the results obtained from this evaluation activity. We first present the results obtained for the Feature Assembly modelling technique and next the ones for the Feature Assembly Framework.

6.1 The Feature Assembly Modelling Technique

Concerning the effectiveness and ease of use of the Feature Assembly Modelling technique, Antidot's team reported the following. To analyse and model[4] one major module of the CMS, it took one person two hours and a half. The resulted model contained 28 features and 21 connections between features (14 feature relations and 7 feature dependencies). They found this an acceptable modelling time, although there was some overhead because it was the first time they use a variability modelling technique. In total, three persons were involved in the modelling of the CMS. A small issue was the learning time needed for the notations used, although they appreciated the similarity with the UML notations (as they are using UML for system modelling). Despite this, we have noticed that already after the second modelling meeting, the team was very comfortable with the modelling technique, capable of making decisions concerning feature types and dependencies. Other remarks concerned the expressiveness of the modelling approach. We report their major remarks:

R1. Some features needed many feature dependencies and this was cumbersome to specify.
R2. The distinction between some feature dependencies was not clear and this made it difficult to decide which one to use (e.g., 'uses' versus 'requires')
R3. They were wondering at which level of detail they had to model.
R4. It was not clear how they could specify external features/components.
R5. Sometimes it was difficult to decide which perspective to use for modelling certain features.
R6. It was not clear if and how they could model different versions of the same feature.

Some of these remarks are due to the lack of experience with the Feature Assembly modelling technique and the lack of good documentation for the method (e.g., an elaborated user guide), such as remarks R2, R3, and R5. Also remark R5 was given because they assumed that a feature could only belong to one perspective, which in not the case. Remark R1 triggers some important questions: Is it necessary to always model all dependencies? Is a high coupling not an indication of some bad design decision? And if all dependencies are really justified, can we not find an easy way to specify them, e.g., by introducing some abstraction mechanisms to reduce the number of links that need to be specified? There is no unique recipe for solving this issue, on the other hand knowing this kind of information at a design time allows considering design patterns to eliminate such coupling between features in code [16]. Remarks R4 and R6 were indeed very valuable, as the method currently doesn't provide support for this. Currently, the Feature Assembly Modelling technique treats all features (external and non-external) similarly. Also it does not support versioning of features. These issues should be considered in future work.

Furthermore, the case study has revealed/confirmed the following merits of adopting such a variability modelling technique:

1. Feature Assembly let them reconsider their "***features***" in order to increase the modularity of the software. Using the Feature Assembly Modelling technique, dependencies between features became more visible and they can use this to improve the design for achieving a lower degree of coupling between modules/components at the code level. In their own words they mentioned "***We found that our software is not as modular as we thought it was, therefore we are now rethinking our feature dependencies to make our components more modular to increase the reusability in our system***"
2. Explicitly modelling variabilities and commonalities triggered new potential variation points. As a result, more variability will be planned in the next version of the product.
3. Documenting and understanding the feature dependencies helps them in better defining their test scenarios, as the feature dependencies are reflected as module dependencies in the code. In their own words they mentioned ***"understanding the feature dependencies already gives important information for building testing scenarios"***
4. Feature Assembly models help them better identifying the impact of change in features.
5. The system perspective provides a better view on the important features of their product, providing a different level of abstraction and understanding of their system.

The company also reported that Feature Assembly models will help them with understanding and managing the evolving variability of their product over time.

6.2 The Feature Assembly Framework

The evaluation exercise of the Feature Assembly Framework prototype performed by Antidot confirmed our hypothesis of the importance of the ability to unlock the knowledge contained in design models (i.e. Feature Assembly models). The team of Antidot confirmed that providing a visual navigation mechanism for inspecting the models was indeed useful. Furthermore, allowing users to visually interact with the Feature Assembly models is useful when tracing a certain feature for its relations or dependencies. In their case, they had some features that represented the backbone of their system and which they found very useful to inspect using the prototype. This functionality is particularly important when more than one person is involved in the modelling (in their case three persons were involved). Also, they reported that being able to control the depth of display for a model during visualization is indeed useful for providing different levels of detail.

Furthermore, Antidot recommended adding some important meta-data to the information stored. For example, they recommended adding a description for each perspective and a

[4] This was their first use of a variability modeling technique and therefore we cannot compare the obtained results to a previous experience.

definition date for the features. A definition date could also help them overcome the issue of lack of versioning support for the features mentioned in the previous section (we actually updated our prototype to include this and let them test it again).

Among the discussions we had was the discussion of the applicability of the Feature Assembly Framework [15] for reusing already existing features in the design of new products. Being a small size company their reuse schema was based on "opportunistic reuse" [17], i.e. the reuse of components and code at the implementation level. Reuse at a design level was not given much attention. Introducing them to the concept of "design with reuse" has actually led them to reconsider the independency of their features to enable more reuse opportunities. It was agreed that the power of considering reuse at the design level ("design with reuse") is that it promotes component reuse rather than code reuse and as such also enables "design for reuse". To achieve "design for reuse" the following guidelines were identified:

1. Identify which features are candidate standalone (i.e. consolidated and independent) features.
2. Analyse which of the feature dependencies are essential and should be enforced for these features when reused.
3. Improve the models such that the feature dependencies between standalone features are minimized.
4. Use the meta-data to describe these features, in order to be able to easy retrieve them later on, in particular by the use of tags. Restricting the tags to a specific set (e.g. using a predefined set of keywords) was not recommended, but rather a growing pool of tags was advised.

To enable "design with reuse" the following requirements were identified:

1. A good search mechanism is needed to identify already existing and reusable features.
2. The need to invest time in carefully modelling (existing) software features.

7. DISCUSSION

We can conclude that the work done during this evaluation, as well as the discussions held, confirmed the value of the presented approach; it also revealed interesting future work. The presented case study clearly answered our research questions stated earlier, the company clearly stated that they see added value in applying feature analysis and modelling to their product(s), this answers our first research question (RQ1).

The Feature Assembly Framework was also appreciated for providing an interactive medium for finding information about features in the Feature Assembly Models. For this to payoff, the company has to enforce a strict policy for adding meta information (e.g., feature description, feature keywords, stakeholders involved, customers who have this feature, etc.) and therefore making it available for later. From the discussions we had it was also clear that not all stakeholders need the same detailed level of information. For example, developers are interested in all levels of details for the modules they are responsible for, but for other modules they are only interested in the feature dependencies. It was clear that even this small company does have a need to unlock information implicitly available inside the company (RQ2).

The case study has also confronted us with the gap between industry and research in the domain of software variability. We started, as many other researchers, with an approach to be used when developing a new product line, however it turned out that in practice, small and medium scale companies acquire variability over time in their products and need mechanisms to deal with the variability of existing products or turn existing products into product lines. Although our approach was originally not designed for this purpose, it could however also be applied usefully in this context. Nevertheless, the experience indicated that (at least a part of the) research should be more driven by the challenges faced by the industry, and researchers should not try to push solutions for which there is no need in practice. Furthermore, the case study has confirmed the need to evaluate research prototypes in collaboration with industry to validate their effectiveness and to reveal additional needs.

The presented case study only provided a partial answer to our third research question considering feature reuse (RQ3). Feature Assembly modelling allows making more modular designs. Furthermore, the Feature Assembly Framework helps efficiently retrieve features for reuse. Therefore we may say that it increases the chances of successful reuse inside the company, therefore increasing the chances of reducing development cost. However, actual reuse can only be achieved while developing a new product. This has not been performed during the case study. Therefore, it was not possible to answer RQ3 with complete certainty.

The time taken by Antidot to learn to use the Feature Assembly Modelling technique was quite impressive. The company was also very positive on the ease of use and intuition of the modelling concepts and notations. They reported no problems with the understandability of the modelling semantics. The only negative issue mentioned was that sometimes it was not very obvious for them which feature dependency (we provide eight different dependencies) fits best to describe a certain situation. Also the notations used for the dependencies were sometimes difficult to remember. However, they found each one of the proposed dependencies essential. Therefore, we believe that this will improve with more practice of the technique. Although, we did not measure the effectiveness and usability in a quantitative way, the answer on our fourth research question (RQ4) is definitely positive.

Furthermore, research on software product lines focuses mostly on the architecture, implementation, and configuration levels. It is our opinion, and this is confirmed by our validation, that modelling issues concerning variable software are as important. In addition, there is a need for extending the research on variability with Information Management aspects to deal with growing amount of information needed for and about variable software. No matter how large or small the company teams are, tools that allow flexible information sharing are required.

8. THREATS TO VALIDITY

As we only validated the approach with one company, it may be possible that experiences in other companies could be different. However, the company was unknown to the researchers before the case study was started and the company also didn't have any reason to favour the approach or the researchers. Therefore, we can state that the results obtained are rather objective.

The fact that the company is a small-scale company may have had an impact on the results.

As already mentioned, the company has not been using the concepts of variability modelling before, neither the concept of "feature" to describe their product capabilities. This may have affected the results in two different ways. First, introducing a new modelling technique may have introduced some learning time

(which was indeed the case). Secondly, because Antidot has not used a variability modelling technique before they cannot compare the ease of use and expressiveness of Features Assembly to other feature modelling techniques.

The case study was done in a rather informal way, i.e. using meetings and discussion. We believe that this is justified for a first (pilot) validation case study, as the first purpose was to obtain as much spontaneous feedback as possible. In later case studies and experiments, a more rigorous approach will be used.

9. CONCLUSIONS AND FUTURE WORK

In this paper, we presented the results of an evaluation of our Feature Assembly approach performed by a small-scale software company. This evaluation was fruitful in many ways. Firstly, it gave us some insight on how a company works and on their challenges concerning managing the continuous growth and variation of their products. Secondly, the validation has clearly shown the importance of modelling software in terms of the composing features in order to better understand and identify the sources of complexity in the product. This is particularly important in products that contain variability or that acquire variability over time. We started this case study with some research questions in mind to help us evaluate our Feature Assembly Approach. Our questions have been answered, moreover, new issues were raised concerning the company's needs when modelling and managing the variability in their products (e.g., the need for explicit feature versioning). Most of these issues are worth further investigation.

For our future work, we seek applying the Feature Assembly approach to more industrial cases; this will certainly help improving the technique. It will also help us understand what meta-data is useful for unlocking information concerning features. Also the presented case study has pointed out the importance of tool support [11] that may go beyond simple proof of concept tools. We plan to develop a Feature Assembly modelling editor to help companies rapidly create their feature assembly models. Furthermore, the issue of feature versioning will be considered in future work.

10. ACKNOWLEDGMENTS

This research is sponsored by Innoviris, the Brussels institute for research and innovation (www.innoviris.be) through the VariBru project (www.varibru.be).

The authors like to thank Sebastien Le Grand and Frederic Arijs from Antidot for their valuable contribution during this case study. The authors also like to thank Tom Puttemans for his prototype implementation of the Feature Assembly Visualization tool.

11. REFERENCES

[1] K. Kang, S. Cohen, J. Hess, W. Novak, A. Peterson, "Feature-oriented domain analysis (FODA) feasibility study", Technical Report CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie-Mellon University, 1990

[2] K.C. Kang, J. Lee, P. Donohoe, "Feature-Oriented Product Line Engineering", IEEE Software, vol. 19, no. 4, pp. 58-65, 2002

[3] A. Hubaux, , A. Classen, , M. Mendonca, , P. Heymans, A Preliminary Review on the Application of Feature Diagrams in Practice, In proccedings of VaMoS 2010 (VaMoS'10), Linz, Austria, January 27-29, University of Duisburg-Essen, pp. 53-59, 2010

[4] W. Codenie, N. González-Deleito, J. Deleu, V. Blagojevic, P. Kuvaja, J. Similä: A Model for Trading off Flexibility and Variability in Software Intensive Product Development, In proccedings of VaMoS 2009, ICB-Research Report, Universität Duisburg-Essen, pp. 61-70, 2009

[5] R. Van Ommering, J. Bosch, Widening the scope of software product lines — from variation to composition, In Proceedings of the Second International Conference on Software Product Lines (SPLC 2), Gary J. Chastek (Ed.). Springer-Verlag, London, UK, pp. 328-347, 2002

[6] B. Curtis, H. Krasner, N. Iscoe, A Field Study of the Software Design Process for Large Systems, Communications of the ACM, Volume 31 Issue 11, Nov. pp. 1268-1287, 1988

[7] D. M. Berry, K. Czarnecki, M. Antkiewicz, M. AbdelRazik, Requirements Determination is Unstoppable: An Experience Report, 18th IEEE International Conference on Requirements Engineering, pp. 311-316, 2010

[8] M. Svahnberg, J. van Gurp, J. Bosch, A Taxonomy of Variability Realization Techniques. in Software Practice & Experience, 35(8). pp.705-754, (2005).

[9] M. L. Griss, J. Favaro, M. d'Alessandro, Integrating Feature Modelling with the RSEB, Proc. Fifth International Conference on Software Reuse, pp. 76-85, Victoria, BC, Canada , 1998.

[10] L. Abo Zaid, F. Kleinermann, O. De Troyer: Feature Assembly: A New Feature Modelling Technique, In : 29th International Conference on Conceptual Modelling, Lecture Notes in Computer Science, 2010, Volume 6412/2010, Springer-Verlag, pp. 233-246

[11] M. El Dammagh, O. De Troyer, Feature Modelling Tools: Evaluation and Lessons Learned, ER Workshops 2011 pp. 120-129, 2011

[12] D. Nestor, S. Thiel, G. Botterweck, C. Cawley, P. Healy: Applying visualisation techniques in software product lines. SOFTVIS 2008: 175-184

[13] H. Wang, Y. Li, J. Sun, H. Zhang, J. Pan, A semantic web approach to feature Modelling and verification. In: Proceedings of Workshop on Semantic Web Enabled Software Engineering (SWESE'05), 2005

[14] L. Abo Zaid, F. Kleinermann, O. De Troyer, Applying Semantic Web Technology to Feature Modelling. In: The 24th Annual ACM Symposium on Applied Computing, The Semantic Web and Applications (SWA) Track march 2009.

[15] L.Zaid, F. Kleinermann, O. De Troyer, Feature Assembly Framework: towards scalable and reusable feature models, In Proceedings of the 5th Workshop on Variability Modelling of Software-Intensive Systems (VaMoS '11). ACM, New York, NY, USA, pp. 1-9, 2011

[16] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object-oriented Software, Addison-Wesley, 1995.

[17] R. van Ommering, Software Reuse in Product Populations, IEEE Trans. Software Eng., vol. 31, no. 7, 2005, pp. 537–550.

The Role of Domain-Knowledge in Interpreting Activity Diagrams – An Experiment

Ana M. Fernández-Sáez, Peter Hendriks, Werner Heijstek and Michel R.V. Chaudron
Leiden Institute of Advanced Computer Science, Leiden University, Niels Bohrweg 1, 2333 CA Leiden
{fernande,heijstek,chaudron}@liacs.nl, httpeter@gmail.com

ABSTRACT

The UML has been designed as a tool to support software development in general. UML is, however, not always intuitive to use in any given domain. We are unsure how domain knowledge influences the interpretation of generic UML diagrams. A commonly used diagram is the activity diagram. This study outlines an experiment in which subjects with medical knowledge, computer scientists, and a control group of respondents with other backgrounds, are compared. To this end, we used process models from the medical domain. Our conclusions include that medical activity diagrams are marginally better interpreted by medically trained staff than by computer scientist. This is remarkable when compared to the low score of subjects without a medical- or computer science background implying that knowledge of a modeling language can be more important than domain knowledge.

Categories and Subject Descriptors

D.3.2 [**Programming Languages**]: Language Classification – *design languages (UML)*

General Terms

Documentation, Design, Experimentation, Human Factors, Languages.

Keywords

UML, activity diagrams, domain knowledge, empirical study, experiment

Empirical Design Science for Artefact-based Requirements Engineering Improvement

Daniel Méndez Fernández
Technische Universität München, Germany
http://www4.in.tum.de/~mendezfe

Roel Wieringa
University of Twente, Netherlands
http://www.cs.utwente.de/~roelw

ABSTRACT

Requirements Engineering (RE) improvement considers the improvement of methods and techniques for specifying software systems in response to business goals and needs. Current RE improvement approaches assume a rigid one-size-fits-all external process standard and compare the RE process in a company with this standard. This is not as effective as it could be, because different companies aim at different goals in their RE. They are confronted with different problems depending on their domain or the organisational culture, leading to highly volatile RE processes. In order to understand RE improvement in practice, the first author conducted a series of empirical studies of RE processes in companies and initiated RE improvements based on individual improvement needs in these companies. He created a framework for an inductive (problem-driven) and artefact oriented approach to RE improvement, where the focus lied on the investigation and improvement of RE models and other documents rather than on potentially differing and complex processes for creating the artefacts. This framework is strong on empirical problem analysis but still neglects the implementation of the RE improvement proposal. In this paper, we extend the framework with a framework for design science created by the second author, and sketch how this the outcome is validated by additional empirical studies and action research of RE improvement.

Keywords

Artefact-based Requirements Engineering, Empirical Design Science, Software Process Improvement

Modeling and Enterprises – the past, the present and the future

Vinay Kulkarni
Tata Research Development and Design Centre
54-B, Industrial Estate, Hadapsar,
Pune, India 411013
+91 20 66086333
vinay.vkulkarni@tcs.com

ABSTRACT

Industry has been practicing model-driven development in various flavours. In general it can be said that modelling and use of models have delivered on the promises of platform independence, enhanced productivity, and delivery certainty as regards development of software-intensive systems. Globalization market forces, increased regulatory compliance, ever-increasing penetration of internet, and rapid advance of technology are some of the key drivers leading to increased business dynamics. Increased number of factors impacting the decision and interdependency there between is leading to increased complexity in making business decisions. Also, enterprise software systems need to commensurately change to quickly support the business decisions. The paper presents synthesis of our experience over a decade and half in developing model-driven development technology and using it to deliver 70+ business-critical software systems worldwide.

Categories and Subject Descriptors

D.2 [**Software Engineering**]: Design

General Terms

Management, Design, Languages, Theory

Keywords

Modelling, Meta modelling, Model-driven development, Enterprise systems, Adaptation, Analysis, Simulation