

THESIS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

Empowering Empirical Research in Software Design:  
Construction and Studies on a Large-Scale Corpus of  
UML Models

TRUONG HO-QUANG



Division of Software Engineering  
Department of Computer Science & Engineering  
Chalmers University of Technology and University of Gothenburg  
Gothenburg, Sweden, 2019

**Empowering Empirical Research in Software Design:  
Construction and Studies on a Large-Scale Corpus of UML Models**

TRUONG HO-QUANG

Copyright ©2019 Truong Ho-Quang  
except where otherwise stated.  
All rights reserved.

ISBN 978-91-7833-608-1 (PRINT)  
ISBN 978-91-7833-609-8 (PDF)  
ISSN 0346-718X  
The thesis is available in full text online  
<http://hdl.handle.net/2077/61704>

Technical Report No 173D  
Department of Computer Science & Engineering  
Division of Software Engineering  
Chalmers University of Technology and University of Gothenburg  
Gothenburg, Sweden

Cover illustration: Steps of printing a Đông Hồ painting (called “Gà Dạ Xương”)  
Photos taken by Phùng Hồng Kôn.

This thesis has been prepared using L<sup>A</sup>T<sub>E</sub>X.  
Printed by Chalmers Reproservice,  
Gothenburg, Sweden 2019.

*“Extraordinary Claims Require Extraordinary Evidence.”*  
*- Carl, Sagan.*





# Abstract

**Context:** In modern software development, software modeling is considered to be an essential part of the software architecture and design activities. The Unified Modeling Language (UML) has become the de facto standard for software modeling in industry. Surprisingly, there are only a few empirical studies on the practices and impacts of UML modeling in software development. This is mainly due to the lack of empirical data on real-life software systems that use UML modeling.

**Objective:** This PhD thesis contributes to this matter by describing a method to build and curate a big corpus of open-source-software (OSS) projects that contain UML models. Subsequently, this thesis offers observations on the practices and impacts of using UML modeling in these OSS projects.

**Method:** We combine techniques from repository mining and image classification in order to successfully identify more than 24.000 open source projects on GitHub that together contain more than 93.000 UML models. Machine learning techniques are also used to enrich the corpus with annotations. Finally, various empirical studies, including a case study, a user study, a large-scale survey and an experiment, have been carried out across this set of projects.

**Result:** The results show that UML is generally perceived to be helpful to new contributors. The most important motivation for using UML seems to be to facilitate collaboration. In particular, teams use UML during communication and planning of joint implementation efforts. Our study also shows that the use of UML modeling has a positive impact on software quality, i.e. it correlates with lower defect proneness. Further, we find out that visualisation of design concepts, such as class role-stereotypes, helps developers to perform better in software comprehension tasks.

## Keywords

Software Modeling, Software Design, Empirical Research, UML, Modeling Practice, Impacts of Modeling, Open Source System, Mining Software Repository, Data Mining, Data Curation, GitHub.



# Acknowledgment

To accomplish this 5-year Ph.D project, I have received lots of encouragement from colleagues, friends and my family. I would take this opportunity to thank:

My main supervisor Prof. Michel R. V. Chaudron, for the continuous support to my Ph.D study, for your patience, motivation, and immense knowledge. Thanks for being not only a great academic adviser but also an intimate friend.

My co-supervisor Regina Hebig, for voluntarily supporting me and providing me with tips and comments whenever needed.

My former co-supervisor Patrizio Pelliccione, for offering interesting discussions and constructive comments in the first two years of my Ph.D.

My examiner Prof. Ivica Crnkovic, for your encouragement, for hard questions which incite me to widen my research from various perspectives.

Directors of Ph.D study Jan Jonsson and Agneta Nilsson, for always giving constructive recommendations and reminding me to add buffers to my often-ambitious-research plans.

To all of my colleagues at the Software Engineering Division, including of course administrative staff: Thank you all for creating such a friendly and productive work environment. I particularly thank Rodi Jolak for sharing the office with me, for interesting discussions and lots of push-up exercises at work. Hugo, Federico, Grischa, Salome: for being good and helpful neighbours.

My research would have not been possible without support from my collaborators. I would express my deeply thanks to all 24 people who co-authored papers with me. In particular, I thank Gregorio Robles and Miguel Ángel Fernández for being part of the best team that I've ever had. Many thanks to Arif Nurwidiantoro, Alexandre Bergel, Adithya Raghuraman, Alexander Serebrenik, Bogdan Vasilescu for our regular discussions despite the major time differences. To Dave, Hafeez and Bilal: I am thankful to be part of our team. Thanks to your help, I came to the Ph.D life less nervous.

To all friends, near and far away, especially the Vietnamese “gangs” in Gothenburg: I sincerely thank you for sharing lots of joys during the years and recharging my battery when it was getting low.

And to the most important people in my life: My lovely wife and daughter - thank you for stepping into my life and making it full of joy and happiness everyday. I also owe much of my success to my parents and grand-fathers, who supported me spiritually throughout writing this thesis and my life in general. Last but not least, I would give many thanks to my brothers and their families for the great suggestions and motivation during my Ph.D life.

*Truong Ho-Quang*  
Gothenburg, 2019.



# List of Publications

## Included publications

This thesis is based on the following eight (8) papers:

- [A] **T. Ho-Quang**, M.R.V. Chaudron, I. Samúelsson, J. Hjaltason, B. Karasneh, H. Osman “Automatic Classification of UML Class Diagrams from Images”.  
*Published in the Proceedings of the 21st Asia-Pacific Software Engineering Conference (APSEC 2014), Jeju, Korea, December 1 - December 4, 2014.*
- [B] R. Hebig, **T. Ho-Quang**, M.R.V. Chaudron, G. Robles, F. Miguel Angel “The Quest for Open Source Projects that Use UML: Mining GitHub”.  
*Published in the Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems (MODELS 2016), Saint-Malo, France, October 2 - October 7, 2016.*
- [C] **T. Ho-Quang**, R. Hebig, G. Robles, M.R.V. Chaudron, F. Miguel Angel “Practices and Perceptions of UML Use in Open Source Projects”.  
*Published in the Proceedings of the 39th International Conference on Software Engineering - Software Engineering in Practice Track (ICSE SEIP 2017), Buenos Aires, Argentina, May 20 - May 28, 2017.*
- [D] M.H. Osman, **T. Ho-Quang**, M.R.V. Chaudron, “An Automated Approach for Classifying Reverse-engineered and Forward-engineered UML Class Diagrams”.  
*Published in the Proceedings of the 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA) (pp. 396-399), Prague, Czech Republic, August 29-31, 2018.*
- [E] **T. Ho-Quang**, M.R.V. Chaudron, R. Hebig, G. Robles “Challenges and Directions for a Community Infrastructure for Big Data-driven Research in Software Architecture”.  
*Accepted as a chapter in the book “Model Management and Analytics for Large Scale Systems”, To be published by Elsevier (Expected release date: November 1, 2019).*
- [F] A. Raghuraman, **T. Ho-Quang**, M.R.V. Chaudron, A. Serebrenik, B. Vasilescu “Does UML Modeling Associate with Higher Software Quality in Open-Source Software?”.  
*Accepted at the 16th International Conference on Mining Software Repositories (MSR2019), Montréal, Canada, May 26 - May 27, 2019.*

- [G] **T. Ho-Quang**, A. Nurwidyantoro, M.R.V. Chaudron “Using Machine Learning for Automated Classification of Class Responsibility Stereotypes in Software Design”.  
*Under Submission.*
- [H] **T. Ho-Quang**, A. Bergel, A. Nurwidyantoro, M.R.V. Chaudron “Interactive Role Stereotype-Based Visualization To Comprehend Software Architecture”.  
*Under submission.*

## Other publications

The following publications were published during my PhD studies, or are currently in submission. However, they are not appended to this thesis, due to contents overlapping that of appended publications or contents not related to the thesis.

- [a] H. Osman, M.R.V. Chaudron, P. van der Putten, **T. Ho-Quang** “Condensing Reverse Engineered Class Diagrams Through Class Name Based Abstraction”  
*4th World Congress on Information and Communication Technologies (WICT’14), Malacca, Malaysia, December 8 - December 10, 2014.*
- [b] D.R. Stikkolorum, **T. Ho-Quang**, M.R.V. Chaudron “Revealing Students’ UML Class Diagram Modelling Strategies with WebUML and LogViz”  
*41st Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Funchal, Madeira, Portugal, August 26 - August 28, 2015.*
- [c] D.R. Stikkolorum, **T. Ho-Quang**, B. Karasneh, M.R.V. Chaudron “Uncovering Students’ Common Difficulties and Strategies During a Class Diagram Design Process: an Online Experiment”  
*Educators Symposium at ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (EduSymp@MODELS 2015), Ottawa, Canada, September 29, 2015.*
- [d] R. Hebig, **T. Ho-Quang**, M.R.V. Chaudron, G. Robles, F. Miguel Angel “The Quest for UML in Open Source Projects Initial findings from GitHub”  
*Published in the Proceedings of the Doctoral Consortium at the 12th International Conference on Open Source Systems (OSS 2016), Göteborg, Sweden, May 30, 2016*
- [e] R. Jolak, E. Umuhoza, **T. Ho-Quang**, M.R.V. Chaudron, M.Brambilla “Dissecting design effort and drawing effort in UML modeling”  
*Published in the Proceedings of the 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Vienna, Austria, 2017.*
- [f] G. Robles, **T. Ho-Quang**, R. Hebig, M.R.V. Chaudron, F. Miguel Angel “An extensive collection of UML files in GitHub”  
*Published in the Proceedings of the 14th International Conference on Mining Software Repositories (MSR 2017).*
- [g] R. Jolak, **T. Ho-Quang**, M.R.V. Chaudron, R.R.H. Schiffelers “Model-Based Software Engineering: A Multiple-Case Study on Challenges and Development Efforts”  
*Published in the Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, 2018.*
- [h] Y. El Ahmar, X. Le Pallec, S. Gérard, **T. Ho-Quang** “Visual Variables in UML: a First Empirical Assessment”  
*Workshop in Human Factors in Modeling (HuFaMo 2018), MODELS 2018, Austin, US, 2018.*

- [i] M.R.V Chaudron, A. Fernandes-Saez, R. Hebig, **T. Ho-Quang**, R. Jolak “Diversity in UML Modeling Explained: Observations, Classifications and Theorizations”  
*In International Conference on Current Trends in Theory and Practice of Informatics (SOFSEM 2018), Krems, Austria, January 29 - February 2, 2018.*
  
- [j] A. Nurwidyantoro, **T. Ho-Quang**, M.R.V. Chaudron “Automated Classification of Class Role-Stereotypes via Machine Learning”  
*Published in the Proceedings of the Evaluation and Assessment in Software Engineering conference (EASE 2019), Copenhagen, Denmark, 14 April - 17 April, 2019.*
  
- [k] **T. Ho-Quang**, M.R.V. Chaudron, G. Robles, G.B. Herwanto “Building an Infrastructure for Empirical Software Architecture Studies: Challenges and Directions”  
*Accepted at ICSE workshop on Establishing a Community-Wide Infrastructure for Architecture-Based Software Engineering (ECASE 2019)*



## Research Contribution

My contributions to Paper A are study design, data analysis and the majority of paper writing. The tool that was used for extracting image-processing features in the paper was implemented by I. Samelsson and J. Hjaltason. The remaining authors contributed with reviews and improvement suggestions.

Studies reported in Paper B and C were conducted in collaboration with the Grupo de Sistemas y Comunicaciones (GSyC) <sup>1</sup> at the Universidad Rey Juan Carlos (URJC)<sup>2</sup>. In the two papers, the major effort in classifying UML images and validating the classification results was made by me.

In Paper B, I participated and contributed in designing the study, formulating research questions, analyzing data and discussing results. I wrote a majority of the publication regarding Introduction, Methodology and Threats to validity.

In Paper C, I took the leading role in study design, data collection and data analysis. My main effort in this work consists of identifying UML images, executing the survey and analyzing the data. In term of paper writing, I wrote the majority of the sections Research Questions, Methodology, Results/Findings and Conclusion.

In Paper D, my main contribution lies in: i) coordinating the process of building the ground truth; ii) implementing a tool for extracting machine learning features from .xmi files; and iii) interpreting the classification results. I also contributed by writing a major part of Methodology and Discussion sections. As an extension to the paper, I applied the trained classification model to classify the entire Lindholmen dataset.

In Paper E, I took the leading role in proposing CoSARI - a reference architecture for a community-based infrastructure for big-data driven research in software architecture. I also actively participated in: i) reporting experiences on building and sharing Lindholmen dataset; and ii) discussing challenges to empirical research in the field and requirements for such the infrastructure.

In Paper F, I contributed mostly in filtering/validating “UML projects” from the Lindholmen dataset and interpreting the result of data analysis. I also participated in writing up the Methodology and Threats to Validity sections.

Paper G is an extended version of Paper [j] (in the list of Other publications). In both papers, I took the leading role in designing the study, building the ground truth and analysing the classification results. As part of its data analysis, I developed a visualisation tool to explore the collaboration patterns between role-stereotypes. I also contributed by writing a major part of the paper, including Introduction, Methodology, Discussion of new applications of role-stereotypes.

The study described in Paper H was done in collaboration with the Intelligent Software Construction laboratory (ISCLab <sup>3</sup>) of the University of Chile. In the paper, I worked closely with the second authors to create RoleViz - a tool that visualises class role stereotypes. I took a leading role in designing, conducting the evaluation user-study and analysing the obtained data. In this paper, I wrote a major part of Methodology, Data Collection & Analysis, Discussion and participated in writing/reviewing the other parts.

---

<sup>1</sup>Home page of GSyC - <https://gsyc.urjc.es/>

<sup>2</sup>Home page of the Universidad Rey Juan Carlos (Madrid, Spain) - <http://www.urjc.es/>

<sup>3</sup>ISCLab's homepage: <https://isclab.dcc.uchile.cl/>



# Contents

<b>Abstract</b>	<b>v</b>
<b>Acknowledgement</b>	<b>vii</b>
<b>List of Publications</b>	<b>ix</b>
<b>Personal Contribution</b>	<b>xiii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Research Focus	2
1.1.1 Goals of the Ph.D Study	2
1.1.2 Scope and Expected Outcomes of the Ph.D Study	3
1.1.3 Research questions of the Ph.D thesis	7
1.2 Background	7
1.2.1 The Unified Modeling Language (UML)	7
1.2.2 Existing Corpora of Software Modeling Artifacts	8
1.2.3 UML use and impacts of using UML in software engineering projects	11
1.3 Methodology	13
1.3.1 Constructive research	14
1.3.2 Empirical research	15
1.4 Contributions	16
1.4.1 Paper A: Automatic classification of UML class diagrams from images	17
1.4.2 Paper B: The quest for open source projects that use UML: mining GitHub	18
1.4.3 Paper C: Practices and perceptions of UML use in open source projects	20
1.4.4 Paper D: An Automated Approach for Classifying Reverse-engineered and Forward-engineered UML Class Diagrams	21
1.4.5 Paper E: Challenges and Directions for a Community Infrastructure for Big Data-driven Research in Software Architecture	23
1.4.6 Paper F: Does UML Modeling Associate with Higher Software Quality in Open-Source Software?	24
1.4.7 Paper G: Using Machine Learning for Automated Classification of Class Responsibility Stereotypes in Software Design	26

1.4.8	Paper H: Interactive Role Stereotype-Based Visualization To Comprehend Software Architecture . . . . .	27
1.5	Summary of Research Findings . . . . .	28
1.5.1	Answer to RQ1. How to build a large corpus of models? . . . . .	28
1.5.2	Answer to RQ2. How can we share and promote use of the corpus of UML models? . . . . .	31
1.5.3	Answer to RQ3. What are purposes of using UML in OSS projects? . . . . .	31
1.5.4	Answer to RQ4. How is UML used in OSS projects? . . . . .	32
1.5.5	Answer to RQ5. What are practices for using UML modeling in software development? . . . . .	32
1.5.6	Answer to RQ6. What are perceived impacts of using UML in OSS projects? . . . . .	33
1.5.7	Answer to RQ7. Does the use of UML modeling correlate with lower defect proneness? . . . . .	33
1.5.8	Answer to RQ8. Does using class role stereotypes correlate with better understanding of designs of software system? . . . . .	34
1.6	Discussion . . . . .	34
1.6.1	Software Modeling and Design Practices in Industry . . . . .	34
1.6.2	Alternatives of the Machine Learning Approach . . . . .	38
1.7	Threats to validity . . . . .	39
1.8	Future Work . . . . .	40
1.8.1	Extending the Lindholmen data set . . . . .	40
1.8.2	Extending the understanding about UML use and impact: enablers, inhibitors and context . . . . .	42
1.8.3	Building guidelines for UML use . . . . .	46
1.8.4	Other directions . . . . .	47
<b>2</b>	<b>Paper A</b> . . . . .	<b>49</b>
2.1	Introduction . . . . .	50
2.2	Related Work . . . . .	51
2.2.1	Image classification . . . . .	51
2.2.2	Diagram feature extraction . . . . .	52
2.3	Research Questions . . . . .	52
2.4	Approach . . . . .	52
2.4.1	Overall framework . . . . .	52
2.4.2	Image processing . . . . .	53
2.4.3	Feature extraction . . . . .	54
2.4.4	UML CD classification . . . . .	57
2.4.5	Analyse Result . . . . .	58
2.5	Experiment Description . . . . .	58
2.5.1	Dataset . . . . .	58
2.5.2	Evaluation measures . . . . .	58
2.5.3	Experiment settings . . . . .	59
2.6	Analysis Of Results . . . . .	59
2.6.1	RQ1: Influence of features . . . . .	59
2.6.2	RQ2: Classification algorithms performance . . . . .	60
2.6.3	RQ3: Set of features Performance . . . . .	61

2.7	Discussion . . . . .	62
2.7.1	Image Processing Time . . . . .	62
2.7.2	Image Processing Features Performance . . . . .	63
2.7.3	Classification Algorithms . . . . .	64
2.7.4	Threats to validity . . . . .	64
2.8	Conclusions and Future Work . . . . .	64
<b>3</b>	<b>Paper B</b>	<b>67</b>
3.1	Introduction . . . . .	68
3.2	Research questions . . . . .	69
3.3	Related research . . . . .	70
3.3.1	Use of UML in FOSS . . . . .	70
3.3.2	Mining . . . . .	71
3.4	Methodology . . . . .	71
3.4.1	Occurrence of UML . . . . .	72
3.4.2	Data Collection . . . . .	73
3.4.3	UML filters . . . . .	74
3.4.4	Metadata Extraction and Querying . . . . .	76
3.5	Results . . . . .	76
3.5.1	RQ1: UML in GitHub projects . . . . .	77
3.5.2	RQ2: Versions of UML models . . . . .	77
3.5.3	RQ3: Time of UML model introduction . . . . .	78
3.5.4	RQ4: Time span of active UML . . . . .	80
3.5.5	RQ5: Duplicates . . . . .	82
3.6	Discussion . . . . .	83
3.7	Threats to validity . . . . .	86
3.7.1	Threats to construct validity . . . . .	87
3.7.2	Threats to external validity . . . . .	87
3.7.3	Threats to conclusion validity . . . . .	88
3.8	Conclusions . . . . .	88
<b>4</b>	<b>Paper C</b>	<b>91</b>
4.1	Introduction . . . . .	92
4.2	Research Question . . . . .	93
4.3	Related work . . . . .	94
4.3.1	Modeling in Industry . . . . .	94
4.3.2	Modeling in Open Source Software . . . . .	94
4.4	Research Methodology . . . . .	95
4.4.1	Data Collection . . . . .	95
4.4.2	Filtering the obtained projects and contributors . . . . .	96
4.4.3	Conducting the survey . . . . .	96
4.4.4	Data Analysis . . . . .	98
4.5	Results/Findings . . . . .	99
4.5.1	Respondent Demographics . . . . .	99
4.5.2	Why is UML used? . . . . .	100
4.5.3	Is UML part of the interaction of contributors? . . . . .	101
4.5.4	What is the impact/benefit of UML? . . . . .	104
4.6	Discussions . . . . .	106
4.6.1	Comparison to Insights to Related Works . . . . .	107

4.6.2	Implications . . . . .	107
4.6.3	Threats to Validity . . . . .	108
4.7	Conclusion and Future work . . . . .	109
4.8	Appendix 1. Distribution of survey respondents by countries . .	110
4.9	Appendix 2. Distribution of survey respondents by continents .	110
<b>5</b>	<b>Paper D</b>	<b>111</b>
5.1	Introduction . . . . .	112
5.2	Related Work . . . . .	113
5.3	Research Questions . . . . .	115
5.4	Approach . . . . .	115
5.4.1	Data Collection . . . . .	115
5.4.2	Feature Extraction & Establishing Ground Truth . . . .	117
5.4.3	Model Learning . . . . .	119
5.4.4	Evaluation of Results . . . . .	119
5.5	Result and Findings . . . . .	120
5.5.1	RQ1: Analysis of Selected Features . . . . .	120
5.5.2	RQ2: Classification Model Performance . . . . .	121
5.6	Discussion and Future Work . . . . .	123
5.6.1	Feature Selection . . . . .	123
5.6.2	Dataset . . . . .	123
5.6.3	Classification Algorithm . . . . .	123
5.6.4	Threats to Validity . . . . .	125
5.7	Conclusions . . . . .	125
<b>6</b>	<b>Paper E</b>	<b>127</b>
6.1	Introduction . . . . .	128
6.2	Related Work . . . . .	128
6.2.1	Existing Corpora of Software Modelling Artefacts . . . .	129
6.2.2	Other Software Architecture Collections . . . . .	130
6.2.3	Mining Architectural Knowledge . . . . .	130
6.2.4	Scientific Workflow Systems . . . . .	131
6.3	Experiences in Creating & Sharing a Collection of UML Software Design Models . . . . .	132
6.3.1	Models Extraction from GitHub . . . . .	132
6.3.2	Data Curation . . . . .	133
6.3.3	Sharing the Lindholmen dataset . . . . .	133
6.4	Challenges for Big-data Driven Empirical Studies in Software Architecture . . . . .	133
6.5	Directions for a Community Infrastructure for Big-data Driven Empirical Research in Software Architecture . . . . .	136
6.6	Overview of CoSARI . . . . .	139
6.6.1	Overview of the CoSARI Framework . . . . .	139
6.6.2	Main Use-cases of CoSARI . . . . .	143
6.6.3	Ongoing and Future Work . . . . .	144
6.7	Summary and Conclusions . . . . .	145

<b>7</b>	<b>Paper F</b>	<b>149</b>
7.1	Introduction . . . . .	150
7.2	Methodology . . . . .	151
7.2.1	Data . . . . .	151
7.2.2	Operationalization . . . . .	152
7.2.3	Analysis . . . . .	153
7.3	Results and Discussion . . . . .	153
7.4	Threats to validity . . . . .	154
7.5	Conclusions . . . . .	155
<b>8</b>	<b>Paper G</b>	<b>157</b>
8.1	Introduction . . . . .	158
8.2	Class Role Stereotypes . . . . .	159
8.3	Related Work . . . . .	159
8.4	Methodology . . . . .	161
8.4.1	Data Collection . . . . .	161
8.4.2	Ground Truth step 1: Criteria for Role Stereotypes . . . . .	162
8.4.3	Ground Truth step 2: Manual Labeling and Consolidation . . . . .	164
8.4.4	Feature Extraction . . . . .	164
8.4.5	Machine Learning Classification Experiments . . . . .	167
8.4.6	Generalizability of the Trained ML Classifier . . . . .	167
8.5	Experiment Results . . . . .	168
8.5.1	Multi-role Classification of all Stereotypes . . . . .	168
8.5.2	Single Role (Binary) Classification . . . . .	168
8.6	Classification Feature Importance . . . . .	170
8.7	Generalizability of the Classifier . . . . .	172
8.7.1	Generalizability Experiment 1: Single Case Training . . . . .	172
8.7.2	Generalizability Experiment 2: Double Cases Training . . . . .	173
8.8	New Applications of Role Stereotypes . . . . .	175
8.8.1	Stereotype-specific Design Metrics . . . . .	175
8.8.2	Using Role Stereotypes for Profiling Software Design Intention/Principles . . . . .	177
8.8.3	Collaboration Pattern between Stereotypes . . . . .	181
8.9	Threats to Validity . . . . .	182
8.10	Conclusion and Future Work . . . . .	182
<b>9</b>	<b>Paper H</b>	<b>185</b>
9.1	Introduction . . . . .	186
9.2	Role Stereotype . . . . .	187
9.3	RoleViz . . . . .	187
9.3.1	RoleViz in a Nutshell . . . . .	187
9.3.2	Compilation Unit . . . . .	188
9.3.3	Package . . . . .	189
9.3.4	Interaction . . . . .	189
9.4	Research Questions . . . . .	191
9.5	User Study . . . . .	191
9.5.1	Baseline . . . . .	191
9.5.2	Comprehension Tasks . . . . .	192
9.5.3	Participants . . . . .	193

9.5.4	Training Period . . . . .	194
9.5.5	Work Session . . . . .	194
9.6	Data Collection & Analysis . . . . .	195
9.6.1	Background Questionnaire . . . . .	195
9.6.2	TLX Questionnaire . . . . .	195
9.6.3	SUS Questionnaire . . . . .	196
9.6.4	Understanding Questionnaire . . . . .	196
9.6.5	Post-study Questionnaire . . . . .	197
9.7	Result . . . . .	197
9.7.1	Demographics of Participants . . . . .	197
9.7.2	Are the Comprehension Tasks Comparable? . . . . .	198
9.7.3	RQ1: Comparison between RoleViz and Softagram . . . . .	199
9.7.4	RQ2: Participant’s perception on the features of RoleViz . . . . .	201
9.8	Discussions . . . . .	202
9.8.1	Does participant’s experiences correlate with their perceived SUS, TLX and Understanding scores? . . . . .	203
9.8.2	Threats to Validity . . . . .	203
9.9	Related Work . . . . .	204
9.10	Conclusion and Future Work . . . . .	205

## **Bibliography**

**207**



# Chapter 1

## Introduction

Modeling is used in many walks of life, spanning over time and disciplines. Going back to early civilizations such as Ancient Egypt, Rome and Greece, modeling was used to create small-scale plans prior to building temples and sculptures [1]. In various science and engineering domains, *modeling* is widely used to provide abstractions of a system (to be implemented) at some levels of detail and precision. In the Software Engineering domain, *software modeling* is defined as “the designing of software applications before coding”, according to the Object Modeling Group (OMG) [2].

In modern software development, software modeling is considered as an essential part of the software architecture and design activity. Software models - one of the main outcomes of the modeling process - provide means to design solutions to the problem domain that needs to be addressed by software systems. Literature promises many benefits of software modeling in software development. For example, by modeling a system, one can be assured that design decisions captured in the software models are well documented, thus minimizing loss of information and misinterpretation in communicating the decisions taken during development. Software models also help to facilitate communication between team members. Besides, there exists evidences that software modeling has also been used in many industrial projects [3–5].

Despite the widely assumed benefits of software modeling, its actual usage and impacts to software development projects are subject to research. In the last decade, a number of empirical studies into software modeling has been conducted. As the Unified Modeling Language (UML) has become the de facto standard for software modeling in industry (since 1990s), many of these studies explore the effectiveness of UML modeling in software development, focusing on a costs and benefits perspective, and on industrial practice [3, 4, 6–13]. The findings, however, are diverse and partially contradictory. In a large-scale interview, Petre [13] found that the majority of the interviewees refused to use the UML because of its complexity, lack of formal semantics, inconsistency, and issues of synchronization between different diagrams. However, there are case-studies where UML is actively used and positively impacts software system. In another large scale survey (of over 250 professionals) about the state of the practice of MDD, Hutchinson et al. found that 73.4% of the people that have adopted MDD in their development process assert that models are used for

understanding a problem at an abstract level [14].

These seemingly contradict results trigger the following questions: Why is UML modeling successfully adopted in some cases but not in others?, Are there any common practices for applying UML modeling in software projects? Are there any common impacts of UML modeling in the development?. We find that it is difficult to properly answer the questions because: i) there is a surprising lack of empirical evidence on the use and impacts of software modeling [10, 12]; ii) software modeling artifacts (including software models) exist in many forms and formats, making it hard to systematically collect them for research; and iii) existing corpora of software models do not capture relevant context information in which software models are used (such as: purpose/goals of the models, stage of development where modeling is used, team composition). As a consequence, existing results are often not generalizable or comparable between different projects and case studies [15, 16].

The overall aim of this Ph.D study is *to empower empirical studies in software design and modeling by collecting and studying a large corpus of software modeling artifacts from real-life software systems*.

According to the Cambridge dictionary, “to empower” is defined as “to encourage and support the ability to do something”.<sup>1</sup> In this Ph.D study, on the one hand, we aim to provide a large corpus of curated software modeling artifacts that *enables researchers in the field to conduct various empirical studies into software modeling and design*. On the other hand, we expect knowledge learned from conducting empirical studies on the corpus could *encourage new (empirical) research directions into software modeling and design*.

The remainder of this chapter is constructed as follow. Section 1.1 presents in detail the research goals, research questions and scope of the thesis. Section 1.2 provides reader with background of the study. Section 1.3 discusses the methodology. Section 1.4 presents a short summary of each paper and their contribution to the goals of the thesis. In Section 1.5, we put together findings from individual papers to answer eight research questions of the study. Subsequently, we provide a discussion on the fulfillment of the research goals of this Ph.D study. Section 1.6 provides further discussions on the software modeling and design practices. Section 1.7 presents the threats to validity to this Ph.D study and Section 1.8 wraps up this chapter with recommendations for future work.

## 1.1 Research Focus

This section will provide readers with details about the research goals, research questions and scope of this Ph.D study.

### 1.1.1 Goals of the Ph.D Study

This Ph.D study aims at enabling empirical studies in software design and modeling by collecting and studying a large corpus of software modeling artifacts from real-life software systems. The three goals of this Ph.D study are:

---

<sup>1</sup><https://dictionary.cambridge.org/dictionary/english/empower>

**Goal G1.** To build and share (with researchers and practitioners in the field) a large corpus of curated software modeling artifacts.

**Goal G2.** To conduct empirical studies on practices of software modeling in real-life software development.

**Goal G3.** To conduct empirical studies on impacts of software modeling in real-life software development.

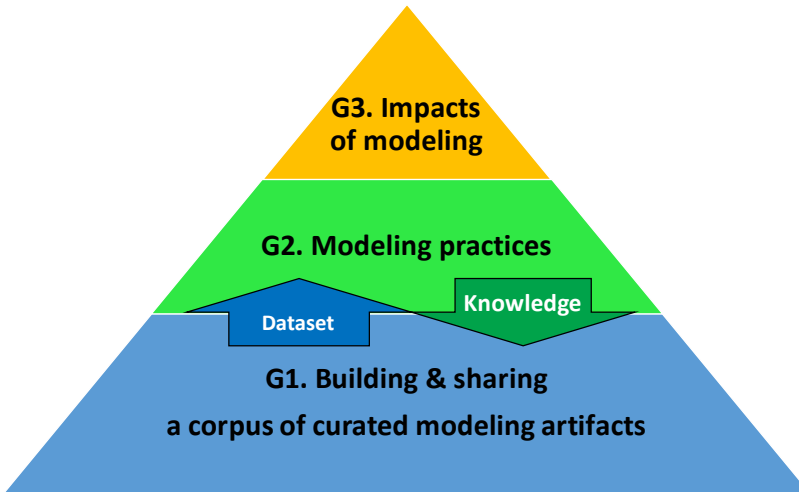


Figure 1.1: Pyramid of goals of the PhD Study

Figure 1.1 presents the three goals of this Ph.D study in a pyramid shape, indicating the direction in which empirical insight/knowledge is gained from analysing raw-data. In particular, at the bottom layer of the pyramid, **G1** aims at building and establishing a large corpus of software modeling evidence. This is the foundation on which empirical studies in **G2** and **G3** (presented in higher layers of the pyramid) build.

Figure 1.1 also shows the bi-directional relationship of the goals. On the one hand, **G1** provides empirical data for **G2** and **G3**. On the other hand, knowledge gained from conducting quantitative and qualitative analyses on the dataset can be used to better organize the original data.

The Ph.D study is designed in a replicable manner. Accordingly, data obtained from **G1** is open for public access. Methods are described step by step, limitations and threats to validity are carefully discussed in the chapters.

### 1.1.2 Scope and Expected Outcomes of the Ph.D Study

In this section, we firstly present the general scope of this Ph.D study. Subsequently, we present the scope and expected outcomes of the three main goals of this thesis.

### 1.1.2.1 General Scope of the Ph.D Study.

This Ph.D study concerns with practices and impacts of software modeling and software design in software development. Therefore, it is needed to understand the scope of this Ph.D study towards software modeling and software design as the following:

**Definitions: software design and software modeling.** According to the Cambridge dictionary, *design* (Noun) is “the plans, drawings, etc., that show how something can be made”.<sup>2</sup> *Software design* can be understood as the process of making decisions about the software that is to be built or created. Different from software design, *software model* is an abstract representation of a system. *Software modeling* is defined as “the process of creating a software model, i.e. choosing what to represent and how to represent it”.

**Scope: Software design and software modeling.** Modeling and designing often go hand-in-hand. A model is used to understand, reason, analyse, break-down, which leads to adaptation and refinement of the design. Software design and models are often documented together in same documents and there is no clear distinction between them. In the context of this Ph.D study, we aim to collect both software design and software modeling artifacts. We chose to start with collecting software models in a common modeling language which is the Unified Modeling Language (UML). This choice of data collection has a consequence on the type and subjects of empirical studies of this Ph.D study. In particular, we are able to study practices and impacts of UML modeling in software projects.

### 1.1.2.2 Scope and Expected Outcomes of Goal G1

Goal **G1** refers the creation of a large collection of curated software modeling artifacts. We scope the type of modeling artifacts, place to collect data, data curation activities and the expected outcomes as the following:

**Scope: What software modeling artifacts?** Software modeling artifacts exist in a very wide range of files and formats. Most common modeling artifacts include software models, software architecture documentation. The focus of our corpus is on software models that describe (parts of) the design of software systems by means of the UML modeling language. In particular, we shall collect UML models and relevant context information (such as source code, descriptive information of the corresponding projects) as the main parts of the corpus.

**Scope: Where to collect data?** When it comes to data collection, an obvious question is *where to collect the data from*. One option is to collect UML models from industrial companies. This has pros and cons. The main advantage of industrial data is industry-relevant contexts (including rationale, business decisions) behind UML use. The main drawback lays in data availability. Particularly, it is difficult to collect industrial cases where UML is used, because companies consider their design as commercially sensitive information or as

<sup>2</sup><https://dictionary.cambridge.org/dictionary/english/design>

a reflection of their state of IT-affairs. This could further limit the study's replication. An alternative option is to collect UML practices from OSS projects. Data availability and transparency are clearly the main advantage of OSS projects compared with industrial cases. Public access to not only UML file but also other resources such as source code, documentation, issues, commit message, etc. would allow researchers to put the UML file in its usage context. The main challenge is to identify OSS projects that use UML. This is due to the large variety of UML file formats and the lack of support from most open source platforms, such as GitHub, for model versioning. Goal **G1** of this thesis focuses on finding UML modeling artifacts in OSS projects and takes the challenge of identifying OSS projects that use UML.

**Scope: Data curation.** According to Lord et al., data curation is defined as “the activity of managing and promoting the use of data from its point of creation, to ensure it is fit for contemporary purpose, and available for discovery and reuse” [17]. The authors further added that “Higher levels of curation will also involve maintaining links with annotation and other published materials”. In the context of this Ph.D study, data curation is undertaken as the sequential steps of data collection and might include the following activities:

- Enriching understanding about the collected data. This can be done by means of observational or analytical assessment on the characteristics of the data.
- Enriching the (meta-data of the) collected data with knowledge gained from systematic analyses/studies on the data, e.g. by annotating research findings to the data.
- Filtering relevant data for conducting empirical research in the field of software modeling.

**Expected Outcomes.** A corpus of UML models and their relevant context information from GitHub OSS projects. The UML models should be labeled by some their characteristics such as “type of the UML models”, etc. The corpus should be designed in an easy-to-maintain manner so that other researchers can reuse and contribute to labeling of the models.

### 1.1.2.3 Scope and Expected Outcomes of Goal G2

Goal **G2** aims at conducting empirical studies on practices of software modeling with the following scope and expected outcomes:

**Scope: What are modeling practices?** According the Cambridge dictionary, the noun “practice” means “something that is usually or regularly done, often as a habit, tradition, or custom”.<sup>3</sup> In the context of this Ph.D thesis, “modeling practices” is understood as “the ways/habits in which modeling is often done or applied within a software project”.

---

<sup>3</sup><https://dictionary.cambridge.org/dictionary/english/practice>

As being the behaviours formed inside human brain, (modeling) practices can be categorised into conscious and unconscious practices. Conscious modeling practices refers to modeling practices which are well acknowledged by software practitioners, e.g. using UML as the common modeling language, using UML models for communication. Unconscious modeling practices, in contrast, are those practices that appear across many software projects without much conscious attention, e.g. introduction time of the UML models, time span of UML models.

In the context of this Ph.D thesis, we take both types of modeling practices into account. The type of studying modeling practices would subsequently drives the selection of methods for collecting the practices. For example, to collect conscious practices, it might be enough to conduct a survey among modeling practitioners, while identifying unconscious practices might be done via statistical analysis on a big sample of projects.

**Scope: Notion of “good” modeling practices.** It is a common thought that following “good” practices often lead to producing good results ([18], pp.110). In fact, it is difficult to explicitly conclude whether a modeling practice is good or bad without a consideration of the context in which the practice is used. For example, a modeling tool that is effectively used in a project might not be generally applied to other projects where the needs are different (e.g. some projects might prefer drawing tools over modeling tools). In this Ph.D thesis, we report our observation on the modeling practices without making judgement on whether they are good or bad.

**Expected Outcomes.** Quantitative and qualitative analyses of the practices of UML modeling in software systems. Examples of such analyses are time period when UML is introduced, whether UML models are updated, and rationale behind the use of UML.

#### 1.1.2.4 Scope and Expected Outcomes of Goal G3

Goal **G3** refers conducting empirical studies on impacts of software modeling in software development. The scope and expected outcomes of **G3** are the following:

**Scope: Impacts in software development.** Impact is defined as “a marked effect or influence”, according the Cambridge dictionary.<sup>4</sup> In the context of this Ph.D research, we study the effects of software modeling and design on various aspects of software development, e.g. communication between team members, software quality. For example, empirical findings regarding impacts of UML modeling should investigate whether use of UML models related/associated with improved/reduced quality aspects of the software systems. We follow the cost and benefit model proposed by Chaudron et al. to categorise the impacts [19].

**Expected Outcomes.** We hope to establish relation between the use of UML modelling and different aspects of software development.

<sup>4</sup><https://dictionary.cambridge.org/dictionary/english/impact>

### 1.1.3 Research questions of the Ph.D thesis

To reach the goals of the Ph.D thesis, we formulate the following research questions:

- **RQ1.** How to build a large corpus of models?
- **RQ2.** How can we share and promote use of the modeling corpus?
- **RQ3.** What are purposes of using UML in OSS projects?
- **RQ4.** How is UML used in OSS projects?
- **RQ5.** What are practices for using UML modeling in software development?
- **RQ6.** What are perceived impacts of using UML in OSS projects?
- **RQ7.** Does the use of UML modeling correlate with lower defect proneness?
- **RQ8.** Does using class role stereotypes correlate with better understanding of designs of software system?

Among the research questions, **RQ1** and **RQ2** aim for **G1**; **RQ3**, **RQ4** and **RQ5** aim for **G2**; **RQ6**, **RQ7** and **RQ8** target **G3**.

Different from **RQ3** to **RQ7** which focus on practices and impacts associated with UML modeling, **RQ8** questions the impacts of using a software design concept (i.e. class-role-stereotype) on assisting developers in their software comprehension tasks.

## 1.2 Background

In this section, we aim to provide readers with background knowledge on which the thesis builds. Firstly, this thesis focuses on the Unified Modeling Language (UML) as the main software modeling artifact. Section 1.2.1 gives a brief overview of the UML.

One of the main focuses of this thesis is to build a corpus of modeling artifacts for empirical research in the field. Section 1.2.2 will give a review of existing corpora of modeling artifacts, including corpora of UML models.

Another focus of this thesis is conducting empirical research in software modeling. Section 1.2.3 highlights a number of empirical findings in the field, with a focus on impacts of using UML in software engineering projects.

### 1.2.1 The Unified Modeling Language (UML)

The evolution of UML is described in detail by Kobryn [20]. The UML emerged from the competition in creating notations for object-oriented design by Booch [21], Rumbaugh et al. [22], Jacobson et al. [23] and other researchers in the early nineties of the 20th century. The 1.1 version was proposed in January 1997 and officially adopted by the Object Management Group (OMG)

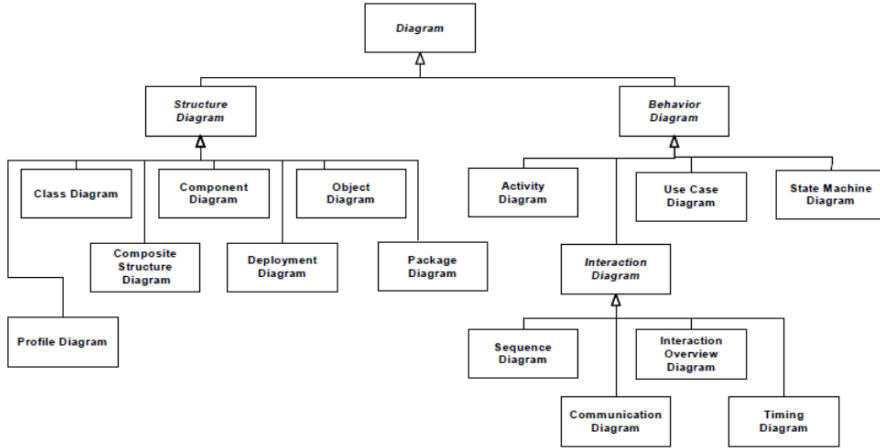


Figure 1.2: Taxonomy of UML diagrams (UML 2.0)

later that year. Since then, the UML has undergone many revisions, with UML 2.0 released in 2005 and most recently UML 2.5 in June 2015.

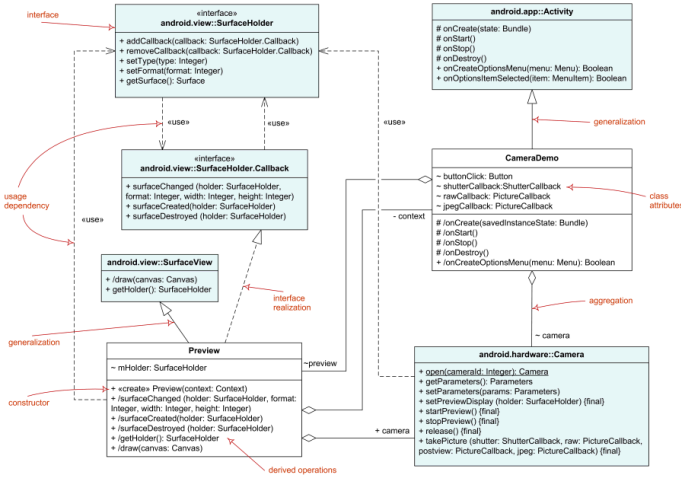
UML 2.5 has 14 diagram types. They are divided into two categories, namely structure- and behavior- diagrams. Structure diagrams show the static structure of systems, while behavior diagrams show the dynamic behavior of systems, including their methods, collaborations, activities, and state histories. Figure 1.2 shows a taxonomy of UML diagrams.

In the OMG’s view, “modeling is the designing of software applications before coding”. The OMG promotes model-driven architecture as the approach in which UML models of the software architecture are developed prior to implementation. However, as the UML is a language and not a method, there are much more ways of using it, e.g. for reverse engineering, refactoring, documenting an existing system, etc. In addition, the UML is a methodology-independent language, one could use it in different software development processes (e.g. when planning, analyzing requirements) and in different software development methods (e.g. water fall or agile approach). In this thesis, the UML is considered in all contexts where it is used, not limited to model-driven per se. Figure 1.3 shows examples of three different types of UML diagrams.

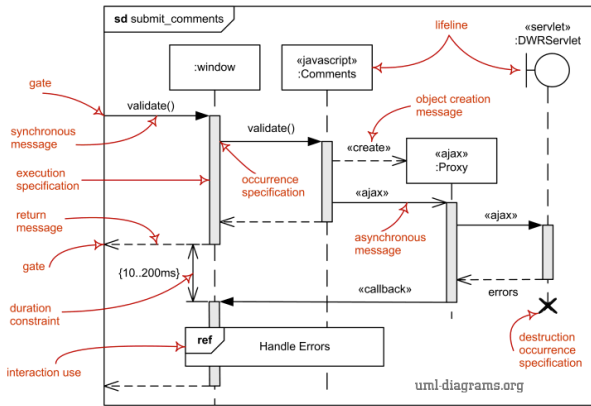
## 1.2.2 Existing Corpora of Software Modeling Artifacts

Störrle et al. introduced the Software Engineering Model Index (SEMI) which contains a list of contemporary model repositories [24]. We take this as a starting point for our search of software modeling corpus. In fact, 3 out of 8 corpora to be reviewed in this section are listed in SEMI. In the paper, the authors also outline four main challenges when building a successful model repository: i) Archiving (“How to archive data with very high reliability, for very long time, yet readily accessible, and economically viable?”), ii) Access Support (“How to search for models?”), iii) Intellectual Property (“How to manage intellectual property such as models?”), and iv) Incentives (“How to motivate researchers/practitioners to publish their models?”).

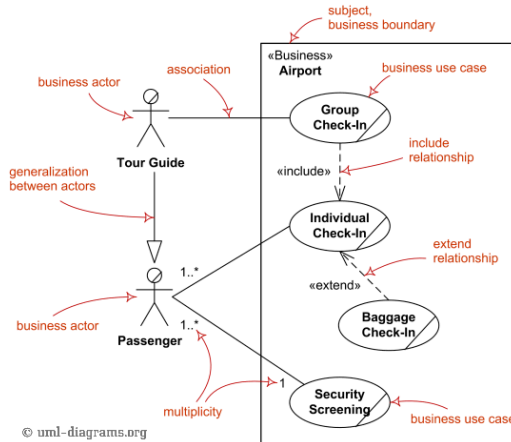




(a) Class diagram



(b) Sequence diagram



(c) Use-Case diagram

Figure 1.3: Examples of UML diagram (Source: <http://www.uml-diagrams.org/>)

The *Repository for Model Driven Development (ReMoDD)* is created to support researchers and practitioners in sharing exemplar models and other modeling practices [25]. Currently, it contains around 90 modeling artefacts, including models in different modeling languages and artefacts of some MDD conferences. Models are stored in various formats, mostly PDF but also some in XMI.

The *Open Models Initiative (OMI)*<sup>5</sup>, similar to ReMoDD, offers a platform that allows researchers and practitioners to share models. It is currently hosting around 70 models stored mostly in image-formats. There is no report on whether the models are derived from industrial or academic contexts.

Karasneh et al. used a crawling approach to automatically fill an online repository with so far more than 700 model images<sup>6</sup> from Google Image Search [26]. This work focus on the models only and do not take their project context into account. Further, Karasneh et al. do not distinguish between models that stem from actual software development projects and models that are created for other reasons, e.g. teaching.

Mengerink et al. collected a data set of 9,188 OCL expressions derived from 504 EMF meta-models in 245 GitHub repositories [27]. To this end, the authors firstly performed a couple of GitHub searches, then downloaded all *.ecore* and *.ocl* files in the result list, then removed all duplicated files and finally parsed all the unique files to extract OCL expressions.

Basciani et al. built MDEForge as a web-based modeling platform which aims at fostering a community-based modeling repository [28]. The number of meta-models hosted in this platform is not available.

GenMyModel<sup>7</sup> is a web-based online tool that supports collaborative modeling for UML, BPMN, RDS, and flowcharts [29]. At the time of writing, GenMyModels claims to host about 777,000 diagrams. However, it is not clear how many of these diagrams are open to public access and how many are private.

While they do not represent software designs, it is interesting to look at the neighbouring discipline of business process models as they do share a lot of commonalities, a.o. in being a model-centric approach. Moreover, there have been significant efforts in building corpora of BPM models: The *BPM Academic Initiative (BPM AI)* is a platform where business process models are shared for teaching purposes [30]. A business process model is defined as a set of business activities and execution constraints between these activities [31]. It can be used to describe complex interactions between business partners and to indicate related business requirements on an abstract level. Currently, BPM AI claims to host 29,285 business process models in various machine-readable formats. The dataset has however not been updated since 2012. The process of collecting models is not clearly mentioned; apparently, most of the models in the dataset derive from students as part of modeling assignments.

In summary, the existing corpora are rather small in term of number of modeling artifacts. In addition to their small size, these repositories seldom include other artifacts than the models, making it impossible to study the models in the environment of actual projects. In this Ph.D study, we collect not

---

<sup>5</sup><http://openmodels.org>

<sup>6</sup><http://models-db.com/>

<sup>7</sup><https://www.genmymodel.com/>

only UML models but also meta-data of the projects in which the models are found of the UML models (e.g. UML file commits, corresponding committers, project founders, number of contributors, etc.).

### 1.2.3 UML use and impacts of using UML in software engineering projects

In this section, we provide reader with an overall picture of UML use and impacts of using UML in industry and in OSS projects by summarizing related works.

#### 1.2.3.1 UML use in industry

**UML is widely studied in industry, however, there are needs for more experiments and case studies** Budgen et al. [12], in their systematic review, identified 49 empirical studies of UML published up to the end of 2008. Among them, 12 papers were about UML metrics, 14.5 about model comprehension, and 7.5 about model quality (half points indicate papers with more than one focus). Only 2 papers addressed UML adoption, i.e. by Anda et al. [3] and Grossman et al. [4]. These two studies identify a range of benefits and drawbacks associated with the adoption of UML. They particularly highlight the need for further research relating to the UML and its adoption as well as the need for, and importance of, an adequate level of training.

Another systematic review conducted by Fernández-Sáez et al. [5] identified 38 papers (published up to the end of 2010) that report 63 empirical studies of UML use in software maintenance processes. Only 3 of them are case-studies. Most research (60 empirical studies) concerns the maintainability and comprehensibility of the UML diagrams themselves. The authors conclude that there is a need for more experiments and case studies to be performed in industrial contexts.

Nugroho and Chaudron [10] also argue that “Despite the fact that UML is widely used in practice, little is known about how UML is actually used”.

**UML adoption and its impacts are still under discussion** Modeling has been widely studied in industry, in particular in several survey studies. Torchiano et al. [32] found that models help to improve design and documentation. However, they also found that model usage is connected to extra effort, especially due to a lack of supporting tooling. Forward et al. [33] found that models are primarily used for design and documentation, while code generation is rather seldom. Gorschek et al. [34] focused on a different population, which are programmers, partially working in industry and open source. Within their sample design models are not used very extensively. However, models and UML are found to be used mainly for communication purposes. Further, they report on a higher use of models for less experienced programmers.

Hutchinson et al. studied the state of the practice of model driven development (MDD) for which over 250 professionals were surveyed [14]. Analyses of the responses show that people that have adopted MDD in their development process use models for various purposes, e.g. for communication with team

members, for testing and for model simulations (for verification and validation purposes).

Chaudron et al. discuss the effectiveness of UML modeling with a special focus on its cost and benefits [19]. Figure 1.4 shows their views on benefits of UML modeling to different aspects of software development characteristics. In particular, UML modeling can benefit software development projects in many ways, e.g. developers can use UML models to obtain better understanding about problem domain as well as the solution space. We discuss our findings about impacts of UML modeling with a reference to this benefit model.

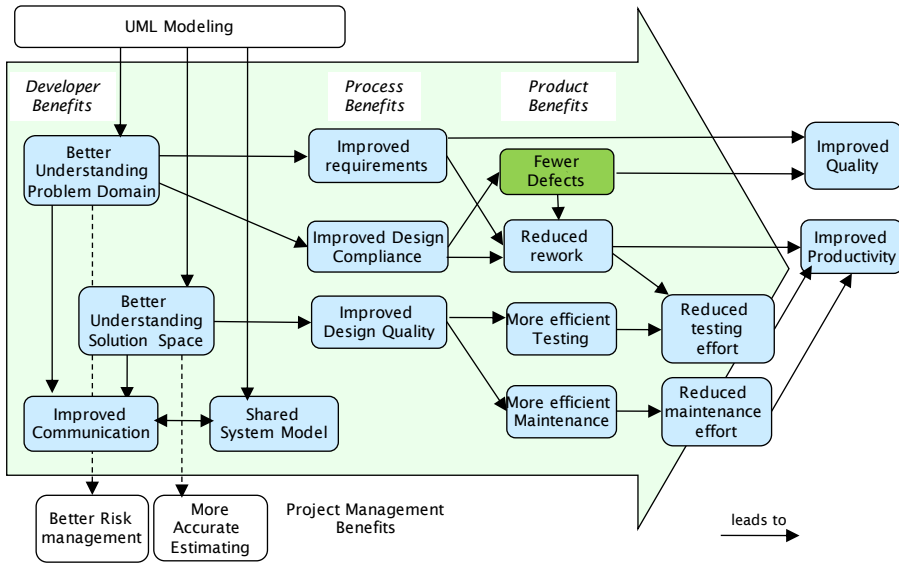


Figure 1.4: Benefits of UML modeling. Source: Figure 6, Chaudron et al. [19]

Dobing and Parsons [6] report on a survey of 171 UML users (plus 11 who use UML components within another OO methodology). The authors found that "only class diagrams are being used regularly...". The majority of respondents found that all aspects of UML are useful for most projects. The authors also suggest that complexity and lack of usage guidelines are the biggest concerns with UML.

Lange et al. [8] conducted a web-based survey in 14 industrial companies. On the basis of the responses, they identified four main classes of problems encountered: scattered information (e.g., design choices dependencies); incompleteness, disproportion (more detail for some parts than others), inconsistency.

Nugroho et al. [10] conducted a survey among 48 professional developers (from 10 different countries) about their perception toward correspondence between source code and design. Respondents identify incompleteness of the UML design as the most prominent factor that often forces them to deviate from a UML design.

Scanniello [11] conducted a survey at 22 companies regarding the use of UML in software development and maintenance. Survey responses show that the majority of the companies (20) use UML in the analysis and design phases.

Besides these big surveys, *case studies* were performed in order to investigate the impact of the modeling/UML usage. For example, Baker et al. [7] found

an increase of productivity when using UML in Motorola. Also Nugroho et al. [35] investigated an industrial case study and found that UML usage has the potential to reduce the defect density and, thus, increase the quality of software. Just as in the case described by Kuhn et al. [36], most of the case studies draw a picture of model use, where models are actually artifacts that are produced and consumed by different people. Anda et al. [3] reports a case study in ABB. This paper found anecdotal advantages of modeling such as improved traceability. This paper also pointed to potential trade-offs, such as time spending to integrate legacy code with models and organizational changes needed to accommodate modeling.

Petre et al. [13] reported a series of interviews conducted over 2 years with more than 50 practicing professional software developers. The author found out that the majority of interviewees do not use UML, and those who do use it tend to do so selectively and often informally.

Dzidek et al. [9] performed a controlled experiment to investigate the influences of the use of UML to maintenance task (20 professional developers). The result of the experiment shows a positive influence of the presence of UML for maintainers. UML also helps novice developers to produce code of better quality.

#### 1.2.3.2 UML in OSS projects

Much less study has been done on UML use in OSS. One reason for this is the challenge to actually find cases that can be studied. For example, Badreddin et al. studied 20 projects, without finding UML and concluded that it is barely used in open source [37]. Similarly, Ding et al. [38] found only 19 projects with UML when manually studying 2000 open source projects.

There are several investigations of single or very small numbers of cases of open source projects that use UML, e.g. by Yatani et al. [39], who found that models are used to describe system designs, but are rarely updated. Osman et al. [40] studied to what extent classes in the diagrams are implemented in the code. Finally, Kazman et al. [41] investigate the Hadoop Distributed File System to learn how documentation impacts communication and commit behavior in the open source system. There are some studies that approach model use in open source with a quantitative perspective, studying large numbers of projects. For example, to study the use of sketches, Chung et al. [42] collected insights from 230 persons contributing to 40 open source projects. Finally, Langer et al. [43] studied the lifespan of 121 enterprise architect models in open source projects.

## 1.3 Methodology

In this thesis, we employ constructive research methods and empirical research methods to achieve the three goals **G1**, **G2** and **G3**. In particular, G1 can be addressed constructively: by building a system that satisfies the goal. Goal G2 and G3 deal with the practices and impact in actual software development projects. Hence these are best answered by conducting empirical research on real-life software systems.

Table 1.1 shows the use of the research methods in the 8 papers included in this thesis. Sections 1.3.1 and 1.3.2 discuss the use of each method across the papers. A more-detailed discussion of the research strategies is found in the included papers.

Table 1.1: Summary of research methods used in this Ph.D study

Paper		A	B	C	D	E	F	G	H
Research Methods	<i>Constructive</i>	X	X	X	X	X			X
	<i>Empirical</i>	Survey		X	X				
		Experiment						X	
		Case study							X
		User study							

### 1.3.1 Constructive research

According to Crnkovic [44], the constructive research method implies building of an artifact that solves a domain specific problem in order to create knowledge about how the problem can be solved (or understood, explained or modeled) in principle. Artifacts such as models, diagrams, plans, organization charts, system designs, algorithms and artificial languages and software development methods are typical outcomes of constructive research. In the context of this Ph.D study, we followed the constructive research approach to achieve goal **G1** and part of goal **G3** as the following:

**Constructing a process to build and share a large corpus of software modeling artifacts.** To achieve goal **G1**, we construct a process to: i) identify UML models from OSS projects in GitHub; ii) crawl the identified models and meta-data of corresponding projects from GitHub to form the corpus; iii) curate the corpus and iv) promote the use of the corpus among researchers and practitioners in the field. The steps are documented and can be (selectively) applied to collect other modeling artifacts (such as software architecture documentation) from various Git data sources other than GitHub (such as GitLab or any company’s in-house Git server).

The constructive method is applied in the five studies A, B, C, D, and E. In particular, in paper A, we proposed an automated method for automatically identifying UML class diagram images among ordinary images. In paper B, the method described in Paper A was combined with other data-mining techniques (e.g. GHTorrent [45], CVSanaly [46]) in a complete process to identify and crawl UML diagram files in 10% of GitHub non-forked repositories (about 1.2 million projects). In paper C, we applied the process described in paper B to obtain UML files and meta-data of corresponding projects from all GitHub non-forked repositories (around 12 million projects). The collected data (called Lindholmen dataset) has gone through some steps of curation. The curation steps are discussed in detail in paper C and paper D. Finally, in paper E, we presented a reference architecture of an infrastructure (called CoSARI) that promotes the use of the corpus as well as collaborative research in the field of software architecture.

**Constructing a visualisation tool for software comprehension.** We applied the constructive method to build a visualisation tool called RoleViz for software comprehension. In particular, the visualisations in RoleViz were developed via a number of iteration of collecting ideas, building prototype, collecting and analysing feedback, and implementing/updating the visualisation. The process is described in paper H.

### 1.3.2 Empirical research

The aim of empirical research methods is to gain knowledge by means of direct and indirect observation or experience [47]. In the context of this Ph.D study, empirical methods are used to enrich understanding of the use of UML modeling and its impacts within OSS projects. In particular, four empirical methods, i.e. survey study, experiment, case study, and user study were used to achieve goal **G2** and goal **G3**.

**Survey study.** Survey study is commonly used to identify the characteristics of a broad population of individuals [47]. In this Ph.D study, survey studies were used to establish a representative picture of the practical use and impacts of using UML in OSS projects. Survey research is most closely associated with the use of questionnaires for data collection. Survey research can also be conducted by using structure interviews, or data logging techniques [48]. We used different data collection methods for the survey studies presented in paper B and C.

In paper B, we surveyed 3,295 OSS projects which were identified as using UML modeling. By quantitatively analysing meta-data of the projects, we could gain understanding on the practical use of UML in OSS projects at the descriptive level, e.g. whether UML models are used/updated, and if so, active time of UML models, etc.

In paper C, qualitative and quantitative data was collected using a survey questionnaire. While quantitative data gives us an overview of OSS developer's opinion regarding purposes and impacts of using UML, qualitative data provides us with the rationale behind developer's answers. We combined quantitative and qualitative analysis in order to draw implications concerning UML use.

**Experiment.** An experiment is “an investigation of a testable hypothesis where one or more independent variables are manipulated to measure their effect on one or more dependent variables” [48]. Experiments are therefore often used to determine in precise terms whether a cause—effect relationship exists between the variables.

In the context of this Ph.D study, an experiment was used to study the intuitive and widely held belief that “the use of UML modeling, on average, should correlate with higher software quality”. In particular, in paper F, we employed a “quasi-experiment” to compare the *defect proneness* between two groups of OSS GitHub projects: a treatment group of 50 projects using UML models; and a control group of 93 projects sampled randomly using GHTorrent [45]. The term “quasi” refers to the fact that the study unit (*having UML models*) was not randomly assigned to experimental groups. Independent variables include *project age*, *primary programming language*, *number of contributors*, etc. In

order to test our hypothesis, we built a multiple linear regression model, a robust technology which enables us to estimate the effects of *having UML models* on defect proneness *while holding the other independent variables fixed*.

**Case study.** According to Yin, empirical case study refers to “an empirical inquiry that investigates a contemporary phenomenon within its real-life context, especially when the boundaries between phenomenon and context are not clearly evident” [49]. Case studies help to achieve in-depth understanding of how and why certain phenomena occur and the mechanisms by which cause-effect relationships might happen [50].

In paper G, a multiple-case study was conducted to understand the existence of class role-stereotype and its relationship to design characteristics of source code in three studied software systems. In particular, we quantitatively analysed the correlation between class’s role-stereotype and its design characteristics (such as *WMC - weighted method per class* and *CBO - coupling between objects*). Result obtained from the analysis allows us to propose new applications of role-stereotype, e.g. in capturing software system’s design style and intention.

**User study.** User studies focus on understanding user behaviors, needs, and motivations through observation techniques, task analysis, and other feedback methodologies. According to Kuniaysky, it is “the process of understanding the impact of design on an audience” [51]. In the context of this Ph.D study, a user study is used to understand the impact of software modeling in assisting developers in software comprehension tasks. In particular, in paper F, we created an interactive visualization called RoleViz that visualizes system architectures in which architectural elements are annotated with their role-stereotypes. Then, we conducted a comparative user-study in which developers use RoleViz and Softagram (a commercial tool for software architecture comprehension) to solve two separate comprehension tasks on a large open source system. We compared RoleViz against Softagram in terms of participants’: (i) perceived cognitive load, (ii) perceived usability, and (iii) understanding of the system.

## 1.4 Contributions

In this section, we summarize and state the main contributions of the eight papers on which this Ph.D thesis builds. Figure 1.5 offers an overview of the contributions of the eight papers to the main goals and research questions of the Ph.D thesis. The included papers are presented in a rectangle shape and the research questions are presented in a round shape. The papers and research questions are placed at one or more layers of the Ph.D Goal Pyramid corresponding to the goals they contribute to. Papers that span over multiple layers of the pyramid are expected to contribute to more than one goals, e.g. paper C contributes to all three goals of the Ph.D study. The arrows between rectangles indicate the direction in which the papers are built and extended. In particular, the paper/study at the head-end extends the study/paper at the tail-end. For example, the machine learning classification model presented in paper A was used as part of the mining process in paper B.



In this overview, paper E has two instances at both the bottom and top layers of the pyramid for two reasons:

- [a] Paper E aims at building an infrastructure for collaborative research in software design, thus belong to the foundation layer.
- [b] The infrastructure in paper E, once built, will enable the application of the automated classification model presented in paper G. The dashed rectangle and arrow indicate that such infrastructure has not been built yet.

Research questions are subsequently answered in section 1.5.

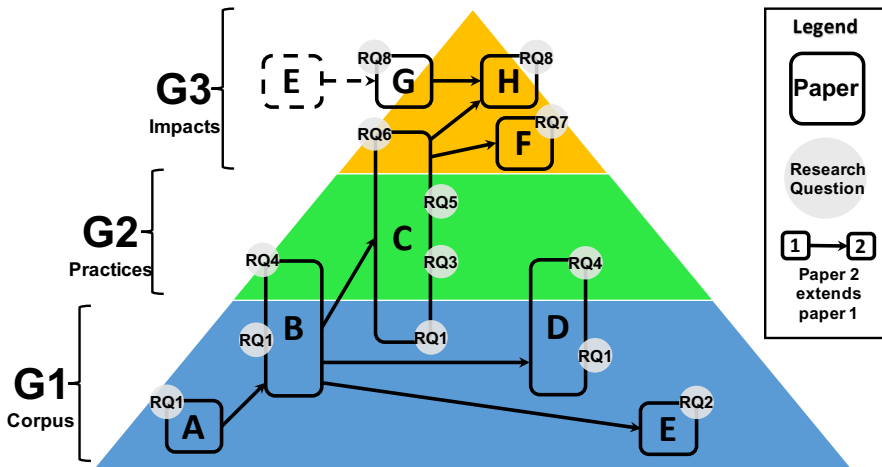


Figure 1.5: Contributions of papers to specific goals and research questions

### 1.4.1 Paper A: Automatic classification of UML class diagrams from images

The UML is a graphical modeling language. Therefore, it is common to store UML models in graphical file formats such as .png, .jpg, etc. While there is a need for collecting UML models for empirical research, current research methods often lack the systematical identification of images that represent UML diagrams.

Paper A presents an automated classification method for images that represent UML class diagram (UML CD). Building the classification would on one hand help improving the data collection process of the existing UML repositories, such as the one by Bilal et al. [26]. On the other hand, and more importantly, this could further open up the possibility to automate the UML crawling process and to build a larger collection of UML class diagrams.

To build the classifier, we use a combination of image processing and machine learning. Figure 1.6 shows the processes of the paper A (this is cloned from Figure 2.1 in the Chapter 2). We first introduced 23 features that capture statistical and geometric characteristics of UML class diagrams. A tool that extracts these features from images was built accordingly. Finally we employed 6

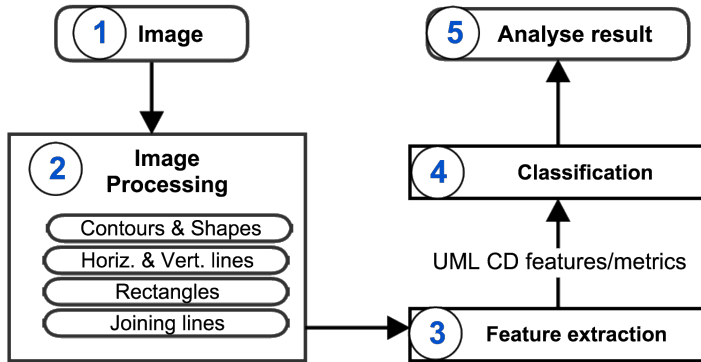


Figure 1.6: Overall process of paper A

well-known classification algorithms, including Decision Table, Random Forest, Support Vector Machine, Logistic Regression, REP-Tree and J48 Decision Tree [52], to build the classifier. The classification performance of the defined image processing features and algorithms was assessed in terms of classifying accuracy and robustness via several metrics, i.e. information gain, specificity and sensitivity. A dataset of 1 300 different images was collected from the Internet through Google Image Search (750 UML CD images and 750 non-UML CD images) for training and testing the classifier<sup>8</sup>. This dataset was shared as a benchmark.

The main contributions of the paper are:

- [a] A machine learning method to build an automated classifier of UML CD images. We find out that our method could reach 95.9% and 91.4% (respectively) of correct classification of input images for UML CD and non-UML CD.
- [b] Evaluation of classification performance of six algorithms on different feature-sets in terms of classifying accuracy and robustness.
- [c] A dataset of 1,300 images was shared as a benchmark for other classification methods.

Overall, paper A encourages us to believe that it is possible to automate the process of identifying images that contain UML notations with high accuracy. This is a first step towards building up an automated process to identify UML models from OSS projects (part of goal **G1**). This work can be extended in different ways: e.g. i) Using the same approach to create classifiers for sequence diagram, use-case diagram images; and ii) Involving text-recognition to enhance classification performance.

## 1.4.2 Paper B: The quest for open source projects that use UML: mining GitHub

Little is known about UML use in open source projects. This is due to the so far limited success in identifying open source projects that use UML modeling.

<sup>8</sup>Image set - <http://bitly.com/dtsUMLClassifier>

The lack of available data is the reason why so far no answers could be given to different basic questions on the amount of UML files in open source projects, the time span during which models are created or updated during the open source project, or the question which of the project’s contributors do create models.

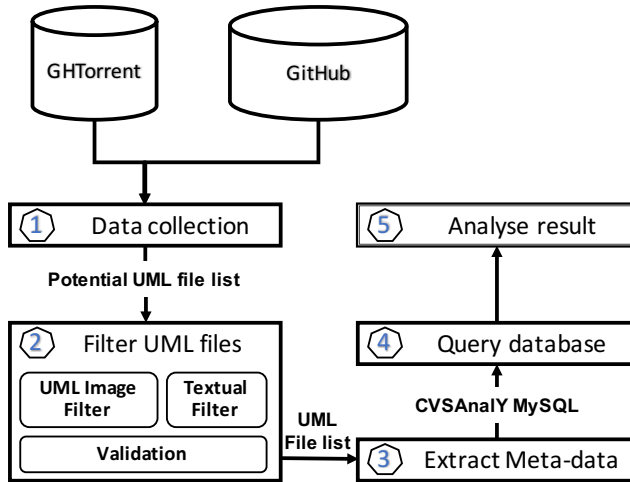


Figure 1.7: Overall process of paper B

In paper B, we contribute to this body of knowledge by following a five-step approach. Figure 1.7 shows the steps in a sequential order (this is cloned from Figure 3.1 in the Chapter 3). First, we combined different technologies, including the classifier in Paper A, to form a semi-automated process for collecting UML models stored in images, .xmi, and .uml files from over 1.2 millions GitHub projects (10% of the whole) (Step 1 and 2). Secondly, in step 3, 4 and 5, we collected meta-data of the projects that used UML and quantitatively analyzed the data to address the following research questions:

- Are there GitHub projects that use UML? Which are these projects?
- Are there GitHub projects in which the UML models are also updated?
- When in the project are new UML models introduced?
- What is the time span of “active” UML creation and modification?
- Are UML models copied across projects?

The main contributions of the paper are:

- [a] A semi-automated process for collecting UML models stored in over 1.2 millions GitHub projects (goal **G1**).
- [b] A list of 21,316 UML diagrams from 3,295 GitHub projects and their meta-data. The list is available at the replication package of the paper <sup>9</sup>. This is the first time the modeling community can establish a corpus comparable to collections already exist for source code only (goal **G1**).

<sup>9</sup><http://models.cs.chalmers.se/oss/Downloads/ReplicationPackage/>

- [c] Quantitatively answering a number of research questions regarding the use of UML in OSS projects (goal **G2**).

### 1.4.3 Paper C: Practices and perceptions of UML use in open source projects

Paper B reveals some facts about the use of UML based on quantitative analysis on meta-data of OSS projects. These facts trigger us to discover the rationales behind the use of UML and impacts of using UML in OSS projects.

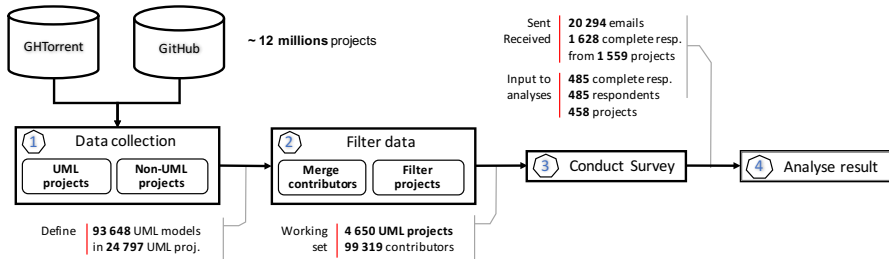


Figure 1.8: Overall process of paper C

Paper C represents our large-scale survey on software developers of GitHub projects that use UML. The overall process is shown in Figure 1.8 (this is cloned from Figure 4.1 in Chapter 4). We first extended the data collection method presented in paper B to 100% of all GitHub repositories (over 12.8 millions repos). This resulted in a dataset of over 93,000 UML models from over 24,000 GitHub projects. A number of filters was then applied in order to filter out those projects that were suspected not to be representative for serious software engineering projects, e.g. project being active less than 6 months or having less than 2 contributors.

For the survey, we collected up to three persons per project, targeting persons who had introduced UML models (1UC), persons who had updated UML models (UC) and persons who had not committed UML models (NUC), to send our survey to. As a result, we received 485 survey responses from 458 GitHub projects. Analyzing the responses allows us to answer the following main research questions:

- Why is UML used in OSS projects?
- Is UML part of the interaction of (a team of) contributors?
- What is impact/benefit of UML?

Findings from survey responses allow us to not only answer the research questions, but also to compare UML use and impacts of using UML in OSS projects and related empirical research. Furthermore, a number of recommendations/implications on the use of UML were given to different UML practitioners.

The main contributions of this paper are:

- [a] The Lindholmen dataset which is the biggest dataset of UML models and meta-data from OSS projects (goal **G1**).

- [b] A method for automatically curating a corpus of software modeling artifacts (goal **G1**).
- [c] Insights from a large scale survey of OSS developers that use UML on:
  - the modeling practices that were used in their projects (goal **G2**).
  - impacts of UML modeling to different aspects in their projects (goal **G3**).

#### 1.4.4 Paper D: An Automated Approach for Classifying Reverse-engineered and Forward-engineered UML Class Diagrams

Taking a closer look into the UML models in the Lindholmen dataset, we could be able to draw some observations on characteristics of the models. One important observation is that we have identified two main types of UML models:

- Forward design models: those models that are hand-made as part of the forward-looking development process; typical uses of these models are in communicating and guiding the design [53].
- Reverse engineered models: those models that are reverse engineered from the source code, hence the models follow the construction of the implementation and mostly serve as after-the-facts documentation [53].

As the two types of UML models are fundamentally different in the context in which they are used, we saw the need of curating the Lindholmen dataset by classifying UML models into forward-design or reverse-engineered models. In paper D, we address this problem by provided an automated classification model for classifying Forward Engineered Class Diagram (FwCD) and Reverse Engineered Class Diagram (RECD). To build the classifier, we used a machine learning approach. Figure 1.9, which is cloned from Figure 5.3, shows the steps that were taken.

Firstly, we scanned the Lindholmen dataset to collect a set of 999 UML class diagrams in various image formats. In the Lindholmen dataset, a project might have UML models in different formats than image formats, e.g. in `.xmi` and `.uml` files. Then, the manual labeling process was done by three UML experts who have at least five years using UML class diagrams for different purposes. Features for use in the classification were then extracted from the images by using a 2-step process: firstly, we used the Image2UML tool to extract UML models from the input images into `.xmi` files [54]; secondly, we built scripts to extract 16 features from the `.xmi` files.

We experimented with 12 well-known classification algorithms (such as Decision Table, Random Trees, Random Forest, etc.) and compared their classification performance (i.e. in terms of *correctness*, *precision*, *recall*, *F-Measure* and *AUC*) to obtain the best classifier. The best performing classification algorithm is Random Forest with an accuracy rate, F-Measure and AUC scores of 90.74, 0.94 and 0.96, respectively. As a final step, we use the classifier to classify all UML class diagrams in the Lindholmen dataset. The classifier was

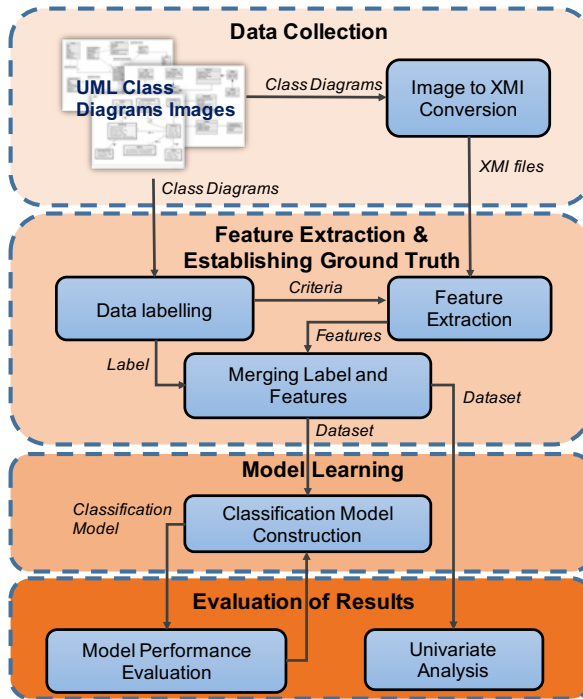


Figure 1.9: Overall processes of paper D

able to identify 10,845 FwCD and 9,821 RECD. The result was then annotated in the dataset.

The main contributions of this paper are the following:

- [a] A dataset with ground truth for classifying FwCD and RECD. The data is available in the replication package of this paper <sup>10</sup>.
- [b] Identification of features that can be used to classify FwCD and RECD diagrams.
- [c] A suitable machine learning algorithm for classifying FwCD and RECD.
- [d] A comparative analysis of the performance of various machine learning algorithms.

Overall, by applying the classification model presented in paper D, we are able to make a step further in curating FwCD and RECD in the Lindholmen dataset. With this, we proved that it is possible to build an automated “curator” with high accuracy by using features extracted from the UML model’s content. This is different from the curation reported in paper C in which UML models were curated by process data (about their frequency of use).

<sup>10</sup><http://models.cs.chalmers.se/oss/Downloads/SEAA2018/ReplicationPackage/>

### 1.4.5 Paper E: Challenges and Directions for a Community Infrastructure for Big Data-driven Research in Software Architecture

While much of the research in software architecture has been inspired by industrial experiences, little of the research was validated beyond individual case studies. At the same time, many scientific disciplines are currently harvesting fruits from large scale data collection about their subjects of study. Paper E contributes a discussion of challenges and directions for big-data driven studies of software architecture. In particular, in paper E, we discussed the following five challenges (see details in Section 6.4):

- C1:** Finding a common representation for software architectures.
- C2:** Capturing relevant context information.
- C3:** The high effort for crawling big-data.
- C4:** The High effort for curating.
- C5:** The need for collaboration in empirical software architecture research.

Given the large amount of effort that is needed for big-data driven studies of software architecture, a promising direction is to look into a community-based infrastructure for enabling and supporting this type of research. We proposed the following nine requirements/directions to building such infrastructure (see details in Section 6.5):

- R1:** Be able to host big & heterogeneous data of software architecture documentations (SAD).
- R2:** Share not only data but also software architecture knowledge and analysis.
- R3:** Enable links to contextual data.
- R4:** Support evolving artefacts and architectural knowledge.
- R5:** Keep annotations separately.
- R6:** Crowd-source annotations.
- R7:** Enable comparison & aggregation of research findings.
- R8:** Encourage discussion and peer-review.
- R9:** Promote collaborative research and enrich a collaboration network.

In this paper, we also shared lessons that we learned through the building of various tools. Indeed, these tools could form building-blocks in such an infrastructure. Based on these lessons, we synthesized a reference architecture for creating a community-wide infrastructure for big-data-based research in software architecture (called **CoSARI**). Figure 1.10 (which is cloned from Figure 6.2) shows the architecture of CoSARI. A detailed description of CoSARI can be found in Section 6.6 of Chapter 6.

The main contributions of paper E are:

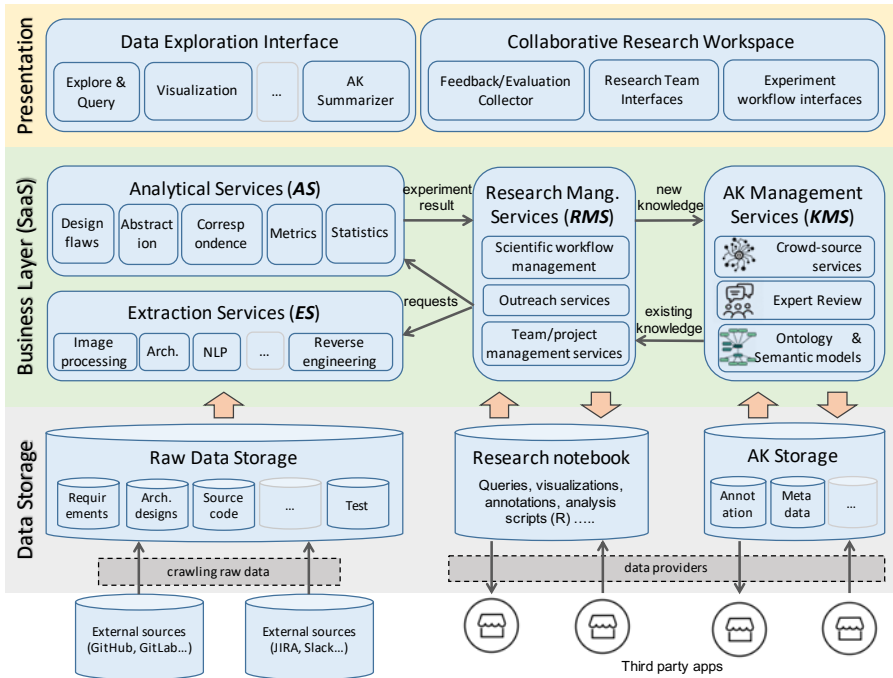


Figure 1.10: Architecture of CoSARI

- Lessons learned from building a big corpus of software models (i.e. the Lindholmen dataset).
- A discussion on challenges and directions for big-data driven studies in software architecture.
- CoSARI - A reference architecture of a community-based infrastructure for big-data driven studies in software architecture (goal **G1**)

### 1.4.6 Paper F: Does UML Modeling Associate with Higher Software Quality in Open-Source Software?

The benefits of modeling the design to improve the quality and maintainability of software systems have long been advocated. Yet, the empirical evidence on this remains scarce. In paper F, we fill this gap by empirically studying the relationship between UML modeling and software defect proneness in a large sample of OSS GitHub projects. We conducted a quasi-experiment to compare the defect proneness between two groups of OSS projects: a treatment group of 50 projects using UML models; and a control group of 93 projects sampled randomly using GHTorrent [45]. Figure 1.11 shows the steps that were taken.

In the treatment group, 50 projects were selected from the curated list of 4,650 projects reported in paper C (from the Lindholmen dataset) based on criteria such as the project's number of issues, the language (used in the description of issues), the project's GitHub stars, and the project time span. Applying the same criteria, we collected another set of 93 GitHub projects



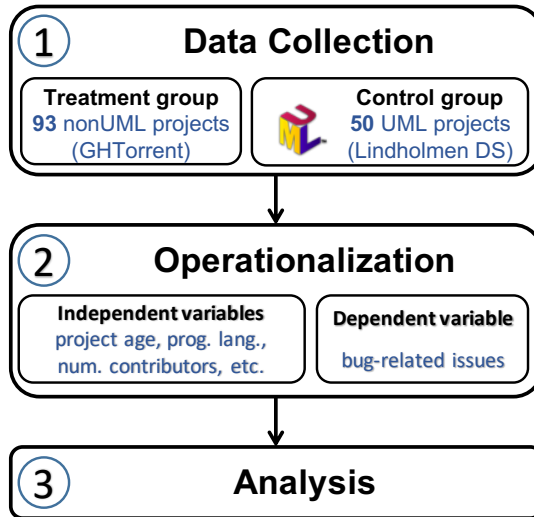


Figure 1.11: Overall process of paper F

(that are not part of the Lindholmen dataset) to form the control group by using GHTorrent. In these projects, our crawler and classifier together did not find files that represented UML designs of the system.

Next, we operationalized the variables for the study:

- i) Dependent variable being *number of bug-related issues*; and
- ii) Independent variables are project-level measures such as *project age*, *number of commits*, *number of contributors*, *number of GitHub stars*, and *has license*.

Finally, we built a multiple linear regression model to estimate the effects of having UML models on defect proneness. The result shows that projects in which UML models were found experience 35% fewer bugs reported, and thus have a lower defect proneness, than projects without UML models.

Overall, the main contributions of paper F are:

- [a] Empirical evidence on the relation between (UML-based) software modeling and an aspect of software quality, i.e. defect proneness (goal **G3**).
- [b] A method to measure defect proneness in OSS projects using textual description of issues.
- [c] Data and source code of the study are made available for other researchers to reuse <sup>11</sup>.

<sup>11</sup><https://github.com/adi1234567890/UML-defect-proneness>

### 1.4.7 Paper G: Using Machine Learning for Automated Classification of Class Responsibility Stereotypes in Software Design

The concept “Class role-stereotype” stems from Wirfs-Brock’s theory about responsibility driven software design. Role-stereotypes indicates generic responsibilities that software classes play in the design of software system. Wirfs-Brock identifies 6 of these role-stereotypes, namely: *controller*, *information holder*, *interfacer*, *structurer*, *service provider* and *coordinator*. Knowledge about class role-stereotypes can help in various tasks in software development and maintenance, such as program understanding, feature location and quality assurance [55–59]. Paper G presents an automated machine learning-based approach for classifying the role-stereotype of classes in Java projects. Figure 1.12 shows the steps that were taken.

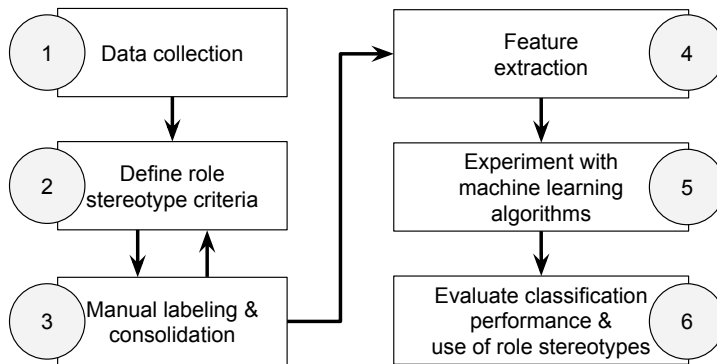


Figure 1.12: Overall process of Paper G

Firstly, we selected three open source software projects and collected their source code. Three experienced developers carefully manually labeled all classes of the three projects, which altogether are 1,547 classes. As a result, we obtained a ground truth of 1,547 classes labeled with corresponding role stereotypes. In the next step, we built a small application to extract 23 features from the the source code of these projects. These features were then used as input for building classification models. We evaluated the performance of the classification models in terms of *precision & recall*, *F1-score* and *MCC score*. The evaluation allowed us to draw conclusions about which classification models yield the best classification performance, and which features are most predictive for this purpose.

In summary, the main contributions of paper G are the following:

- [a] A machine learning method to address the problem of automatically inferring the class role-stereotype of a class based on its source code features.

The fact that the classifier works in an automated way enables the rapid labeling of role-stereotypes for large collections of classes, and in practice for large code-bases of entire systems. This, in turn, enables novel analyses that sheds light on the anatomy of software designs, such as for example

analysis of the frequency of particular collaboration patterns between different stereotypes.

- [b] An illustration of the potential uses of class role-stereotypes, i.e. in generating tailored thresholds for design metrics in design-smell detection and in profiling software system’s design intention and style.
- [c] The ground truth of 1,547 classes labelled with corresponding role stereotypes <sup>12</sup>.

### 1.4.8 Paper H: Interactive Role Stereotype-Based Visualization To Comprehend Software Architecture

Software visualization has long been recognized as a helpful tool for comprehending the architecture of large software systems. Software visualisation has a long traditions of representing various structural perspectives of software systems. In paper H we enriched this perspective by adding the notion of *class role-stereotype*. The role-stereotype of a class carries information about the responsibilities that a class plays in software systems as well as the types of collaborations that it typically has with other classes. We created an interactive visualization called *RoleViz* that visualizes system architectures in which architectural elements (e.g. class, interface) are coloured according to their role-stereotypes. Figure 1.13 shows an example of the visualisation.

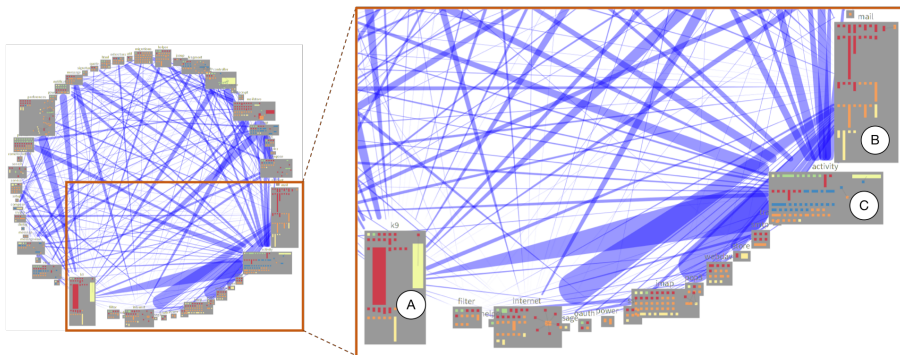


Figure 1.13: An overview of RoleViz visualisation

Subsequently, we conducted a user-study in which developers use RoleViz and Softagram (which is a well-known software architecture visualization tool used by industrial software developers and architects) to solve two software comprehension tasks on a large open source system. A comparative evaluation was done to answer the following two research questions:

- How does RoleViz compare to Softagram? In particular, we compare the two visualisation tools in terms of:
  - participant’s perceived cognitive load,
  - participant’s perceived usability,

<sup>12</sup>[http://models.cs.chalmers.se/oss/Downloads/JSS2019\\_SCAM\\_ReplicationPackage/](http://models.cs.chalmers.se/oss/Downloads/JSS2019_SCAM_ReplicationPackage/)

- participant’s understanding of the software system regarding the tasks.

By “understanding”, we refer to the participant’s ability to: a) locate components/entities of the system relevant to the tasks, b) describe the responsibility of the located components/entities and relationship between them, and c) formulate a plan to solve the tasks.

- What are the perceptions of the participants on the current features of RoleViz?

Determining whether RoleViz meets the expectation of the participants is crucial to identify where exactly RoleViz falls short of features. In addition, this research question helps formulating the future direction of RoleViz.

The main contributions of this paper are the following:

- [a] RoleViz - an innovative visualization tool that overlays roles on top of a software architecture.
- [b] A user study that assesses how RoleViz can help developers in real comprehension task, e.g. bug fixing. This study helps to reveal some impacts of software architecture visualisation and role-stereotype in comprehension tasks (goal **G3**).
- [c] A comparison of effectiveness between RoleViz and Softagram .

Overall, paper H provides a novel visualisation that assists developers in comprehending software systems. This, in the long run, can be combined with the classification tool reported in paper G to provide an automated presentation of architecture of software systems under the view of class role-stereotypes.

## 1.5 Summary of Research Findings

In this section, we combine the findings from the individual research study chapters to provide answers to the eight research questions of this Ph.D study.

### 1.5.1 Answer to RQ1. How to build a large corpus of models?

We created the Lindholmen dataset (available for public access and use at <http://models.cs.chalmers.se/oss/>) which contains so far 93,648 UML models from 24,797 GitHub repositories and meta-data of the repositories. Figure 1.14 shows the data schema of the Lindholmen dataset <sup>13</sup>. Creation of this big corpus comprises two steps: collecting UML models from GitHub and curating the data.

<sup>13</sup>A downloadable version is available at <http://models.cs.chalmers.se/oss>

**Data Collection.** The UML models in the Lindholmen dataset were collected from GitHub. We learned that UML diagrams are often stored in various image formats. Paper A presents a machine-learning-based method for automatically determining whether an arbitrary image represents a UML class diagram. Paper B demonstrates one approach to systematically identify UML models in a large part of GitHub projects (about 1.2 millions GitHub projects). This approach joins together different technologies, i.e. mining software repositories (provided by the research group of Gregorio Robles in Madrid), text-based search and the classification technique proposed in paper A. In the study presented in paper C, the data collection method described in paper B was applied to obtain UML models from all repositories in GitHub (around 12 million at that time). This collection method can in principle be used to automatically collect UML models from any large source code hosting/versioning system. We published the replication package of our approach online so that other researchers can replicate these steps (<http://models.cs.chalmers.se/oss/>). To our best knowledge, this is the first time such a large corpus is established and made available to the research community.

**Data curation.** For our project, data curation denotes the principled creation (including cleaning, completing), verification and enriching of data. Curation of the Lindholmen dataset was done in both manual and automated manners. Specifically, manual efforts were spent on verification (are the identified images indeed class diagrams) and on identifying UML sequence diagrams. Through manual effort we managed to identify 6,576 UML sequence diagrams in GitHub repositories. The field *uml\_type* of table *umlfiles* (in the Lindholmen dataset) is used to store information about types of the UML models (currently it can have 3 values: *Class*, *Sequence* and *Other*).

Some data curation can also be automated. Paper C demonstrates a way to curate a list of 4,650 software engineering projects. The 50 projects in Paper F are obtained by further filtering this list with more strict conditions in terms of project time-span, language and number of GitHub issues. These lists of projects were documented as part of the replication packages in the corresponding papers but were not annotated in the Lindholmen dataset (based on the annotation guidelines presented in Section 1.1.2). Paper D presents an automated approach to classify UML class diagrams into Forward (FwCD) and Reverse Engineered (RECD) diagrams. This way of creating the UML models is relevant for understanding their use and their impact in software projects. This data was used to enrich the Lindholmen database (it is represented in the meta-data by the field *isFwd* in the table *umlfiles*).

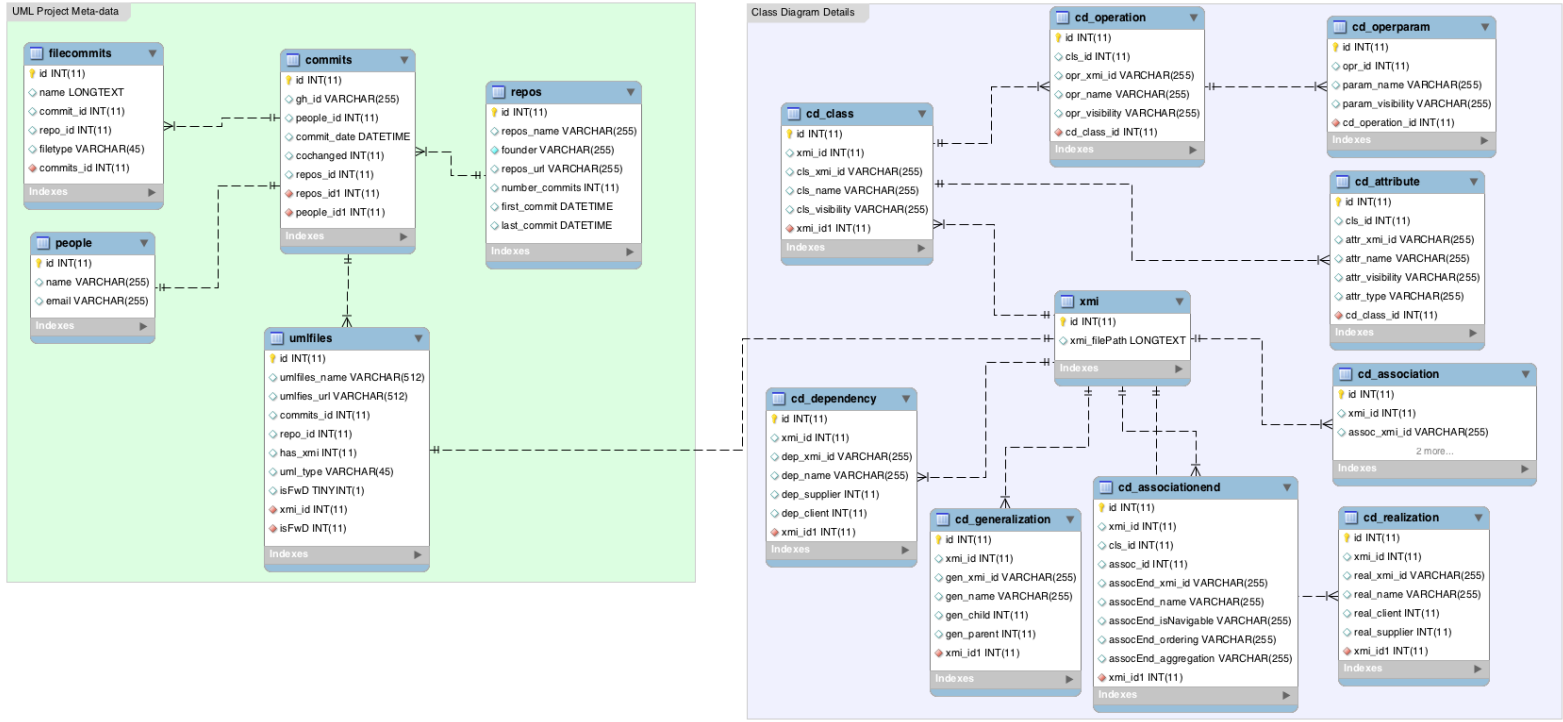


Figure 1.14: The data schema of the Lindholmen dataset

### 1.5.2 Answer to RQ2. How can we share and promote use of the corpus of UML models?

We have put significant efforts in sharing and promoting the Lindholmen dataset to various public audiences since September 2016. The first version of the our dataset was published as the main outcome of paper B at the 19th MODELS conference, targeting model-driven software and systems engineering community. The second version of the Lindholmen dataset (with 93k+ UML models) was shared in the Software Engineering In Practice (SEIP) track of the 39th ICSE conference as part of the paper C with a target to industrial audiences. We subsequently presented the dataset to various research communities, i.e. the Data Mining community and open source community (paper [f] and [d] in the “Other publications” list, respectively).

Since the introduction of the Lindholmen dataset, the dataset has been well received by researchers in many research communities. In particular, we regularly receive requests for using the data set. We have been actively supporting the researchers to effectively use the dataset for their research. To the best of our knowledge, 11 published studies (by authors other than the researchers that were involved in its creation) have used data extracted from the Lindholmen data set, i.e. Arora et al. [60], El Ahmar et al. [61,62], Kretschmer et al. [63], Schulze et al. [64], Ott et al. [65], de la Vega et al. [66], Babur et al. [67], Agt-Rickauer et al. [68], Torre [69], Baddreddin and Rahat [70]. We reported our experiences in building, sharing and promoting the use of the Lindholmen data set in paper E.

In paper E, we also show a vision on building a collaborative environment for big-data driven studies in software architecture (called CoSARI). The tools and methods used/produced as part of this Ph.D study can be used as building-blocks in this environment. In this vision, the application area of the tools and methods go beyond the scope of software modeling: the approach can be generalized/customized to support other types of software development artifacts. The paper also suggests and promotes an interesting empirical research direction in which knowledge is shared and built on top of empirical evidences and findings.

### 1.5.3 Answer to RQ3. What are purposes of using UML in OSS projects?

We ran a survey under 485 practitioners from 458 OSS projects (paper C) in order to answer research question **RQ3**. A summary of key findings is the following:

The majority of UML models are intended for creating software designs and documenting software systems. Non-UML contributors (NUCs) use UML models to comprehend a system and to communicate with team members. In most software projects that use UML models, these models are used for the following purposes: communication, making architecture decisions and for mentoring. UML models are adopted/implemented in most cases during the implementation phase. Most often, these models are implemented by a group of 2 - 5 persons.

### 1.5.4 Answer to RQ4. How is UML used in OSS projects?

In the study reported in paper B, by quantitatively analysing 21,316 UML models from 3,295 GitHub repositories, we were able to draw quantitative answers to RQ4:

Models are introduced during all possible phases in the lifespan of an OSS project. Indeed, a peak of introducing models is during the first 10% of the duration of the projects. A few projects are active with UML during their entire lifetime. However, most projects work very shortly on UML, usually at the beginning of the project. The majority of models is never updated. Those projects that do update their models do this regularly. Out of our entire corpus of 21,316 UML models, we found that 2,300 models (10.8% were duplicated. Half of the duplicates spread over more than one GitHub repositories.

Paper D contributes to answer this research question by applying a machine learning classification model to automatically classify FwCD and RECD in the Lindholmen dataset. With the machine learning model built in , we are able to identify 10,845 FwCD and 9,821 RECD.

### 1.5.5 Answer to RQ5. What are practices for using UML modeling in software development?

By studying UML use and impacts of using UML in OSS (paper B and C) and other settings (such as industry, from related work), we were able to see both shared and unique aspects of UML modeling practices. We split the answer to this research question into three parts, i.e. common practices, unique practices and our opinion on what can be learned from others contexts.

**Common practices.** We found a similarity in the ways of using of UML between OSS and industry: *UML is mainly used for design and documentation*, and much less so for code generation within OSS. Similar observations have been made for industrial usage by Torchiano et al. [32] and Forward et al. [33].

The finding that *UML is used for communication purposes* within OSS aligns with observations that were already made about the use of documentation by Kazman et al. [41] and about the use of sketches by Chung et al. [42]. This aligns with the insights of Gorschek et al. [34] and Hutchison et al. [14], who also observed a use for communication within industrial and OSS programmers.

The observation that *new contributors seem to benefit from the use of UML* confirms the first anecdotal evidence that Chung et al. collected [42]. Gorschek et al. found similar tendencies in their survey, where the use of models was found to be higher for novices [34].

**Unique UML modeling practices.** We found a hint of a contrast in the use of UML: We observed that *the architectures defined through UML models are often implemented by multiple developers*. This is similar between OSS and industry. We also observed that in most cases all these contributors had participated in the model creation. This seems to be in contrast to the practice in many industrial cases, where those who create the models are not necessarily the ones who create the code, as, e.g., observed by Kuhn et al. [36].



Besides, we made two observations on the OSS modeling practices for which we have not seen any industrial evidence. The first phenomenon is “*passive benefits*”: many of the participants of OSS projects who are not themselves involved in creating UML models, do consider the existence of UML models beneficial to the project. The second phenomenon being “partial adoption”: many models are only partially adopted during implementation. It would be interesting to see whether this conforms to or is in contrast to industrial practice.

We also note that there are industrial practices for which we cannot find evidence that they are also used in OSS projects. We further discuss these practices in the Discussion section.

**Implications.** By observing the practices of UML modeling in both the settings of open source projects and that of industrial projects, we were able to see the aspects where each side could learn from each other. In particular:

- To OSS practitioners: Use UML to coordinate team-work!
- To OSS seniors: Provide UML to support junior peers!
- To Industrial companies: Let’s adopt team-modeling!
- To University teachers: Promote the *consumption* (i.e. reading/understanding) of UML models as first experiences when learning UML (rather than creating them).

### 1.5.6 Answer to RQ6. What are perceived impacts of using UML in OSS projects?

For this question we focus on the perception of developers of open source projects. Using a survey (paper C) we collected responses from 485 practitioners in 458 OSS projects. From this survey we find out the following: UML is helpful for new contributors to get up to speed. However, UML does not seem to have the potential to attract new contributors. One third of the respondents reported changes of the working routine due to UML, mainly in the planning phase, the development process and in communication. Most of the reported changes can be considered positive.

### 1.5.7 Answer to RQ7. Does the use of UML modeling correlate with lower defect proneness?

In paper G, we found a small (approximately 2% of the variance explained by the model) but statistically significant effect of using UML models in software development: other variables held fixed, projects that use UML models have about 35% fewer bugs reported than projects without UML models.

### 1.5.8 Answer to RQ8. Does using class role stereotypes correlate with better understanding of designs of software system?

Paper G constructs a method for automatically identifying the design role of classes using a machine learning approach. Based on this method, we propose (in paper H) a software tool called RoleViz for creating interactive visualizations. RoleViz visualizes system architectures in which architectural elements are annotated with their role-stereotypes. We conducted a user-study in which developers use RoleViz and Softagram (a commercial tool for software architecture comprehension) to solve two separate comprehension tasks on a large open source system. We compared RoleViz against Softagram in terms of participant's: (i) perceived cognitive load, (ii) perceived usability, and (iii) understanding of the system. In total, 16 developers participated in our study. Six of the participants explicitly indicated that visualizing roles helped them complete the assigned tasks. The participants achieved better scores on completing software understanding tasks with RoleViz without any cognitive-load penalty.

## 1.6 Discussion

In this section, we extend our findings on UML modeling practices with common practices found in industrial contexts. Further, we discuss lesson learned from using a machine learning approach in various studies under this Ph.D thesis.

### 1.6.1 Software Modeling and Design Practices in Industry

In this section, we discuss and compare general practices in software development and the practices that are known from industrial software development practice. This discussion includes both practices that we have observed during this Ph.D study as well as practices reported in other studies. This discussion by no means provides a complete list of practices used in software design and modeling.

#### 1.6.1.1 Capturing design models

**Capture design discussions and decisions on a Wiki.** A project wiki can be a collection of Web pages that can be edited easily by anyone on your project using a Web browser. This practice implies to keep discussion and decisions related to design at one place for easy access and search as the project goes. This practice is especially useful if the development team is geographically distributed.

**Insert design documentation into the code itself.** This practice suggests to document key design decisions in code comments, typically in the file or class header. When this approach is coupled with a documentation extractor (such as JavaDoc), this assures that design documentation will be readily available to a programmer working on a section of code, and it improves

the chance that programmers will keep the design documentation reasonably up to date.

**Choices of formal and informal design notation.** While scanning images that might contain UML models in OSS GitHub projects (as part of the study in paper A, B and C), we recognized both formal (e.g. UML models) and informal design (e.g. sketches or whiteboard photos) documentation of some software systems. While informal notation gives more freedom of expressing design ideas, formal notation enables design decisions to be clearly communicated and formally verified. Petre et al. suggest the use of mix between formal and informal notation when modeling [71]. McConnell gives guidelines on choosing suitable formality of the models and level of details needed [18]. Figure 1.15 (copy of Table 5-2 of [18]) shows recommendations on suitable level of detail and design formality with regards specific settings of the design team.

Factor	Level of Detail Needed in Design Before Construction	Documentation Formality
Design/construction team has deep experience in applications area.	Low Detail	Low Formality
Design/construction team has deep experience but is inexperienced in the applications area.	Medium Detail	Medium Formality
Design/construction team is inexperienced.	Medium to High Detail	Low-Medium Formality
Design/construction team has moderate-to-high turnover.	Medium Detail	—
Application is safety-critical.	High Detail	High Formality
Application is mission-critical.	Medium Detail	Medium-High Formality
Project is small.	Low Detail	Low Formality
Project is large.	Medium Detail	Medium Formality
Software is expected to have a short lifetime (weeks or months).	Low Detail	Low Formality
Software is expected to have a long lifetime (months or years).	Medium Detail	Medium Formality

Figure 1.15: Design Formality and Level of Detail Needed. Source: Table 5-2, McConnell [18]

**Choose tools that fit purposes of modeling.** While browsing the Lindholmen data set, we observed that the UML models are developed by a large variety of UML-editor tools (e.g. StarUML, Papyrus, Enterprise Architect, Umple, PlantUML, etc.). Developers are now provided with a large and diverse set of modeling tools that support the development of models, their transformation, and their integration within the software development process. This practice suggests developers to choose the modeling tools which is of a good fit to the modeling purposes. It also implies that developers should avoid a

"habitual-, thus unconscious selection" of modeling tools. This is supported by many empirical studies in which the choice of modeling tools affect vastly the adoption of modeling within the studied software projects [7, 14, 32].

Some common questions developers can ask when choosing modeling tools:

- Drawing tool or modeling tool? Do developers need to only express design in a graphical presentation or to actually apply model-checking? In [72] Brambila et al. discuss the difference of the two type of tools in detail. Ossher et al. present FlexiTools which was developed with characteristics of both modeling tools and office tools for pre-requirement analysis' [73].
- Textual-based (e.g. Plantuml) or graphical-based (e.g. Papyrus)?
- Collaborative or single-user tool? - This question expresses the need of collaborating in modeling.

### 1.6.1.2 Modeling style

**Use guidelines for naming in diagrams.** In order to understand a software system and its design, one often needs to go back and forth between source code and design models of the system. Name matching is a technique quickly relate relevant parts of source code and design models. While many projects enforce naming conventions to source code elements, this is not the case for design elements. This fact might result in more efforts to understand the design of the software system.

This practice aims to apply naming conventions in models, thus improving model's maintainability and project's traceability. Given this is a low-effort (to implement) but high-impact (to increase maintainability of the system) practice, we would highly recommend projects to use (and possibly enforce) naming conventions in design models.

**Use guidelines for layout of diagrams.** Layout of UML diagrams has become a factor , e.g. good layout can help developers to better understand the design of the system [74]. This practice aims to support modelers to create a models that are easier to understand and easier to maintain. We recommend that a project defines and establishes among project members a guidelines for layouting a diagram. Example of such guidelines can be found at <http://agilemodeling.com/style/general.htm>.

**Create models in different abstraction levels.** In a software project, design models can be consumed by many people for very different purposes. To facilitate this, a common practice is to create models in different levels of abstraction, e.g. with different level of details. Petre et al. also mentions this practice as a way for expert modelers to capture both high-level as well as important low-level details [71].

### 1.6.1.3 Updating design/models

**Update the design to reflect the implementation** A very common pattern in software development, is that programmers prioritize coding (i.e.

producing source code) and postpone updating the design/documentation. If this practice sustains and a team neglects to update design/documentation, then this leads to increasingly large disparity between the actual implementation and the design/documentation. This fairly quickly makes the design/documentation of little value to the team and this in turn reduces the motivation to update the design/documentation.

As a practice we recommend that a project defines as part of their working methods who is responsible for updating (when) to use a 'definition of done' for (updating) a model.

**Models should be kept under version control.** Models, as other software development resources, are subject to change as the project progresses. While developers seem to pay much attention to capture all updates in source code, less efforts seems to be spent on capture changes in models. We would argue that capturing models change is as important as capturing code change. One of the important reasons is for all team member to easily see what is the latest version of the design/model.

As a practice, we recommend projects to consider a modeling tool in which model changes can be captured by the versioning system.

#### 1.6.1.4 Apply Quality Assurance to Design/Model

Quality Assurance (QA) is a common (best) practice in software development. Quality assurance applies to all steps and artefacts in software development. Several of the practices that are discussed in this section are commonly applied to source code. For unknown reasons these practices are hardly ever applied to models/design. It seems a matter of including some steps in the development process and appointing responsible persons for enforcing the QA-policy also to models/designs. Of course, teams must respect the quality-processes and management must also support them.

Typical steps of the QA for designs could include:

- Establish standards for tools, layout, naming of models of designs
- Define *when* designs should be updated and *by whom*
- Review a design for fitness for purpose (esp. communication).
- Monitor that the implementation follows the design.

**Store models in editable format.** About half of models in the Lindholmen dataset are stored in images which are not editable. This makes it hard to access models content, and almost impossible to make changes on the models. This practice motivates developers to generate and store their models in an editable form, thus improving the accessibility and maintainability of the models. This, in the long term, will benefit the software project in many ways, e.g. it enables model update and automated compliance check between code and design.

**Archive design/modeling works.** This practice suggests developers to document their models for reference and reuse purpose. McConnell calls this practice as "Save design flip charts" [18]. This also implies that models should not be stored alone but also with information about the context in which the models are used. There are many tools that support this practice via auto-generation of design document from multiple software development sources (such as Javadoc). However, the question of how to effectively archive design knowledge happens outside of the computer (e.g. sketches on whiteboard, verbal discussion about design) still remain.

## 1.6.2 Alternatives of the Machine Learning Approach

Machine learning has been utilised in this Ph.D study as the main method for building automated classifiers for various data collection and data analytic tasks. For example, in the study described in paper A, machine learning was used to build a classifier for detecting images that contains UML class diagrams. In the study described in paper G, the machine learning approach was used to automatically referring the role-stereotypes of a given class. In this section, we discuss alternatives for the machine learning approach and the reason why they were not chosen.

**Alternatives of machine learning approach.** From the last decade, neural networks or deep learning have been regarded as powerful tools for dealing with mining big-data. These have been shown to outperform traditional machine learning approaches in many situations. However, we choose to use machine learning over neural network/deep learning for some reasons:

Firstly, a deep learning solution often requires significant amount of computation resources and training data. These are not always available for this Ph.D study.

Secondly, deep learning methods often lack interpretability due to the sophistication of the neural networks. In particular, deep learning methods lack the ability to explain why a result is produced in a sensible way. This, to this Ph.D study would be a big threat to the construct validity. Machine learning approaches have been developed for a long time together with powerful evaluation tools (e.g. infoGain, AUC, confusion matrix). This enables researchers the ability to understand and making adjustments on the machine learning settings to achieve a higher result.

**Parameter tuning.** Researchers have recently shown that parameter tuning applied to traditional machine learning approaches can have significant impacts in the context of software engineering data [75]. In this research, the main effort regarding improving the performance of the machine learning algorithms has been spent mostly in enhancing feature selection and increasing the quality of the training/testing data. At the sametime, less effort was spent on adjusting the parameters of the machine learning algorithms. This might partly be due to the fact that there are very few guidelines on to what and how to adjust the parameters to achieve better classification performance. In the future, we shall apply some existing methods for automatically optimizing the paramters of the machine learning algorithms.

## 1.7 Threats to validity

In this section, we give an overview of the threats to the validity of the results of this thesis, i.e. the answers to research question **RQ1** to **RQ8**. Detailed threats to validity of included publications are discussed in their corresponding chapters, from Chapter 2 to Chapter 9.

Research question **RQ1** aims to establish a process of building a large corpus of UML models from OSS repositories. We use a constructive method to build the corpus. There were a number of threats to construct validity that might cause the loss of UML files when performing the data collection process. First, our collection method, which made use of a number of heuristic filters, might overlook potential UML files which are not complying with searching terms and file-type list. Second, limitations of the materials that were used to collect data could probably cause the loss of potential UML models (false-negatives) or the inclusion of files that do not actually contain UML (false-positives) Examples of such limitations are: out-dated GHTorrent SQL dump, incorrect detection of UML images and the limit of 5000 hits/hour of GitHub API. We partly mitigate these threats by performing manual checking on the validity of the classification to the process.

Research question **RQ2** aims at exploring a method to share and promote the use of the Lindholmen data set. In paper E, we discussed a conceptual solution for the problem. In particular, we proposed a reference architecture *CoSARI* in which the Lindholmen data set and the tools produced during this Ph.D study might serve as building blocks. There might exist other solutions for utilizing the data set that we were not aware of. Besides, the proposed reference architecture (COSARI) lacks proof of concept and carries the risks associated with being oversimplified and/or incomplete. However, we consider these types of risk are acceptable at the conceptual stage. Paper E discusses concrete future works which would hopefully lead to a more comprehensive solution, thus mitigating the risks.

Research questions **RQ3** and **RQ6** aim at revealing purposes and impact of using UML modeling in OSS projects. Answers to these questions are drawn from survey responses of 485 developers in 458 different GitHub projects that use UML (paper C). There is threats to the internal validity of the answers. That is, we focused on projects that do use UML only to ensure that questioned developers have the experience of working in a project with UML. To ensure those persons that prefer to not use UML are not underrepresented, we sent the questionnaire not just to persons who either made of modified UML, but also to contributors who did not create or change any UML files (NUCs). Therefore, we believe that our findings provide valuable insights.

Research question **RQ4** is intended to achieve understanding of the way UML is used in OSS projects. Answers to **RQ4** are drawn from a quantitative analysis of a set of 21,316 UML models from 3,295 (from 10% of all GitHub repositories). There are threats that could affect to the construct validity of this answer. In particular, the loss of potential UML files might affect to this analysis in the sense that it could make us underestimate the number of projects with UML models and the number of UML models. Being aware about this, we focus on getting a descriptive overview of various aspects of the use of UML in GitHub projects and avoid giving statistical conclusions. We expect

no systematic bias concerning the aspects that we investigated.

Research question **RQ5** aims to provide practices of using UML modeling in software development. The answer to this research question is built on understanding about how UML modeling is used in OSS (*RQ3* and *RQ4*) and in other settings (e.g. industry, from related work). We could, therefore, see two sources of threats to validity. On the one hand, with regards to the practices found in OSS projects, we could expect threats to validity from answers to *RQ3* and *RQ4*. However, given these threats were mitigated (as explained above), we believe that our findings in OSS projects are valid and significant. On the other hand, with regards the modeling practices found in various studies in industrial contexts, we expect some sorts of threats to external validity. Specifically, as the practices were found in different studies with unique settings of UML use, it is difficult to deliberately generalize the practices from one to other cases. Therefore, in the answer to **RQ5**, we report on the existence of common and unique practices and avoid to make comparison in different settings.

In research question **RQ7**, we aim at finding effects of UML modeling by studying the correlation with a software quality aspect: defect proneness. The answer to **RQ7** is derived from comparing two groups of projects: 50 projects that use UML and 93 another projects where there is no UML. There exists small threats to internal- and construct validity. We discussed the threats in detailed in section 8.9 of paper G.

The research question **RQ8** aims at understanding the impacts of using class role-stereotype in assisting software developers in comprehending the design of software systems. The answer to this research question is based on a comparative user study with 16 developer participants. Threats to validity of the study associate mostly with the selection of participants and the baseline-visualisation. We discussed the threats and the mitigation strategy in detail in section 9.8 of paper H.

Finally, answers to all research questions have threats to external validity. In particular, data in this research was only taken from GitHub, but not other OSS hosts/platforms such as SourceForge, Google Code, etc. As they differ from each other in terms of size, functionality, users and user's behaviors, there is a threat that answers to questions **RQ1 to RQ8** might not be generalized to other platforms. It is possible that UML is used in a different manner within projects at other platforms. However, as GitHub is one of the biggest player in the field, we strongly believe that our investigation gives valuable insights to a majority of the OSS community.

## 1.8 Future Work

In this section we discuss future work. In particular, sections 1.8.1, 1.8.2 and 1.8.3, are oriented towards possible extensions of the findings of this Ph.D study. We suggest future research directions that now become possible with our data set in section 1.8.4.

### 1.8.1 Extending the Lindholmen data set

In this section, we present our views on what the Lindholmen data set should grow into. Our views are pretty much driven towards the direction of building



CoSARI (presented in paper E). Accordingly, enrichment of the Lindholmen data set can be done by either increasing its quantity (e.g. via adding more models) and/or quality (e.g. via curating the existing data set). We discuss some directions in more detail next.

**Adding more models and design documents.** Currently, the Lindholmen data set contains UML models stored in what we found to be the most common formats: .uml, .xmi, and images formats (.jpg, .png, .bmp). In general, UML models might be stored in many other formats, e.g. formats that are specific for UML-editors such as .plantuml, .argo, .dia and .ump. One way to extend our corpus is to include more of such formats. In our approach we have looked for individual files that represent UML models. Another direction to look for more UML models is to search in files that may also contain other information, but also contain UML models, such as: Word (doc(x)), PDF, HTML, PowerPoint (ppt), among others. To capture these types of models and design documents, the data collection process (described in paper B) needs to be extended to accept more tool-specific formats and document-specific formats. This will require the creation of new tools/techniques that automatically extract models and/or design knowledge from such formats.

Similarly, software models that are not conforming to the UML standard can be added to the dataset. These can be models that are members of the UML-like family of languages, such as SysML, Capella models, or models from other model-based approaches such as Simulink.

**Adding models from industrial cases.** Currently, the Lindholmen data set contains UML models from OSS projects. While there exist some open source projects that are driven by companies, we could not identify these in our dataset. In the future, the Lindholmen data set could be extended by involving more industrial cases. Given that many companies are using Git as their versioning system, the technical process of collecting data from company's Git resources can be done in the same manner as our approach does for GitHub. The main challenge here will be to convince companies to share their designs publicly. Possibly governments could play an exemplary role here. Studying these cases will allow us to compare how UML is used in the settings of OSS and industrial cultures.

**Adding more software development artifacts.** One main reason why we collected UML class diagrams is that they are a commonly used representation for software architecture and designs. However, software designs may be represented in other notations, and it could be interesting to compare these and their effectiveness in software projects.

Also, besides models and design documents, various software development artifacts can be collected so as to understand the contexts in which software modeling practices are used. Currently, the Lindholmen data set contains only descriptive (meta-)data of the projects in which UML models were found. Other software development artifacts such as source code, issues, mailing list, wiki documents could also be collected. Clearly, this leads to extra efforts on collecting and curating relevant data outside the Lindholmen dataset. For

example, in paper F, we collected "issues" directly from GitHub via the GitHub API as a means for operationalizing the defect-proneness of software projects.

A couple of foreseen challenges towards collecting various software development artifacts are: i) Crawling big-data is technically not always an easy task given limited computation-resources, and ii) Many of the interesting artifacts exist outside of GitHub repositories. Therefore, collecting them requires extra efforts on building tools and cleaning noisy data, iii) including more types of artefacts further increases the number of file representations/formats that needs to be supported. For these issues, we specifically call for joint efforts of multiple research teams. One way to collaborate could be where each team takes the responsibility to collect and maintain one (or two) types of artifact(s) in the corpus.

**More and more data curation.** Data curation is an important activity to maintain the currency of the metadata and to make data better accessible, easier to find, more descriptive, and more relevant. Data curation becomes even more critical given the expected increase in the amounts of data in the future. By conducting this Ph.D study, we learned that it is possible to curate the dataset in both manual and automated ways. While manual curation approaches are often time-consuming, automated curation approaches require careful validation to ensure adequate accuracy. In the future, we can explore hybrid approaches in which knowledge about objects to be curated can be used to improve the performance of the automated curation [76,77]. For example, we can probably increase the performance of the classification models presented in paper D by using the knowledge about the performance of the classification features.

Besides, data curation can benefit from adding annotations to the data set. In particular, annotations can be made at project- and model-levels. For example, at the project level, annotations about "project license", "business domain", the "goals of project when using models" (for design or documentation), "general impacts of using UML" can be employed. At the model-level, annotations on layout-style of the UML model, tool that was used to generate the model, general role of the model, quality of the model, etc. could be very beneficial.

### 1.8.2 Extending the understanding about UML use and impact: enablers, inhibitors and context

Within this Ph.D thesis, we performed quantitative and qualitative analyses on the use of UML and impact of using UML in GitHub projects. As a step further, we could aim at eliciting enablers and inhibitors of the impacts of using UML. Toward this goal, we would think of several research possibilities as follows.

**UML use.** Paper B and C discover some patterns of using UML as well as the developer's rationale behind the UML use. These findings can be extended in several ways. One way is to conduct follow-up interviews to gain indeed understanding about specific survey response(s). For example, it is interesting to know the communication methods in which UML models were discussed, and the CASE tools that were used.

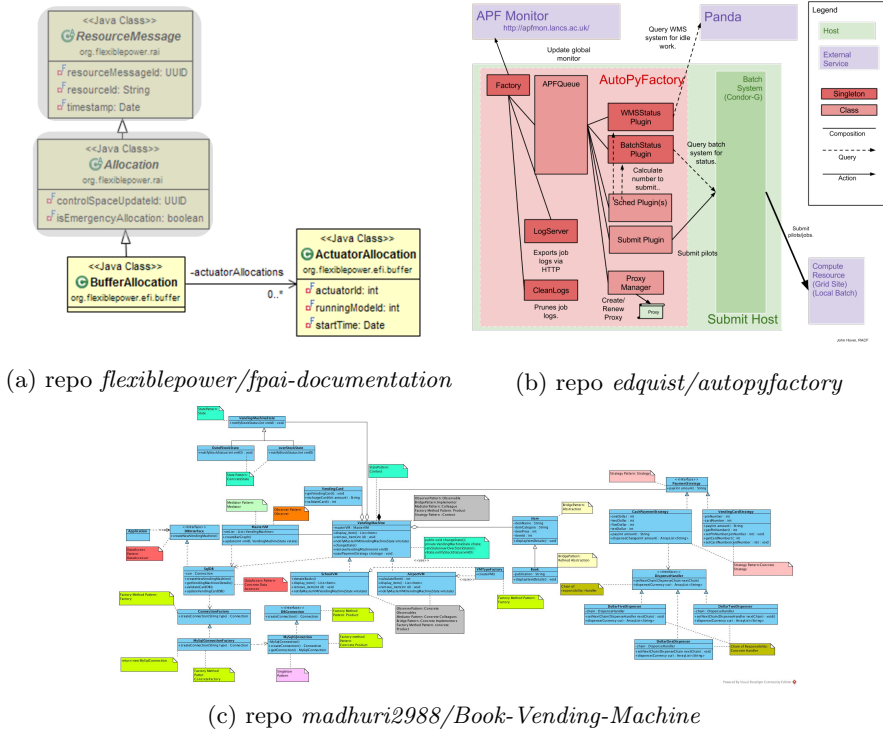


Figure 1.16: Examples of highlighted UML diagrams - Source: GitHub repos

The other extension is to "learn" characteristics of UML models that were successfully used in OSS projects. For example, what model layouts did developers use and what is the average size that models have.

In addition, studying UML that occurs in images can also provide hints on needs that OSS developers have for visual highlighting strategies. In particular, during manual inspection of the image set, we have seen quite many diagrams that were colored and used highlighting in various ways. Those diagrams might possibly be "important ones" as well. Figure 1.16 shows examples of three UML diagrams from three GitHub projects that use highlighting.

Moreover, due to the availability of UML models (and projects that they belong to), we can be able to study how UML models and other artifacts (such as source code, bug-reports, issues, etc.) relate to each other. For example, we know from the survey responses in paper C that UML models are (partly) adopted during the implementation phase but we do not know how strictly models are implemented, or in what way models abstract from the implementation code.

**Assessment of quality of UML models.** One aspect that can affect UML use/adoption is the quality of UML models. This is the case in one of the surveyed projects in Paper C. The founder of the GitHub project answered "I feel that it depends on (...) how elegantly and interesting the models was structured" as a reason(s) why UML models attracted new contributors to his/her project. We see a need for evaluating the quality of UML models and studying its impacts on the use of UML in software projects.

Evaluating quality of UML models could allow us to classify UML models by different quality aspects. Furthermore, knowing the quality of a UML model, we will be able to provide recommendations on how to improve the model. At the project level, understanding the quality of UML models of a project could enable to identify the need for actions for quality improvement.

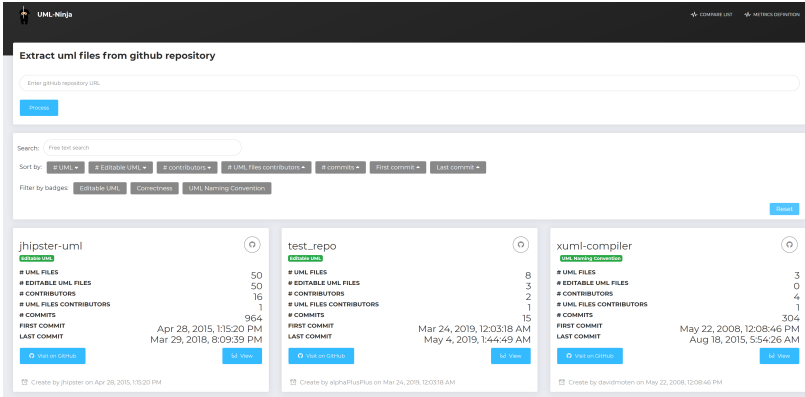
One way to evaluate quality of UML models is to use quality models such as [78–82]. Figure 1.17 shows the relationship between characteristics of quality of UML models and various software metrics and rules proposed by Lange et al. [78]. For example, communicativeness of a UML model is affected by the depth of inheritance tree (DIT) of the UML model.

Metrics and Rules	Modularity	Complexity	Completeness	Consistency	Communicativeness	Self-Descriptiveness	Detailedness	Balance	Conciseness	Esthetics	Correspondence
<b>Dynamicity</b>		✓	✓					✓			
<b>Ratios</b>	✓		✓				✓	✓	✓		
<b>DIT</b>	✓	✓			✓		✓	✓	✓		
<b>Coupling</b>	✓						✓				
<b>Cohesion</b>	✓	✓					✓				
<b>Class Complexity</b>							✓				
<b>NCU</b>	✓	✓					✓	✓			
<b>NUC</b>	✓	✓					✓	✓			
<b>Fan-In</b>	✓						✓				
<b>Fan-Out</b>	✓						✓				
<b>Naming Conventions</b>						✓	✓				
<b>Design Patterns</b>	✓		✓			✓	✓	✓			
<b>Layout-Guidelines</b>							✓			✓	
<b>Multi defs.</b>				✓			✓				
<b>ID Coverage</b>			✓				✓				
<b>Message needs Method</b>			✓	✓			✓				
<b>Code Matching</b>			✓				✓				✓
<b>Comment</b>						✓	✓				

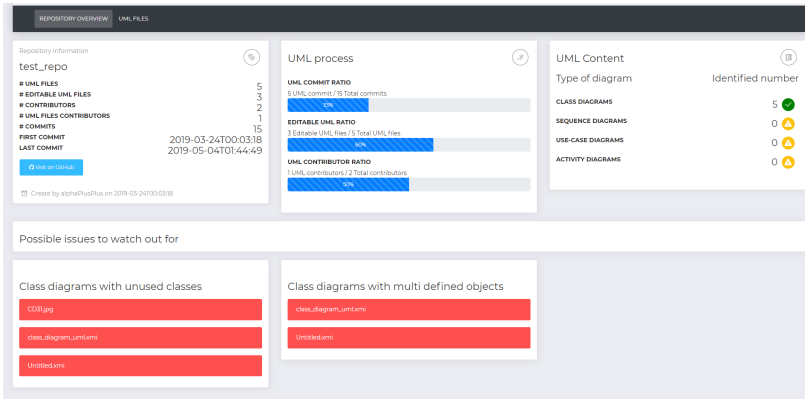
Figure 1.17: Mapping of model quality characteristics and software metrics. Source: Table 5, paper [78]

In this direction of research, we have recently built a tool named Ninja UML to automatically assess different quality aspects of UML models in any OSS project [83]. Figure 1.18 shows the GUIs of NinjaUML tool. The tool expects a URL to a GitHub repository as the input to run. Subsequently, the repository is downloaded and scanned for UML models in various image and textual formats. NinjaUML inherits tools and scripts created in this Ph.D thesis for identifying and crawling UML model files from GitHub.

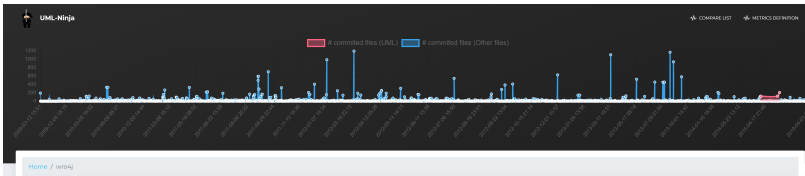
Figure 1.18a shows the list of all repositories that were processed. Figure 1.18b shows details when entering a repository of NinjaUML, i.e. a box showing basic information about the repository, a box showing metrics about UML process such as “UML contributor ratio metrics”, a repository commit history chart (Figure 1.18d). Figure 1.18d shows the metrics that are extracted out of a class diagram.



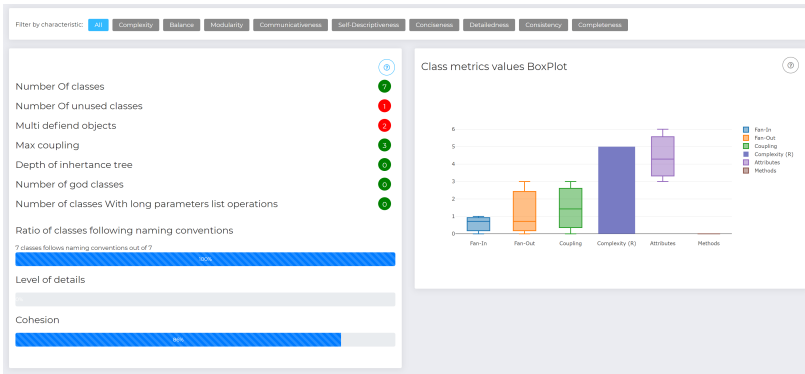
(a) Repository list page



(b) Repository page



(c) Repository commit history chart

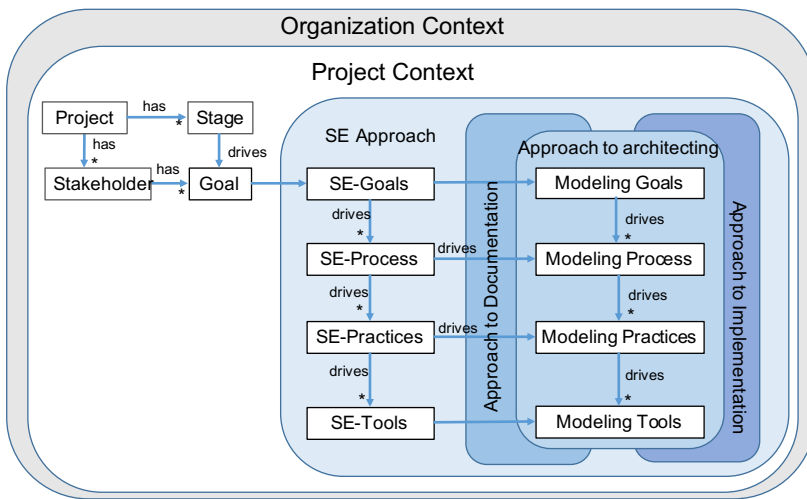


(d) Class diagram metric views

Figure 1.18: GUIs of NinjaUML tool

**Software modeling in context.** Fig 1.19 presents the complex nested contexts that influence the goals of modeling and thereby the various processes, practices and tools used. This figure illustrates that there is a hierarchy of contexts that influence how software practices are used. (generalized from [84]). There are organizational and project factors such as the goals of the stakeholders. For example, stakeholders may prioritize delivery date over quality of the software. Such priorities in turn affect the ways in which modeling is done. In particular they will affect the goals of doing modeling and via this also the processes, practices and tools used for modeling. In the future, for a true understanding of the value of modeling practices, efforts should be spent on studying all these context factors.

Figure 1.19: Impacts of contexts to software modeling approach



### 1.8.3 Building guidelines for UML use

In paper C, we provide implications for UML use, targeting software practitioners and educators. In the future, we aim at building guidelines which have the following properties:

**Content.** Guidelines will be provided to specific subjects based on two sources of understanding: i) understanding about common practices of UML use/adoption observed from a large number of software projects; and ii) understanding about the subjects (individuals or teams/companies) in terms of current use of UML and expected level of UML adoption. The guidelines are expected to cover:

- Methods to assess current status of UML use within a software development team/project. This will be provided in form of a check list or an analytical tool;
- Common mistakes on applying UML;
- Success stories of applying UML in a software projects;

**Characteristics.** The guidelines should be *actionable*. Each guideline should contain the following parts: i) Targeted subjects (individuals or teams/companies); ii) Targeted context and problem; iii) Expected outcomes; iv) Guideline in action; v) Common mistakes; vi) Examples from real-life projects.

Second, guidelines should be *evidence-based*. All guidelines can be traced back to empirical studies and real-life software projects that support them.

#### 1.8.4 Other directions

Our dataset comprises a large number of UML models and meta-data of the projects that they belong to. This dataset is expected to be a valuable source for empirical research in the field, such as design–code traceability, software quality assurance, etc. Below, we present two general cases where our dataset could be useful:

**Evaluation of scientific approaches and modeling tools.** Constructive research on software modeling often has the problem that there are not enough real cases of models to evaluate newly developed approaches and techniques. Currently, this limitation is worked around on the basis of toy examples or artificially generated models. In exceptional cases, researchers are allowed to use obfuscated industrial models or models created with the help of practitioners for the purpose of the evaluation [85]. Our dataset provides real cases of UML models in machine readable form. Professional tool vendors, who provide case tools for modeling, might be able to use the dataset to test new features on real data. One example of a study could be layout generation of diagrams.

**UML for education.** Software modeling and UML have been taught at universities in various courses, ranging from programming courses to analysis & design courses and software architecture courses. Novice software designers or students struggle with different problems during their training tasks [86–88]. A recent research by Karasneh et al. shows that access to a corpus of UML modeling examples helps students who are novices with UML modeling to create better designs [89]. This is aligned with our implication for university teachers in paper C, that teachers should promote students to consume UML models before creating models.





# Chapter 2

## Paper A

**Automatic Classification of UML Class Diagrams from Images**

**T. Ho-Quang, M.R.V. Chaudron, I. Samúelsson, J. Hjaltason, B. Karasneh, H. Osman**

*21st Asia-Pacific Software Engineering Conference (APSEC 2014),  
Jeju, Korea, December 1 - December 4, 2014.*



## Abstract

Graphical modelling of various aspects of software and systems is a common part of software development. UML is the de-facto standard for various types of software models. To be able to research UML, academia needs to have a corpus of UML models. For building such a database, an automated system that has the ability to classify UML class diagram images would be very beneficial, since a large portion of UML class diagrams (UML CDs) is available as images on the Internet. In this study, we propose 23 image-features and investigate the use of these features for the purpose of classifying UML CD images. We analyse the performance of the features and assess their contribution based on their Information Gain Attribute Evaluation scores. We study specificity and sensitivity scores of six classification algorithms on a set of 1300 images. We found that 19 out of 23 introduced features can be considered as influential predictors for classifying UML CD images. Through the six algorithms, the prediction rate achieves nearly 96% correctness for UML-CD and 91% of correctness for non-UML CD.

**Keywords:** Software Engineering; UML; UML class diagram; classification; machine learning; feature extraction.

## 2.1 Introduction

In software development, UML class diagrams (CD) are used to design and illustrate the structure of software. They are a very important tool when engineers need to understand the basic structure of a system, e.g. when a new engineer, that is unfamiliar with a system, needs to maintain it. UML CDs are becoming ever more prevalent within industry and academia, where model-driven development is becoming a common practice, and it is widely agreed that they have become an integral part [90, 91]. Accordingly, studying UML models and sharing of modeling artefacts [92] is an emerging need in recent years. In order to facilitate this need, a set of UML models should be collected in some forms of repository. Recently, both commercial UML repositories [93, 94] as well as general model repositories [25] have been built. B. Karasneh et.al [26, 54, 95] proposed an automated system (named *Img2UML*), which has the ability to extract UML CDs from images and share these via an online repository.

Among these repositories, enriching the one in [54] is easier, because a large portion of UML CDs is available as images on the Internet. However, the problem is that the automated collection of images needs a classifier to identify which image is related or not. We consider two scenarios where the classifier could be very useful:

- Users want to share their UML diagrams in image formats to the repository; and
- Automatic collection of images from various online sources into the repository. We think about several types of sources: image crawler (e.g. Google Image Search); shared image sets (from academia), etc.

Creation of such a classifier will bring a significant opportunity to automate the repository's collection phase. That is our main motivation to conduct this research.

**Research Problem.** This paper specifically aims at providing suitable features and classification algorithms to decide which images should be considered as UML CDs and which images should be left out. The classifier operates by extracting relevant information about the image and processing that information with a machine learner. The classifier is expected to have ability of inclusion of UML CD images and exclusion of non-UML CD images. Among these tasks, eliminating non-UML CD images has greater value than including UML CD images.

Since the input is images, information regarding UML CD that helps classifying the images needs to be discovered through image processing. This paper focusses on using basic image processing features as predictors (input variables used by the classification algorithm). The advantage of using the features is that these can be obtained very fast with little effort. This fits our objective of creating a fast method that will be of practical use to automated classification system.

We analyse the predictive power of the features to discover the influence of individual feature on the performance of the classifier. On the other hand,

to find the most suitable set of features, we prepare some sets of features and evaluating their classification performance.

In addition, with the aim at finding the most suitable classification algorithm, we make a comparison between candidate algorithms based on their classification performance. Costa et.al [96] investigated a range of measures that can be used for evaluating classification performance. In this study, the measures are specifically related to algorithms' ability to eliminate non-UML CDs.

**Contribution.** The contributions of this study are as below:

- Proposal of a set of features for UML CD inclusion/exclusion prediction. It consists of 23 features formulated from the image processing properties. The performance of the features is considered as their importance to the classifier. The suitability of four subsets of features is discovered as well.
- Evaluation on classification performance of six algorithms in terms of classifying accuracy and robustness. Candidate set of algorithms includes: *J48 Decision Tree*, *Logistic Regression*, *Decision Tables*, *Random Forests* and *SVM*, and *REP Tree*.
- Our dataset including images together with the list of extracted features are freely provided in order for researchers to test and make comparisons.

The remainder of this paper is structured as follow: Section 2.2 discusses the related research and section 2.3 indicates research questions. Section 2.4 explains the approach while section 2.5 describes the experiment. We present the analysis of results in section 2.6. Section 2.7 discusses our findings and section 2.8 ends with conclusions and future work.

## 2.2 Related Work

Recognition of special types of graphics is an area of intensive research. A survey of diagrams recognition and treatment can be found in [97]. UML CDs are a type of diagram that graphically represents classes and their relationships to one another. The majority of existing approaches to UML diagram image classification and understanding were developed within the scenario of image feature extraction. This section is aimed at discussing prior research on the topic, with focus on the fields of image classification and UML diagram feature extraction.

### 2.2.1 Image classification

Image classification refers to the labelling of images into one of a number of predefined categories. D. Lu et.al. [98] introduced major steps for image classification process. The steps may include 1) Selection of training samples; 2) Image pre-processing; 3) feature extraction; 4) Selection of suitable classification approaches and 5) Classification performance assessment. In this study, we follow these steps to build our classifier.

Much research has been done in this field, especially for classifying remote-sensing images [99]. With regards to diagrams, chart image classification seems to be one of the most concerned topic [100]. However, until now (to the best of our knowledge) there is no study about classifying UML class diagram images.

## 2.2.2 Diagram feature extraction

Recently, research has been conducted in this field of study, varying in method and approach. B. Messmer et.al. [101] proposed a system for recognizing and automatic learning of sketched graphic symbols in engineering drawings. The objective of this research is to combine pattern recognition techniques with machine learning concepts in order to be able to learn and recognize new symbols in engineering diagrams. In [102–104], a range of methods for online recognition of entirely hand drawn UML diagrams were introduced. However, since the methods were used information regarding the movement of drawing, which are not available in images, sketching tools cannot be carried over to recognizing UML models in final/static images.

L. Fu et.al. [105] presented a method for converting image based engineering diagrams (including UML models) into attributed graphs which can be used for content-based retrieval.

B. Karasneh et.al. [95] proposed a tool to extract class diagrams from computer-generated images. The tool recognizes UML class diagram properties and translates them into XMI format. Geometric-based feature as well as texture features were detected.

## 2.3 Research Questions

This section describes our main research question and three sub-questions. The main question of this research is as follows: How can classification of UML class diagram images be automated?

In order to answer this question, these sub-questions need to be figured out:

**RQ1.** What is the performance of image processing features in predicting the presence of UML CD?

**RQ2.** What is the performance of the classification algorithms in using the features as predictors?

**RQ3.** Which subset of the proposed features performs the best in classifying UML CDs?

## 2.4 Approach

In this section, we describe our approach in conducting this experiment.

### 2.4.1 Overall framework

The overall framework of this experiment is shown in Figure 2.1. To achieve the classifier, we use a machine learning approach.

Input for the process are images (Step 1). The images are then processed by applying a number of sub processes (Step 2) which can be listed as: Recognising

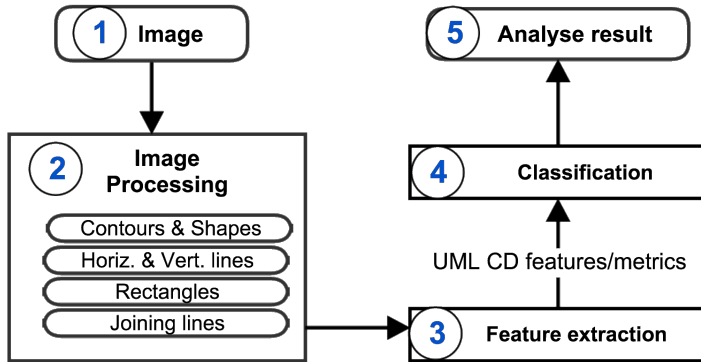


Figure 2.1: Overall framework

contours and shapes; Recognising lines; and joining lines in form of UML connections. Additionally, to avoid prolonged processing time on complex photographs, images have to pass a pre-check before being processed. Section 2.4.2 describes the process in details. The outputs of the process are basic characteristics of detected objects (such as size, area, etc.).

In the feature extraction phase (Step 3), information received from the previous step is calculated into 23 invented features/metrics. The output data of the process is represented as float numbers and is much more complicated when metrics comparing with its input data. Detailed information about the features is described in Section 2.4.3.

The features are then used as input data for a UML class diagram classifier (Step 4), which was trained by using our 1300-image-set in conjunction with classification algorithms. The processes of training and predicting is also discussed in the section 2.4.4.

Finally, at the end of the above steps (Step 5), we evaluate performance of the said features and algorithms.

## 2.4.2 Image processing

This sub-section shows how we process an input image for the purpose of extracting features. The process includes two main phases as follow:

**Pre-check.** In order to avoid prolonged processing time on complex photographs, images have to pass a pre-check before being processed as follows:

- The most used colour in the image has to cover at least 10
- The image's colour-histogram median value must be above 100.

**Image processing.** Shape and line extraction is carried out using three external algorithms: *Hough transform (HT)* [106]; *Suzuki85 (S85)* [107]; and *Ramer-Douglas-Peucker (RDP)* [108]. The contours that *S85* finds are used to find various shapes and are subsequently broken down into straight lines. Using the algorithm in conjunction with *HT* leads to better detection of lines. The lines are then processed, so that horizontal and vertical lines, that are on

the same axes and represent the same line, are joined together into a single line. Rectangles that are not caught by using *S85* are then extracted by finding horizontal lines that are parallel and in the same position on the x-axis, and have the same two vertical lines intersecting them on each end. *RDP* is used to find different types of polygon: rectangles, rhombuses, triangles and ellipses.

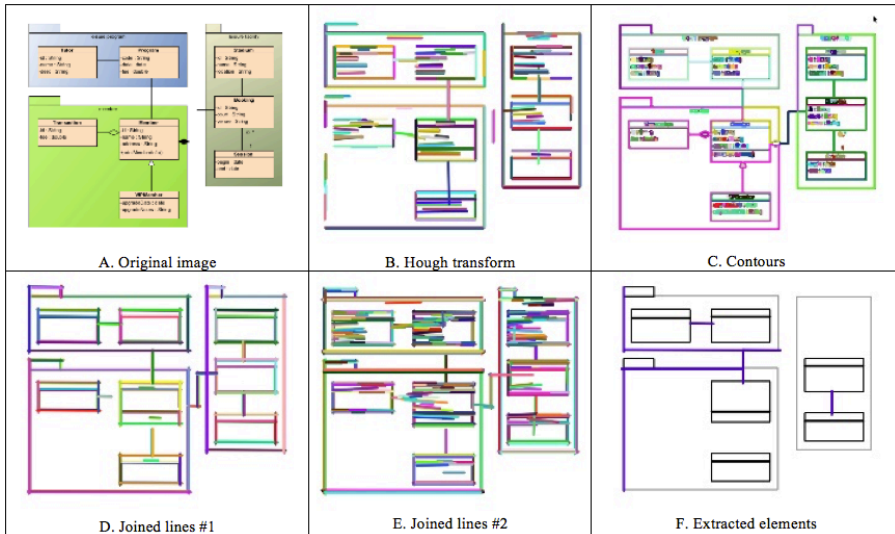


Figure 2.2: Image processing

Figure 2.2 shows the basic steps of the image processing. As can be seen in picture B in Figure 2.2, with *HT*, many of the rectangle lines are not extracted, or the extracted lines are segmented and/or incomplete. Such lines make it very difficult to find the rectangles in the image. *S85* returns an unlimited amount of points in each contour. The extracted contours from *S85* can be seen in picture C. By examining that picture, it is apparent that the algorithm catches more of the lines than *HT*. The lines are joined in three phases:

- (1) The contours that are found are split into lines, and horizontal and vertical lines are extracted;
- (2) Horizontal and vertical lines that *HT* finds are collected and joined with (1);
- (3) Lines, found by *HT*, that are not vertical or horizontal are collected and joined with (1).

After the phases (1) and (2) (picture D), rectangles are collected through the above-mentioned method. After the rectangles have been collected, phase (3) (joining lines; picture E) is conducted, and then all lines within shapes are removed, results in picture F.

### 2.4.3 Feature extraction

As a diagram, a UML CD image can be distinguished from other images by detecting diagram's main characteristics such as lines, rectangles, number of colours, etc. The task becomes more complicated when recognising UML CD from another type of (engineering) diagram. This section explains the problems



and describes the features we extract for solving this classification problem in detail.

2.4.3.1 Which features set UML CD apart from other diagrams?

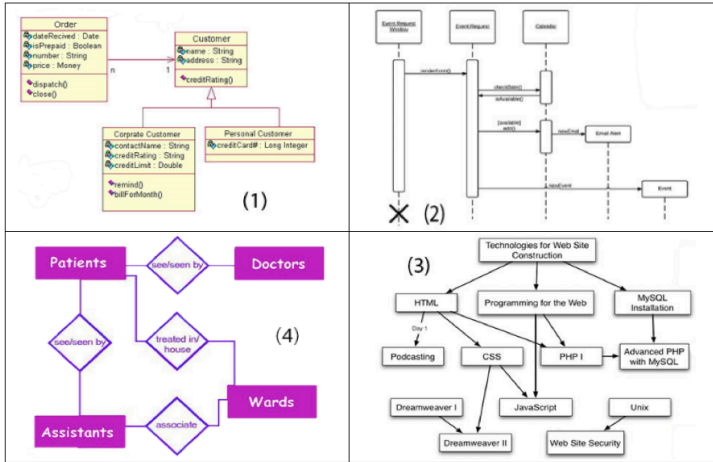


Figure 2.3: Diagram examples.

(1) – UML Class diagram; (2) – UML Sequence diagram; (3) – Flow chart; (4) – E/R model

Diagrams come in all shapes and forms (Figure 2.3), and for this reason, it was important to consider not only CDs, but also other different but similar diagrams such as Entity–relationship models (E/R), UML Sequence diagrams and Flowcharts amongst others, when finding the right features.

Three key factors that can be used to describe UML CDs are: (1) Classes, in the form of rectangles; (2) the classes are related to each other in the form of connecting lines; and (3) the classes are divided into sections with the name of the class, attributes and operations. The 3rd describing factor is, though, not universal. It does not apply to all classes within the diagram, but in almost all UML CDs there are classes divided in this manner. As can be seen in Figure 1, the 1st and 2nd of the defined characteristics of UML Class diagrams can apply to many types of diagrams or charts. Because of that it was also important to extract more information from the image, than only information that is descriptive of CD. As a result, other geometrical shapes (i.e. ellipses, rhombuses, and triangles) and statistical metrics (e.g. distribution of shapes) had to be extracted as well.

In order to obtain a general solution, we considered it important that the input images cover wide ranges of sizes, colours and number of objects. In order to make our features comparable, we use normalisation: all extracted features are represented in the form of ratios and percentages.

2.4.3.2 Extraction features in details

There are 23 features that are calculated from image processing images. Table 2.1 describes the features in details.

Table 2.1: Extracted Features

Feat.	Name	Description
<b>F01</b>	Rectangles' portion of image, %	Dividing the sum of the area of all the rectangles with the area of the image
<b>F02</b>	Rectangle size variation, ratio	Dividing the rectangle size standard deviation with the rectangle average size
<b>F03-06</b>	Rectangle distribution, %	The image is divided into four equally sized sections and the area of the rects inside the sections is then divided by the total area of the rects. The 4 sections sum up to 100%
<b>F07</b>	Rectangle connections, %	Calculated by counting all rectangles that are connected to at least one rect., and dividing that number by the total amount of rectangles in the image
<b>F08-10</b>	Rectangle dividing lines, %	The rectangles are split into three groups, with rectangles that have: no dividing lines (F08); one or two dividing lines (F09); or three or more dividing lines (F10). This produces 03 numbers that represent the percentage of rects. within each group
<b>F11-12</b>	Rectangles horizontally/vertically aligned, ratio	Sides of rectangles, horizontal (F11) and vertical (F12), that are aligned with sides of other rectangles are counted. The numbers are divided with the number of detected rect. in the image, resulting in two ratios on rect. horizontal & vertical alignments
<b>F13-14</b>	Average horizontal/vertical line size, ratio	Average size of horizontal (F13) and vertical (F14) lines that are larger than 2/3 of the images width or height, divided by the images width or height, respectively
<b>F15</b>	Parent rectangles in parent rectangles, %	Rectangles that have rectangles within them can possibly be packages. This feature is the percentage of the area of those parent rectangles that is within other parent rects.
<b>F16</b>	Rectangles in rectangles, %	This is calculated in the same manner as F15, but with rects., instead of parent rects.
<b>F17</b>	Rectangles height-width ratio	The average ratio between the height of the rectangles and the width of the rects.
<b>F18</b>	Geometrical shapes' portion	The same as F01, but with rhombuses, triangles and ellipses
<b>F19</b>	Lines connecting geometrical shapes, ratio	The number of connecting lines from shapes, other than rectangles, divided by the number of detected shapes in the image
<b>F20</b>	Noise, %	Detected lines that are outside of rectangles, divided by the number of all lines
<b>F21-23</b>	Colour frequency, %	Three most frequent colours in the image are found. Then a percentage out of all appearing colours is found for the three colours

## 2.4.4 UML CD classification

This subsection explains how we choose a classifier. The process includes two main tasks: 1) Choosing the most suitable classification algorithm; 2) Training for the classifier with the chosen algorithm. The influence of extracted features and correlation-based feature-sets are discovered as well.

### 2.4.4.1 Choosing the most suitable classification algorithm

We selected the algorithms that represent different approaches in classification. The six classification algorithms are listed as: (1) Decision Table (DT); (2) Random Forest (RF); (3) Support Vector Machine (SVM); (4) Logistic Regression (LR); (5) REP-Tree (RT) and (6) J48 Decision Tree (J48) [52].

At first, we use Information Gain Attribute Evaluator (InfoGain) to find out the influence of extracted features. Secondly, by applying the *Correlation-base feature selection (CFS)* algorithm described in [109] on the extracted features, we prepared several sets of predictors. The set of predictors used for this evaluation are top 3, top 6, top 9 and “top-all” of most suitable features. Then, we apply these sets of features to all classification algorithms to get their false-positive (FP) and true-positive (TP) rates on our dataset.

### 2.4.4.2 Training classifier

This sub-section shows the phase of training the UML CD classifier. To that end, we use our 1300-image-set as training data and a supervised learning approach. The collection of the image-set and configurations for training and testing set are described in 2.5.1 and 2.5.3, respectively.

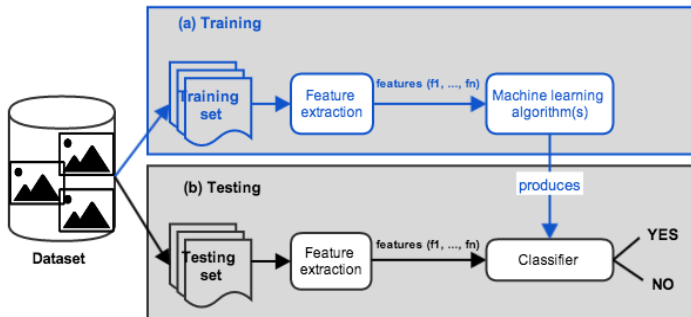


Figure 2.4: Supervised classification - (a) Training phase; (b) Predicting phase

As shown in the Figure 2.4, during the training phase, the feature extraction is to convert each input image to a feature set as mentioned in section 2.4.3. Feature sets are then inputted into the chosen machine learning algorithm to train a model. During the prediction phase, the same feature extraction is applied to the test data, and the extracted feature sets are input into the model to generate the predicted labels.

## 2.4.5 Analyse Result

The InfoGain measures and the FP and TP rates from the classification process are analysed. We compare the performance of the evaluated classification algorithms across all datasets. The detailed analysis is demonstrated in the section 2.6 of this paper.

## 2.5 Experiment Description

This section explains the dataset that we used in this study and the evaluation measure for analysing the results.

### 2.5.1 Dataset

The images that were used for training machine learner collected by using Google Image Search. The image collection consisted of two separate accumulation phases: collecting images that represented CDs; and collecting non-CD images and images that represented similar diagrams.

To search for CDs the phrase “UML Class diagram” was used. Various types of diagram such as blueprint, sequence diagram, chart, flow chart, E/R model, and architectural diagram were found by their corresponded phrases.

It was verified that no duplicates are in the set. The end-result was a collection of 650 UML CDs and 650 non-UML diagrams (1300 images in total). The non-UML images include 60 sequence diagrams, 34 use-cases, 61 ER diagrams, 80 architectural diagrams and 155 charts. Our dataset together with the results that are presented later in the paper can be found online via: <http://bitly.com/dtsUMLClassifier>.

### 2.5.2 Evaluation measures

This subsection describes the evaluation measures used in this experiment. The evaluation measures are the following:

#### 2.5.2.1 Features Predictive Performance

In order to measure predictive performance of extracted features we uses the information gain with respect to the class.

The Information Gain Attribute Evaluation (InfoGain Attribute Evaluation) is a method that evaluates the worth of an attribute by measuring the information gain with respect to the class [110]. This method is able to evaluate the predictive power of an attribute (an extracted feature in our case). Accordingly, we use the method to identify the influence of a feature in UML CD prediction. The InfoGain Attribute Evaluation produce a value from 0 to 1 in which a higher value indicates a stronger influence.

#### 2.5.2.2 Classification Algorithm Performance

We use a confusion matrix to evaluate the machine learning classification algorithms. Table 2.2 shows a confusion matrix. In this table, for the case of the actual data is positive (Y), TP represents the number of correct predictions

(true positive) and FN represents the number of incorrect predictions (false negative) by the classification algorithms. In the case of the actual data is negative (N), FP represents the incorrect predictions (false positive) while TN represents correct predictions (true negative).

Table 2.2: Confusion matrix

Actual Result	Prediction Result	
	Y	N
Y	TP	FN
N	FP	TN

We use Sensitivity and Specificity to evaluate the performance of classification algorithms. Sensitivity (or True Positive Rate) measures the proportion of images, which actually are UML CDs, are correctly identified as UML CDs. Specificity (or True Negative Rate) denotes the proportion of actual non-UML CD images that are correctly classified as non-UML CDs. In other words, while specificity represents the ability of excluding non-UML CD images, sensitivity represents the ability of including UML CD images. The two metrics are calculated from the confusion matrix as below:

$$specificity = TNR = \frac{TN}{TN+FP} ; sensitivity = TPR = \frac{TP}{TP+FN}$$

For our purpose, the exclusion of non-UML CDs is more important than the inclusion of UML CDs. As a result, specificity is considered more important than sensitivity. The two measures range from 0% to 100%. The higher the value of the measures, the better classification algorithm.

### 2.5.3 Experiment settings

This subsection describes the experiment setting in this study. We choose 10-fold cross-validation [111] for performance evaluation where all images are randomly split into ten exclusive folds. For each of the ten experiments, typically a single fold is retained as a validation data, and the remaining nine folds are used as training data. The default settings suggested from WEKA were used for classification algorithms.

## 2.6 Analysis Of Results

This section describes the analysis of results. Every subsection is presented to answer the questions specified in Section 2.3.

### 2.6.1 RQ1: Influence of features

The overall results for this evaluation are illustrated in Table 2.3 in which features are sorted by descending order of InfoGain values.

Overall, 19 out of 23 proposed features are considered as influential predictors (InfoGain value  $> 0$ ) for classifying UML CD images. *F09* and *F08* (ranked first and fifth respectively) are features formulated by calculating splitting lines in the rectangle. Thus, this result shows that splitting lines the rectangle gives a high impact in predicting class diagrams from images.

Table 2.3: Information Gain

No.	Features	Value	No.	Features	Value
1	F09	0.473	13	F18	0.111
2	F20	0.433	14	F14	0.086
3	F01	0.374	15	F10	0.07
4	F13	0.352	16	F21	0.055
5	F08	0.306	17	F19	0.052
6	F02	0.302	18	F22	0.039
7	F07	0.255	19	F15	0.008
8	F04	0.241	20	F23	0
9	F05	0.227	21	F16	0
10	F03	0.208	22	F12	0
11	F06	0.206	23	F11	0
12	F17	0.201			

Note: Features that have InfoGain value greater than 0 are highlighted

Another important feature is *F20*, which is defined to eliminate those images that have too much information outside rectangles. Also, *F01* (ranked third) which denotes rectangle coverage, is one of the most vital features.

*F23*, *F16*, *F12* and *F11* have a trivial impact on the classification. Thus, these features are then omitted from the feature-set.

## 2.6.2 RQ2: Classification algorithms performance

The classification algorithms were evaluated by measuring specificity and sensitivity over 10 runs for the feature set.

Table 2.4: Specificity and Sensitivity Scores

	SVM	RF	J48	LR	RT	DT
Spec.	0.89	0.904	0.901	0.914	0.901	0.895
	0.04	0.04	0.04	0.03	0.04	0.04
Sens.	0.924	0.959	0.925	0.902	0.92	0.919
	0.04	0.03	0.03	0.04	0.04	0.04

Table 2.5: Confusion matrix – Logistic Regression Classification

Actual Result	Prediction Result	
	Y	N
Y	596	54
N	63	587

Table 2.4 shows the evaluation result. The first row and the second row show specificity score and sensitivity score, respectively. Followers are their *standard deviation*.

As can be seen in the Table IV, in term of sensitivity, *Random Forest* shows an excellent result with almost 96% UML CDs images were correctly classified. This follows with *J48* and *SVM* with 92.5% and 92% respectively.

On the other hand, in term of specificity. *LR* performed the best with 91.4% of correctly classified non-UML CDs images. *SVM* performed the worst specificity with 89%.

The results also show that the standard deviation on the results are relatively small (0.01- 0.05) that indicate the results are considered reliable (small variation). In summary, *LR* performs the best in term of eliminating non-UML CD images. Accordingly, *LR* is considered as the best classification algorithm for our classifier with mentioned-extraction features.

The confusion matrix in Table 2.5 illustrates the classification result generated by applying the *LR* classifier to our dataset. From total of 1300 images, 1183 images were classified correctly. 596 out of 650 UML CD images were correctly predicted as UML CDs. Also 587 out of 650 non-UML CD images were correctly recommended as non-UML CDs. On the other hand, among 117 incorrectly classified images, there was 54 false positives (predicted as UML CDs, but actually non-UML CDs) and 63 false negatives.

### 2.6.3 RQ3: Set of features Performance

In this subsection, we describe the sets of features that were used as candidate dataset and the comparison between these sets in terms of the performance that classification algorithms can reach on them. Again, specificity and sensitivity are the two measures that are used to evaluate the performance of the feature set.

For this evaluation, we form four feature-sets by grouping the features into 3, 6, 9 and all features. For groups of feature that have 3, 6, and 9 features, we used *Correlation-based Feature Selection (CFS)* Evaluator to select the suitable features. These sets are as follows:

- Feature set 0 (FS0) = All features
- Feature set 1 (FS1) = F01, F09, F13
- Feature set 2 (FS2) = F01, F02, F09, F13, F18, F20
- Feature set 3 (FS3) = F01, F02, F06, F07, F08, F09, F13, F18, F20

As can be seen in Table 2.6, the set of all feature (*FS0*) shows a more positive result compared with other sets in almost of all classification algorithms. Two out of six classification algorithms gained the best result on both specificity and sensitivity scores with the set of all features. With regards to specificity score, *FS0* is the most suitable feature-set for *SVM*, *LR* and *DT*, while *RF*, *J48* and *DT* perform the best on the 6-feature-set (*FS2*). In terms of sensitivity, *FS0* is considerably higher than other sets as it is the best choice for 4 algorithms.

With focus on the best algorithm (*Logistic Regression*) that is analysed above, *FS0* is the best choice in both specificity and sensitivity, at 91.4% and 90.2%, respectively.

Table 2.6: Specificity and Sensitivity Scores Across Datasets

	SVM	RF	J48	LR	RT	DT
FS0	0.890 +	0.904	0.901	0.914 +	0.901	0.895 +
	0.924 *	0.959 *	0.925 *	0.902 *	0.92	0.919
FS1	0.873	0.898	0.908	0.861	0.903	0.895 +
	0.839	0.926	0.92	0.858	0.919	0.921 *
FS2	0.874	0.907 +	0.916 +	0.882	0.905	0.895 +
	0.894	0.947	0.922	0.853	0.924 *	0.919
FS3	0.885	0.906	0.908	0.901	0.907 +	0.895 +
	0.915	0.949	0.925 *	0.892	0.922	0.919

Note: For each feature set: The first row is specificity and the second row is sensitivity cells that have highest value across all algorithms are highlighted as yellow and orange, respectively. For each algorithm: cells that have highest specificity and sensitivity are marked + and \*, respectively.

## 2.7 Discussion

In this section, we summarize and explain the result in the previous section. We also explain the threats to validity of this study.

### 2.7.1 Image Processing Time

The image processing and input image-set are described in the Section 2.4.2 and Section 2.5.1, respectively. Overall, the average processing time is 5.84 seconds per image. Those images that have big sizes and large amounts of lines need much time to be processed.

In order to discover extraction time's dependence on images size and number of lines, we use Pearson's correlation tests. Obtained results show that there is a moderate relationship between execution time and image's pixel size ( $\text{corr.} = 0.535$ ). Meanwhile, execution time has a relatively high correlation, at 0.85, with the number of lines. Figure 2.5 indicates the relationship along with its linear model.

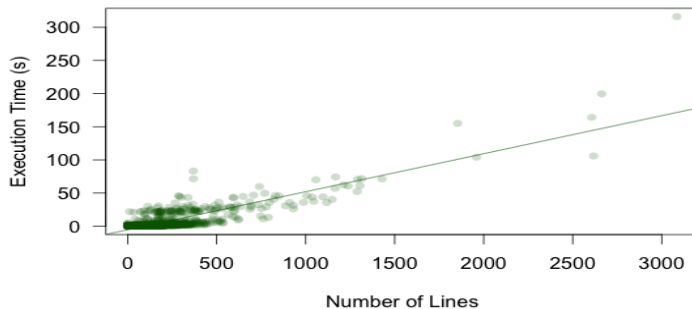


Figure 2.5: Relationship between exec. time and number of extraction lines

As the above discussion attests, processing on architectural diagrams, maps and blueprints might take a lot of time and system resource. Among the 4 most



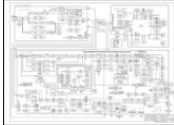


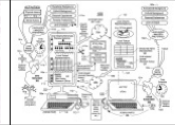
			
t = 315.9 s n = 3085 s = 4758 x 3404	t = 200 s n = 2663 s = 2260 x 1942	t = 164.2 s n = 2607 s = 2106 x 1402	t = 155 s n = 1854 s = 3075 x 2326

Figure 2.6: The most time-consuming cases

(t) – extraction time; (n) – no. of extraction lines; (s) – image size in pixel

time-consuming images showed in the Figure 2.6, only the 2nd one is a UML CD. Therefore, an early recognition of such the images will significantly decrease the execution time. It can be done by applying template-based matching and image pyramids [112].

## 2.7.2 Image Processing Features Performance

Table 2.7 shows the prediction performance of the six classification algorithms using the features. The classification performance ranges from 89% to 91.4% in terms of specificity and from 90.2% to 95.9% in terms of sensitivity. Therefore, we are certain that the proposed features are suitable for classifying UML diagram based on the input images.

The four features whose InfoGain values equal 0 can be considered of too small of influence and can be excluded from the feature set. We check this exclusion by comparing the performance of the classification algorithms on the two features sets: one is full-feature set (so called *FS23*), and the other is the reduced-feature set (so called **FS19**). The result is shown on Table 2.7 explicitly shows that the exclusion helps the classification algorithms to improve their performance of eliminating non-UML CDs. Comparing with *FS23*, while sensitivity scores recorded on *FS19* slightly decrease with at most 0.2% through all algorithms, specificity values increase from 0.2% to 0.7% on 3 out of 6 algorithms.

Table 2.7: Comparison between FS23 and FS19

	SVM	RF	J48	LR	RT	DT
<b>FS23</b>	0.895	0.902	0.901	0.906	0.899	0.898
	0.929	0.961	0.927	0.903	0.92	0.919
<b>FS19</b>	0.89	0.904	0.901	0.914	0.901	0.895
	0.924	0.959	0.925	0.902	0.92	0.919

On the other hand, results from Table III show that features which relate to class's geometric shapes are the most powerful. 8 out of 10 top placed features are about rectangles (distribution, divided lines inside). The two other features are related to connecting lines and information outside rectangles.

### 2.7.3 Classification Algorithms

The results show that *LR* is a suitable classification algorithm in this study as it produces the best specificity score. However, as can be seen from Table 2.7, *LR* is not the best classification algorithm for all feature sets. There is a remarkable decrease when applying *LR* on *FS1*, as its *sensitivity* is 5.3% less than *FS0*. The most suitable algorithm for *FS1*, *FS2* and *FS3* is *J48* Decision Tree whose *specificity* scores ranges from 90.8% to 91.6%.

In term of *sensitivity*, *RF* maintains its first rank and a reliable performance through all feature sets. Its *sensitivity* scores range from 94.7% to 95.9% with a small standard deviation at 3%. From this analysis, we can conclude that *RF* is the most suitable algorithm for detecting the UML CDs.

### 2.7.4 Threats to validity

#### 2.7.4.1 Threats to Internal Validity

Image processing phase is done by applying a process mentioned in the section 2.4.2. However, the process itself has some weakness which is formed by *HT*, *S85*, *RDP*'s disadvantages [113]. The weakness may causes misdetection of classes and connecting lines. Accordingly, the features that are extracted from the images may not be accurate. Picture F, Figure 2.2 is an example of a misdetection: 2 classes are missing in the final step. Using the algorithms in combination with line-segment merging/grouping algorithms such as [114, 115] should improve the weakness.

#### 2.7.4.2 Threats to External Validity

The class diagrams that we used are collected from the Internet. We believe they are representative for the syntactical representation used in various modeling tools including generic drawing tools. One threat to validity is that we have not included industrial class diagrams. In discussions about this research, people claim these industrial diagrams could be larger in terms of number of classes per diagram. On the other hand, our experience is that large diagrams are decomposed into diagrams that consist of around 10 – 12 classes per diagram.

#### 2.7.4.3 Threats to Construct Validity

To measure the classification algorithm performance, we use specificity and sensitivity as our evaluation measures. Specificity and sensitivity can be considered as standard measures in data mining [52]. Therefore, we believe there is little threat to construct validity.

## 2.8 Conclusions and Future Work

This paper presents an automated classification method for images that represent UML Class diagram. To this end, we discuss features extracted from images as input to the classifier for UML class diagrams. In this study, we introduced 23 features that capture statistical and geometric characteristics of diagrams. We find that using these metrics as predictors for the classification

reaches 95.9% and 91.4% (respectively) of correct classification of input images for UML CD and non-UML CD. For this study 1300 different images are collected from the Internet through Google Image Search. We make this dataset available as a benchmark.

We take a step further by examining the classification performance by considering different sets of features. We find out that the full-feature set is the most suitable predictors for most of all classification algorithms. However, we argue that using the full-feature set leads to a time-consuming feature extraction. Therefore, in order to speed up the classification, using other smaller feature-set like *FS2* or *FS3* have only a little lower correct prediction rate, but are faster to compute.

We also study which classification algorithms perform the best on classifying UML CDs. To do that, we calculate and compare their classification performance based on the two measures specificity and sensitivity. Amongst these two measures, specificity is in our case considered more important than sensitivity. Logistic Regression is found to produce the highest correct predication rate, at 91.4%, on identifying non-UML CDs.

Evaluating the performance of classification through the feature sets allows us to identify the most reliable classification algorithms. Random Forest is the most reliable algorithm in term of detecting UML CDs. Meanwhile, J48 Decision Tree obtains top specificity score on 3 subsets of features.

For future work, we will try to improve the performance of the classifier using features based on text-recognition. Another direction would be to try to get a semantic understanding of the diagrams. This could for instance allow the classifier to distinguish organizational charts from class diagrams even if these organizational diagrams cannot formally be discriminated from UML CD syntax.

Also, our classifier allows us to think about a UML CD Crawler which we can use to build a larger collection of UML CDs. Moreover, we consider a classifier for identifying UML sequence diagrams.



# Chapter 3

## Paper B

The Quest for Open Source Projects that Use UML: Mining GitHub

R. Hebig, T. Ho-Quang, M.R.V. Chaudron, G. Robles, F. Miguel Angel

*ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems (MODELS 2016), Saint-Malo, France, October 2 - October 7, 2016.*



## Abstract

**Context:** While industrial use of UML was studied, little is known about UML use in Free/Open Source Software (FOSS) projects.

**Goal:** We aim at systematically mining GitHub projects to answer the question when models, if used, are created and updated throughout the whole project's life-span.

**Method:** We present a semi-automated approach to collect UML stored in images, .xmi, and .uml files and scanned ten percent of all GitHub projects (1,24 million). Our focus was on number and role of contributors that created/updated models and the time span during which this happened.

**Results:** We identified and studied 21 316 UML diagrams within 3 295 projects.

**Conclusion:** Creating/updating of UML happens most often during a very short phase at the project start. 12% of the distinct models occurred several times. Duplicates are in average spread across 1,88 projects. Finally, we contribute a list of GitHub projects that include UML files.

**Keywords:** UML, open source, free software, GitHub, mining software repositories

## 3.1 Introduction

The Unified modeling language (UML) provides the facility for software engineers to specify, construct, visualize and document the artifacts of a software-intensive system and to facilitate communication of ideas [116]. For commercial software development, the use of UML has been introduced and commonly accepted to be a prescribed part of a company-wide software development process.

When it comes to Free/Open Source Software (FOSS) development, characterized by dynamism and distributed workplaces, code remains the key development artifact [37]. Little is known about the use of UML in open source. Researchers in the area of modeling in software engineering have performed some efforts to collect examples of models and of projects that use modelling. However the results are often limited [24]. For example, the Repository for Model Driven Development (ReMoDD) [25] is an initiative driven by an international consortium of leading researchers in the field of modeling. Nevertheless its content is growing at a low rate: after 7 years (summer 2014) it contains around 60 models. Industrial projects are very reluctant to share models because they believe these reflect key intellectual property and or insight into their state of IT-affairs.

Due to the so far limited success in identifying open source projects with UML, many researchers (including the authors themselves at the start of this study) are rather pessimistic finding much use of UML in open source projects. Furthermore, since most open source platforms, such as GitHub, do not provide facilities for model versioning, such as tools for model merging, we were even more pessimistic about finding examples of UML models that were updated over time.

The lack of available data is the reason why so far no answers could be given to several basic questions on the amount of UML files in open source projects that are static or updated, the time span during which models are created or updated during the open source project, or the question which of the project's contributors do create models. Thus it seems that UML is not frequently present in FOSS projects. However, there is no exact quantification of its presence.

GitHub hosts around 10 million of non-forked repositories, which makes it a good starting point to obtain an estimation of the use of UML in FOSS projects. GitHub's web search is limited for this type of endeavor as it targets mainly source code searches by developers. However, there are many other ways to access GitHub data (GHTorrent or the GitHub API), but as we will show obtaining data on UML usage is not trivial.

In this paper we present our efforts to mine GitHub in order to gain a list of open source projects that include UML models. Due to the required manual steps, it is not yet feasible to investigate all 12 million GitHub projects. Instead we focus on a random sample of 10% of all GitHub projects (1,24 million of the 12 million repositories). It turned out that for achieving this goal we required to join forces and expertise from different fields. The first challenge is the identification of non-forked repositories in GitHub with the help of the GHTorrent [45] in order to retrieve candidates for files that might include UML diagrams. Since these many of these diagrams are stored in formats that can



also include other information than models, e.g. images or XML based files, it is further necessary to perform an automated recognition of those files that actually are UML. Therefore, it is required to perform two different checks, one for XML based formats and one for images, which is a state of the research technology that just became available in 2014 [117]. Finally, with the retrieved list of UML models, the git repositories of these projects were triggered in order to retrieve information about the repositories and further information about commit and update histories of these models. As a result we gain out of over 1 240 000 repositories a first list of 3 295 projects containing UML models.

The contributions of this paper are: **1.** A first list of 3 295 GitHub repositories including altogether including 21 316 models. This list can be used by other researchers in future to find case studies and experimental data, e.g. for developing model versioning technologies or for studying how design decisions in models transfer to the code. **2.** Based on this data we give for the first time answers descriptive questions about the number of models that are subject to updates, the number of model duplicates that can be found, and the point in a projects life time where models are created and updated. **3.** Furthermore, this research provides the basis to ask when UML models are introduced and updated. Surely the approach has still limitations, for example we will not be able to identify how often the models are read. However, we believe that these first descriptive results are just a starting point. They enable us and other researchers to formulated and address more advanced questions about UML usage and its impacts on a project in future work.

The remainder of our paper is structured as follows. In Section 3.2, we formulate a number of research questions. Section 3.3 shows our review on relevant works. We describe our study approach in detail in Section 3.4. Our findings are presented and discussed in Section 3.5 and Section 3.6, respectively, including the threats to validity. We conclude our paper in Section 3.8.

## 3.2 Research questions

The data set that we are assessing in this work would allow for a multitude of analysis, e.g. for assessing the distribution of different model types more precisely than it has been done in related work so far. However, answering all questions at once is not possible due to space limitations, but also due to limitation of time. Therefore, we decided to focus in this paper on a set of descriptive questions that had not been addressed in related work so far and that provide a necessary starting point and frame for future analysis:

**RQ1:** *Are there GitHub projects that use UML? Which are these projects?*

**RQ2:** *Are there GitHub projects in which the UML models are also updated?*

These first two questions are interesting for two reasons. First, their answer represents a description of the state of practice that was simply not available so far. Second, projects with updates are ideal candidates for future investigations on model usage. For example, they might be used to evaluate facilities for model versioning.

**RQ3:** *When in the project are new UML models introduced?*

Is it at the beginning of the project or later? What span of the project life time is covered by the phase where UML models are actively created

or modified? Again the descriptive character of this questions is important. Only with the answer, we will be capable to formulate more precise questions on the model usage in future work. For example, whether these figures are homogeneous amongst open source projects or not, will imply directions for future investigations. In long term/ future work this might lead to investigations what form of model usage is most efficient and so on.

**RQ4:** *What is the time span of “active” UML creation and modification?*

With this question we want to know how long is the time span during which models are in active use during a project? A limitation of our methodology is that we cannot investigate how often and when models are read. However, we can have a look at the time span of active UML creation and modification, i.e., the time between the first introduction of a UML file and the last introduction or update of UML files within a project.

**RQ5:** *Are UML files originals?* Special model versioning techniques such as model merging are not explicitly supported by GitHub. Therefore, we are interested in the question how many of the found models are duplicates of other models.

Despite the big interest in these questions, it was until now not possible to answer them. The reason is that simply no systematic knowledge exists about UML in open source projects. Furthermore, even if projects are known, it requires advanced mining of the repository in order to get related information about changes and contributors.

## 3.3 Related research

This paper builds on previous research done in two research communities: the software modelling- and the mining software repositories communities.

### 3.3.1 Use of UML in FOSS

Studies on the usage of UML are frequently done amongst in industry (mostly through surveys) [32, 118]. However, only few studies focus on freely available models, such as can be found in open source projects. Reggio et al. [118] investigated which UML diagrams are used based on diverse available resources, such as online books, university courses, tutorials, or modeling tools. While this work was done mainly manually, Karasneh et al. use a crawling approach to automatically fill an online repository<sup>1</sup> with so far more than 700 model images [26] Both works focus on the models only and do not take their project context into account. Further, they do not distinguish between models that stem from actual software development projects and models that are created for other reasons, e.g. teaching.

An index of existing model repositories can be found online [24]<sup>2</sup>. However, in addition to their small size, these repositories seldom include other artifacts than the models, making it impossible to study the models in the environment of actual projects.

---

<sup>1</sup><http://models-db.com/>

<sup>2</sup>Index of model repositories <http://www2.compute.dtu.dk/~hsto/fmi/models.html>

Further, there are some works addressing small numbers of case studies of modeling in open source projects. Yatani et al. studied the models usage in Ubuntu development by interviewing 9 developers. They found that models are forward designs that are rarely updated [39]. Osman et al. investigated 10 case studies of open source projects from Google-code and SourceForge that use UML. They focused on identifying the ratio of classes in the diagrams compared to classes in the code. They find only seldom cases where models are updated [40].

Finally, there are three works that actually approach a quantitative investigation of models in open source projects. Chung et al. questioned 230 contributors from 40 open source projects for their use of sketches [42] and found that participants tend to not update these sketches. A study that focuses on software architecture documentation in open source projects was performed by Ding et al. They manually studied 2 000 projects from SourceForge, Google code, GitHub, and Tigris. Amongst those projects that used such documentations they identified 19 projects that actually use UML [38].

The work that is probably closest to our study is the one of Langer et al. They searched for files conforming to the enterprise architect file format (which is a format that can be used to store UML files) within Google code, assembla, and GitHub. They identified 121 models. They further assessed the model lifespan (between introduction and last update) to be in average 1 247 days [43]. However, studying a single file format is a rather limited view on UML. Furthermore, the project perspective is not considered and they rather put a focus on the used UML concepts.

### 3.3.2 Mining

Mining software repositories has mainly focused on aspects related directly to (programming) source code. However, projects may include non-source-code sources such as images, translation, documentation or user interface files, that can be usually identified by their extension [119]. By doing so, research has shed some light on the variation and specialization of workload that exist in FOSS communities [120].

The study of specific file formats that are non-source code can be found as well in the research literature: McIntosh et al. have investigated the build system for its evolution [121] and effort [122], or the analysis of infrastructure as code that has become mainstream in the last years [123].

## 3.4 Methodology

In this section, we describe our study approach. The overall process is shown in Figure 3.1.

First, we obtained a list of 10% of the GitHub repositories from GHTorrent [45] that are not forks. This resulted in a list of files of 1 240 000 repositories, those who had a downloadable branch. From this list, potential UML files were collected using several heuristic filters based on the creation and storage nature of UML files (Step 1). Section 3.4.1 and Section 3.4.2 describe our approach in detail.

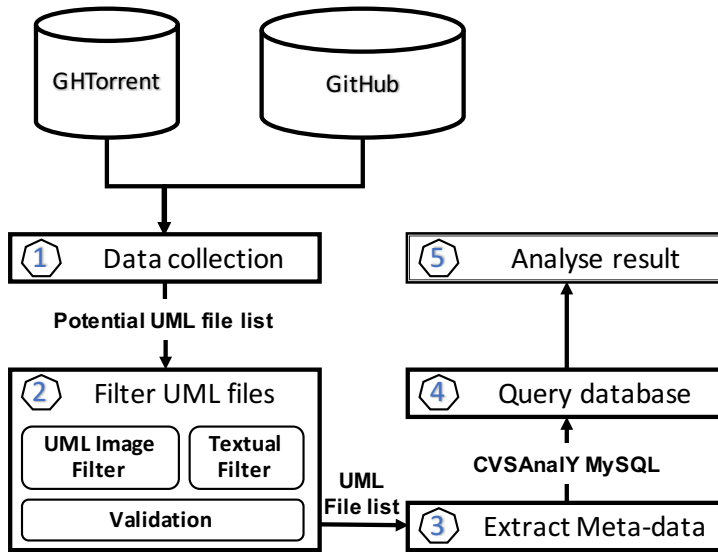


Figure 3.1: Overall processes

An automated process was built to examine the existence of UML notation in the obtained files (Step 2). A manual validation step is taken in order to consolidate the classification result. In the end, we had 21 316 files that contain UML diagrams. We describe the classification method in Section 3.4.3.

We have then obtained the meta-data from those repositories where a UML file has been identified by means of using the CVSAnalY tool [124] (step 3). Section 3.4.4 discusses tool’s settings and the meta-data structure.

In step 4, we queried the metadata (taken in Step 3) with respect to our research questions. We answer the research questions by analyzing the result (Step 5). Note that during the data analysis further files got lost for diverse reasons (see discussion section 3.6). Thus, we were finally able to analyze a set of 21 316 UML model files.

A replication package of our analysis is available online [125].

### 3.4.1 Occurrence of UML

To understand how we searched for files containing UML, it is important to understand how these files are created and stored. Figure 3.2 illustrates the different sources of UML files (at the bottom in green). UML models might be created by manual drawing (sketching). Possibilities to create models directly with a computer are the usage of tools that have drawing functionality, such as Inkscape, or dedicated modeling tools, such as Modelio or Argo UML. Some of the modeling tools even provide the possibility to generate UML models, e.g. based on source code. This differences in tool support lead to a wide variety of ways in which UML models are represented by files. The different possibilities are illustrated in blue at the top of Figure 3.2: Firstly, manual sketches are sometimes digitized with the help of scanners or digital cameras and thus lead to image files of diverse formats. Secondly, tools with drawing capabilities can

either store the UML models as images, such as .jpeg and .png or .bmp, or may have file specific formats, e.g. "pptx". Thirdly, dedicated modeling tools work with tool specific file formats, e.g. the Enterprise Architect tool stores files with a ".eap" extension. Also some tools work with 'standard' formats for storing and exchanging UML: ".uml" and ".xmi". Yet, modeling tools with specific formats often allow to export and import these standard formats and allow to export the models as images. As a consequence, when searching for UML many different file types need to be considered.

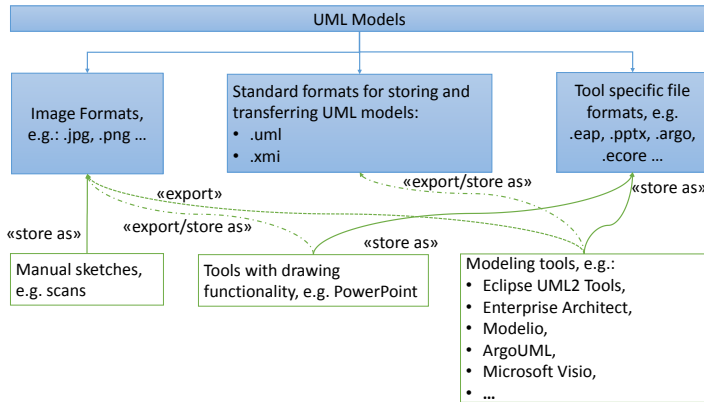


Figure 3.2: There is a large variety of tools for creating and formats for storing UML models

### 3.4.2 Data Collection

For all repositories from GHTorrent [45] that are not marked as forks, we used the GitHub API: i) To obtain file list for **master** branch; ii) If no master branch found, ask for **default** branch; iii) To obtain the file list from **default** branch. With up to three GitHub calls (i, ii and iii) for each repository, given the GitHub API limitation of 5 000 requests/hour, it took over two weeks to retrieve the complete file list once the machinery was set up.

As explained in section 3.4.1, different file formats need to be taken into account. However, as not every image file is UML, also not every xmi file or files with the endings of tool specific format extensions are UML. Therefore, the filtering process does not only consist of the collection of files with a specific extension, but also of a check whether the collected files are really UML files. It makes no sense to collect files in the first step, for which we have no automated support for the second step.

Since image files as well as standard formats are more common and are created by most modeling tools. For each the development of approaches to identify UML has a good cost-benefit ratio. The applied methods are explained below in section 3.4.3. However, for tool specific formats this ratio can be very low. Therefore, we searched only files of those formats where we could exclude two cases:

- The format is used within the tool exclusively for UML models.

- The file extension of the format is not used by other tools. For example the extension of Enterprise architect files (“.eap”) is also used for Adobe Photoshop exposure files.

To identify these formats we used as a starting point the list of UML modeling tools collected on Wikipedia<sup>3</sup>, which we as experts consider as one of the most complete lists available. We checked whether the file formats used by these tools do not fulfill the two obstacles mentioned above.

Thus, we search for following file types:

- Images: Common filenames for UML files (such as "xmi", "uml", "diagram", "architecture", "design") that have following extensions ("xml", "bmp", "jpg", "jpeg", "gif", "png", "svg")
- Standard formats: ["uml", "xmi"]
- UML file extensions that solely relate to specific UML Editor tools: ["aird", "argo", "asta", "dfClass", "dfUseCase", "ecore", "mdj", "simp", "txvcls", "umlx", "ump", "uxf", "zargo", "zuml", "zvpl", "plantuml"]

Hence we do not consider document formats such as word (.doc(x)), .pdf and powerpoint (.ppt(x)). The main reason is that currently technology is not yet capable of extracting UML models out of such general documents.

### 3.4.3 UML filters

At this stage, the files obtained from Step 1 were checked if they really contain UML notation. More specifically, the files which solely belong to specific UML editor tools were automatically added to the final UML file list.

#### 3.4.3.1 Identify UML images

Firstly, all images were automatically downloaded. Files that could not be downloaded or unreadable were eliminated (Result: Successfully downloaded files downloads: 55 747; errors: 1 819). In addition, observations on downloaded images showed a remarkable number of icons and duplicate images. While it's mostly impossible to find reasonable UML content in icon-size images, including duplicate images in candidate set could definitively cause redundancies to classification phase. Therefore, we eliminated icon-size images. Duplicate images were proceeded as: i) Duplicate images were automatically detected; ii) Representative images were added to classification candidate list; iii) After classification phase, duplicate images of an image will be marked as the same label as the image.

In particular, 15 726 images that have icon-dimension-size no bigger than 128 x 128 were excluded. Subfigures 3.3a, 3.3b and 3.3c show examples of such images.

In order to detect duplicate images, we created a simple detection tool by using an open source .NET library "Similar images finder" <sup>4</sup>. Given two

<sup>3</sup>List of modeling tools [https://en.wikipedia.org/wiki/List\\_of\\_Unified\\_Modeling\\_Language\\_tools](https://en.wikipedia.org/wiki/List_of_Unified_Modeling_Language_tools), Last visited 9th December 2015

<sup>4</sup><https://similarimagesfinder.codeplex.com/>

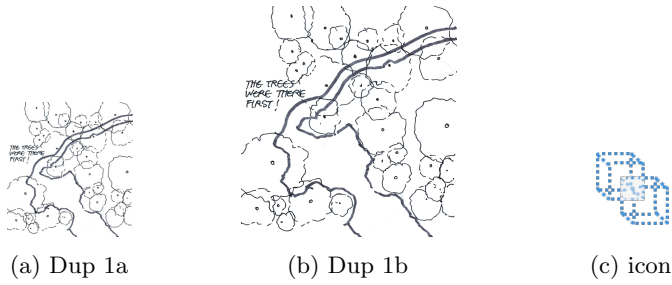


Figure 3.3: Example of duplicates and icon-size images

images, the tool calculates differences between their RGB projections to say how similar they are. In our case, we chose a similarity threshold at 95% since it gave the best detection rate through a number of tests on a subset of our images. Downloading of images took 27 hours.

The final image set of 19 506 images were classified as UML or non-UML images by using a classifier from our prior research [117]. The classifier was trained by a set of 1 300 images (650 UML-CD images and 650 non-UML-CD images). The Random Forest algorithm was chosen since it performed the best in term of minimizing the amount of false-positive rate (expecting below 4%). The automated classification took 26.5 hours. In order to eliminate false-positive and false-negative cases, we manually checked the whole image set. It took 6 working days of effort of an UML expert to complete the checking. This manual check allowed us to prove our classification method and to consolidate classification results. It turned out that the automated analysis had a 98.6% precision and 86.6% recall. The false positives and negatives could be identified due to the manual check.

Gradually, we manually picked up UML in other types (i.e., Sequence Diagram - SD, Component Diagram - CPD, Deployment Diagram - DLD, State Machines - SM and Use-case - UC). UML files that are sketches (SKE) were counted, too. The list of images was marked with a number of labels: "UNREAD", "SVG", "SMALL", "DUP", "CD", "SD", "CPD", "DLD", "SM", "UC" and "SKE".

### 3.4.3.2 Identify UML files among .xmi and .uml files

Both .xmi and .uml files are specific XML formats. The later ones can include uml models, only and we found 10 171 of them. XMI is a standard format that should enable exchange of models between different tools. In theory it should be simple to identify whether an XML file in general contains a UML model: the schema reference in the XML file defines the content's format.

We performed the analysis in 3 steps:

- [a] In practice the schema reference are often generated in different forms by tools. For example, we found following three schema references to the UML: "org.omg/UML", "omg.org/spec/UML", and "http://schema.omg.org/spec/UML". Therefore we first of all searched with a simple search function for the string "UML" and "MOF" (the meta meta model of the

UML language) in a random subset of the models. This way we could come up with a list of 7 strings representing UML schema references.

- [b] In a second step we automatically downloaded the identified xmi files and parsed them for the schema references. We could identify 876 files with UML schema references. However, 359 files could not be downloaded automatically (for diverse reasons). Here we applied a manual check for the schema references.
- [c] In a last step we wanted to double check that the existence of such a schema reference is sufficient to assume that the file includes UML. Therefore, we took a sample of four open source projects containing together 53 (between 1 and 33 respectively) links to xmi files. In addition to the check for schema references, we went manual through the content of the 53 files to assess whether and what kind of models they include. A comparison of the results with the data from the step above confirmed that the existence of an UML/MOF schema is a reliable indicator for rating a file as UML: of the 53 xmi files, 30 had been rated by both approaches as UML, while the other 23 were rated as non-UML.

Finally we run a duplicate detection on .xmi and .uml files by calculating and comparing hash values of the file contents.

### 3.4.4 Metadata Extraction and Querying

We downloaded all repositories where at least one (real) UML file was identified and extracted its metadata with the help of CVSanaly [124]. 100 repositories from the initial list could not be retrieved, due to various reasons, from some giving errors to others having changed to private repositories.

In average, around 30.000 projects per day were downloaded for each Github account. Taking these results a time span of 14 months ( $(12.847.555 \text{ projects} / 30000) / 30$ ) would be required for the analysis, when using one single Github account. As this would have made this study infeasible, we parallelized the retrieval of the JSON files through many Github accounts, which were donated during this process. This reduced the time span to approximately one month. While the download is an automated process, but the parallelization is not. It took around 1 h 30' each day to run and check each set of repositories, using up to 21 Github accounts. Altogether this process took 6 weeks.

After this process, we had 21 316 of the identified UML files from 3 295 repositories and the corresponding meta-data in a SQL database. A new SQL table was added to the ones provided by CVSanaly with just the UML files for easy and efficient querying. A set of Python scripts were then used to answer the RQs stated in this paper by querying the database and aggregating the data accordingly. This final step took 14 days.

## 3.5 Results

This section presents the results of our investigation. Together with this research an ample amount of data have been used, usually handled by scripts developed



by the authors. Detailed information of the former and the code of the latter can be obtained in the replication package<sup>5</sup>.

### 3.5.1 RQ1: UML in GitHub projects

We downloaded 1 240 000 non-forked GitHub repositories obtained from GHTorrent. After filtering the data for potential UML files based on type, we retrieved a list of 100 702 links. Of those, 21 316 were classified as UML.

The further extraction of model related data, turned out to be an additional filter, since details could not be extracted for all files. The reason for this is due to the fact that our retrieval procedure takes so much time that context changes. So, for instance, in the time that goes from the retrieval of information of the files they are included in a project (July/August 2015) to the time where the git repositories were downloaded (November/December 2015), some of them were renamed, deleted or made private.

In consequence, 21 316 files could be retrieved for the following analysis (as summarized in Table 3.1). These files belong to 3 295 GitHub projects. Of these 1 947 include a single UML file, only and 1 169 projects include between 2 and 9 UML files. Furthermore, we identified 158 projects with 10 to 99 UML files and 4 projects with more than 100 UML files. In the following analysis, the later 21 projects are taken separately, when statistics per model are shown. The reason is that they show very different characteristics and would, with their large number of models<sup>6</sup>, strongly bias and hide trends that occur within the other projects. This first list of identified GitHub projects that include UML can be found online [125].

Table 3.1: Found distribution of model files by formats

	xmi	uml	jpeg	png	gif	svg	bmp
Share	3.4%	44.9%	4.7%	29.6%	16.6%	0.6%	0.2%

**Results for RQ1: The here identified repositories with UML files represent already 0.28% of the GitHub repositories. Of these, two thirds of the projects contain a single UML file.**

### 3.5.2 RQ2: Versions of UML models

The next important question was whether models are 'read-only' or also sometimes updated.

Table 3.2 summarizes the distribution of model files by number of updates per model. Our results show that the vast majority of the UML files (18 867) are never updated. Nonetheless, we found that more than 11% of the UML files in our sample (2 449 models) were updated one or more times. Further, the number of updates of models that are updated is on average 3,0 times (although the median, which is more significant given the skewed distribution,

<sup>5</sup>Replication package <http://oss.models-db.com>

<sup>6</sup>One of the projects is "eclipse/emf.compare/", which includes more than 6 000 models. We strongly assume that many of these models are generated, e.g. for tests.

is 1 time). Furthermore, Table 3.2 summarize the distribution of projects by sum of model updates or all models of a project.

26.67% of the projects in our sample include at least one model update. Models are less often updated in projects that have more than 100 models (38.09% in our sample), in contrast to 26.60% of the models in projects with less than 100 models are updated. There are only 11 projects that include more than 100 model updates.

Table 3.2: Distribution of files / projects by number of updates

number of up- dates	models in projects with 1 to 99 models	models in projects with $\geq$ 100 models	projects
0	7 947	10 921	2416
1	946	466	332
2	336	42	157
3	151	19	78
4	107	7	64
5	82	2	51
6	67	4	34
7	38	1	18
8	24	3	17
9	24	1	12
10	11	2	8
<20	70	3	50
<30	24	0	25
<40	14	0	8
<50	1	0	6
<60	2	0	2
<70	0	0	0
<80	0	0	2
<90	1	0	3
<100	1	0	1
>100	0	0	11

**Results for Q2: Only 26% of the investigated projects updated their UML files at least once.**

### 3.5.3 RQ3: Time of UML model introduction

Figure 3.4 shows the dates of the introduction models considering the amount of days since the start of the project, while Figure 3.5 displays the same information by dividing the duration of the project from the start to nowadays in a normalized way (so, the 50% mark would be half of the project duration since its start until today).

Projects with less than 100 UML models seem to have a tendency to introduce models at the project start. In contrast, the 21 projects with 100 or more models show a different graph. We decided to show the numbers

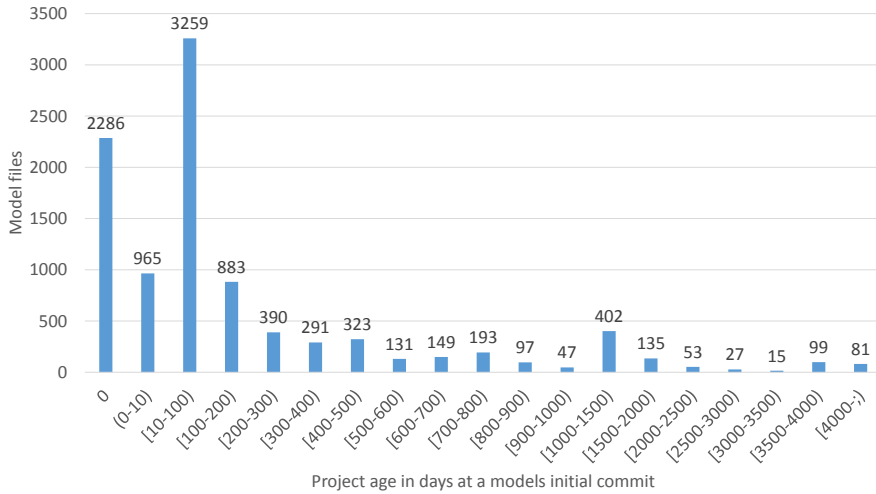


Figure 3.4: Distribution of model files sorted by project's age in days when the diagram was introduced (models within projects that have less than 100 models)

separately, since these projects with partially more than 1 000 models would easily bias the presented view.

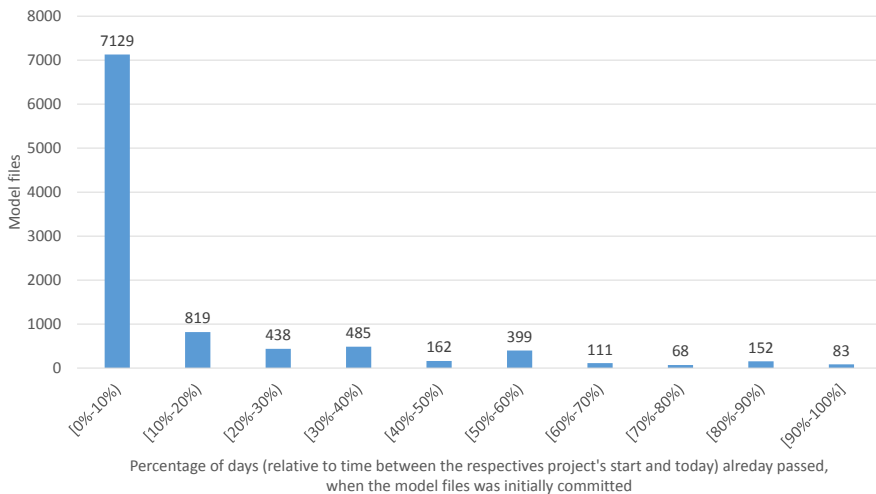


Figure 3.5: Distribution of model files sorted by percentage of project time that passed when the UML file was introduced (for projects that have less than 100 models)

However, we found that calendar time (days) may not be the best way to consider a project's progress, since the amount of activities can highly vary during the lifetime of open source projects. Figure 3.6 shows the distribution of the models based on the time of their introduction when measured by the

percentage of the project's commits.

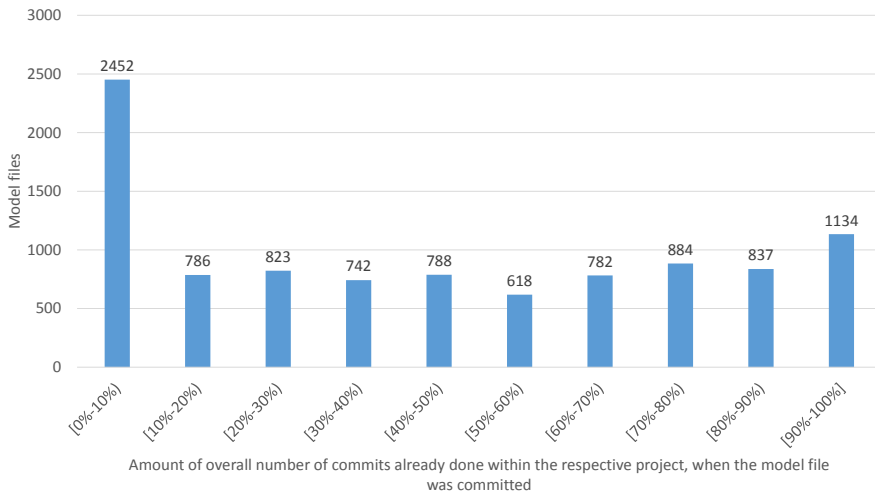


Figure 3.6: Distribution of files sorted by number of overall commits done when the diagram was introduced (For models in projects that have less than 100 models)

An interesting difference between the two views is that the consideration of time in terms of amount of commits shows a much more balanced view. While this may not be the most intuitive notion, it helps to place the modeling activities relative to the active phases of the project. Thus we can see whether model introduction happened before or after a majority of other development activities (such as coding or documenting). In addition, it helps to better represent projects that had their main activity in the past and/or have become inactive. From our results, it can be seen that new models are introduced predominantly in the early phases (above 25% of them in the first 10% of the commits), but that new UML models are introduced in later phases too.

Finally, as mentioned above the results look very different for the 21 projects that have 100 or more models. As Figure 3.7 illustrates there most models are introduced during the last third of the project activities.

**Results for Q3: UML models are introduced in all active phases of a project with a tendency towards the early phases.**

### 3.5.4 RQ4: Time span of active UML

In this RQ, we have looked at the time span of active UML creation and modification, i.e., the time between the first introduction of a UML file and the last introduction or update of a UML file within a project.

Figure 3.8 summarizes these time spans. The maximum time span found is thereby 100% of the projects commits, while the median of the time spans is 5.8%. We found that by far most projects seem to introduce (and update) all models within a single day. Model creation and updating plays only in a minority of the projects a role during more than 10% of the project's commits.

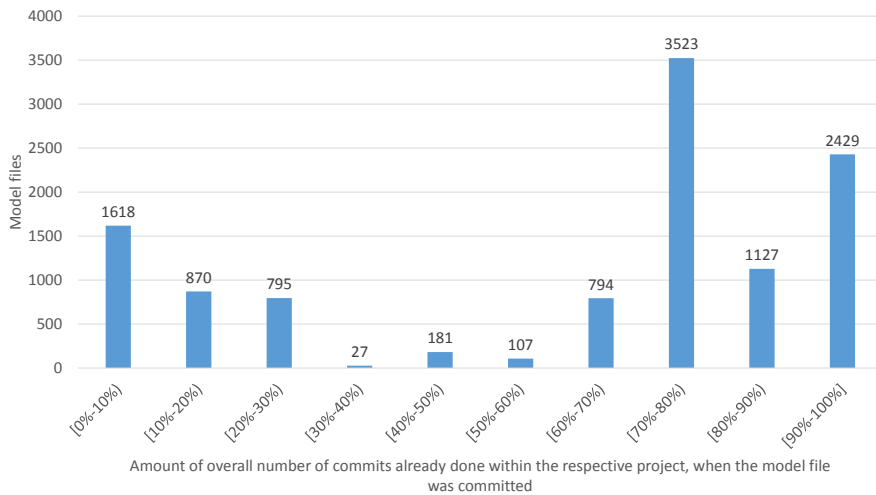


Figure 3.7: Distribution of files sorted by number of overall commits done when the diagram was introduced (For models of the 21 projects that have 100 or more models)

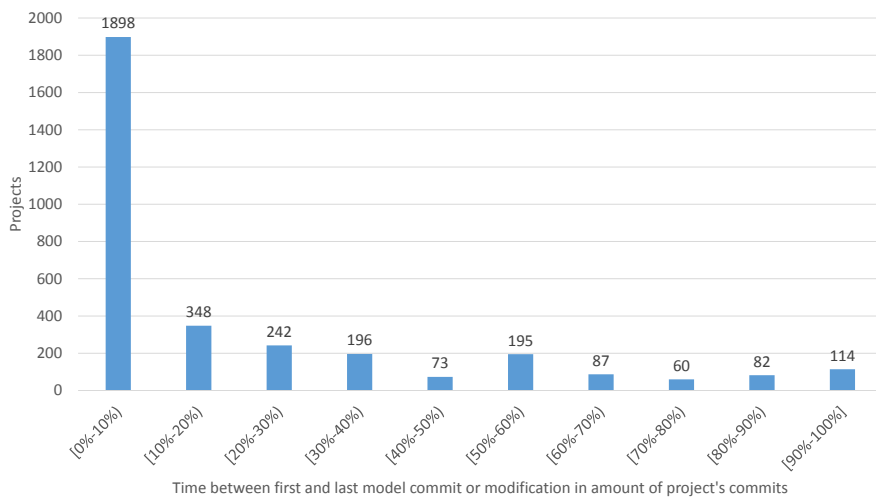


Figure 3.8: Projects by time between first model commit and last model update-or-commit as percentage of project's commits

As with RQ3, we use commits as an alternative measure of the time where UML introductions/updates occur. Figure 3.9 presents the active UML phase for all 3 295 projects from this perspective. The active UML phase of a project is given horizontally in percentages of commits done, starting when the first model is introduced and ending when the last model is introduced or updated. The diagram illustrates nicely the findings from above that a minority of projects (less than 10%) have UML active phases that cover nearly the whole project life time. For a majority of projects the active UML phase is very short, many of them concentrating this activity in the first commits.

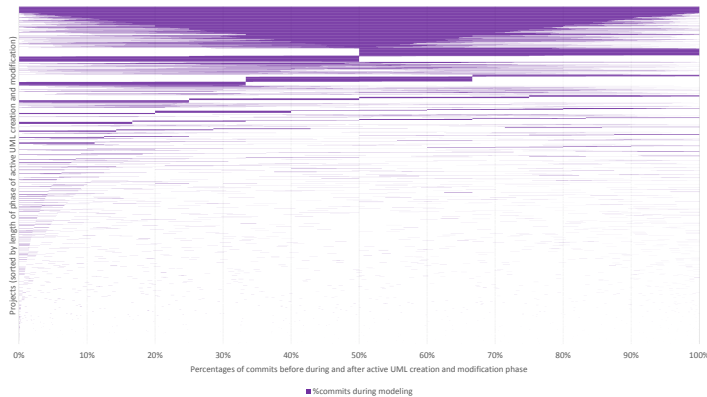


Figure 3.9: Plot of all 3 295 projects illustrating the placement of active UML creation and manipulation phase within the overall project life span. Time is measured in percentages of commits done, when the first model is introduced and the last model is introduced or updated. The projects are sorted by the relative amount of the active modeling phase (projects with a relatively long active modeling phase are at the top, projects with a shorter phase are at the bottom).

**Results for Q4: Few of the studied projects are active with UML during their whole lifetime. In general, the projects work very shortly on UML, usually at the beginning.**

### 3.5.5 RQ5: Duplicates

Our final question was whether the 21 316 found model files are all distinct originals. To answer the question we used automated duplicate detection, as indicated in the method section.

As a result we identified that 16 576 of the 21 316 found models were unique in our sample. The remaining 4 741 model files represent 2300 models of which each occurs at least twice. Thus, 21 316 found model files include together 18 876 distinct models. In Figure 3.10 we summarize how often models with duplicates occurred in our sample. Interestingly, one of the models was found 79 times. In average, models that are duplicated are duplicated 3,63 times.

Furthermore, we investigated, whether duplicates of a model belong to the same project. To our surprise this is only for the half of the models

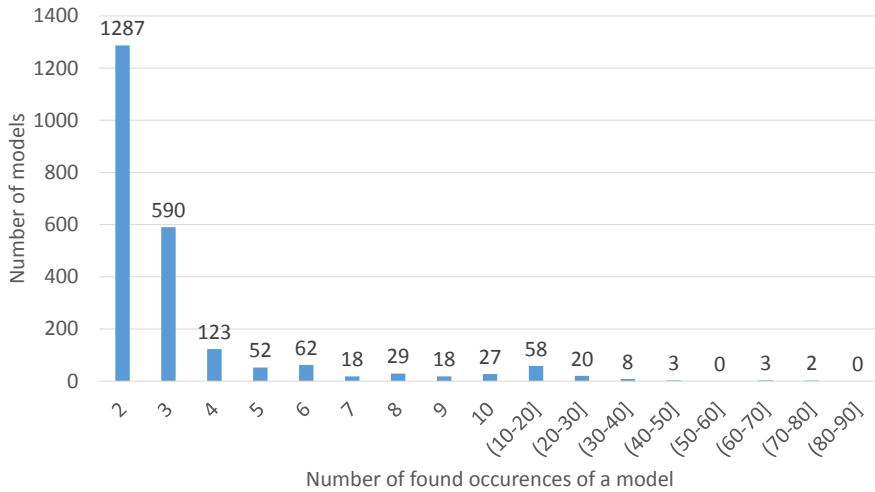


Figure 3.10: Histogram of models that were found at least twice indicating how often models occur.

with duplicates the case. However, the roughly the half of these models have occurrences in multiple repositories (up to 43). In average the number of projects over which duplicates of a model are spread is 1,88. Figure 3.11 summarizes the results in form of a histogram.

While duplicates that occur in the same repository might be result of attempts to model versions, we cannot explain the high number of cases were models occur in multiple projects. A possible explanation might be that models might be stored as part of platforms or plug-ins that are reused in multiple projects. Another explanation could be project forks that are done manually by cloning repositories instead of using GitHubs fork mechanism.

**Results for Q5: While most models seem to be unique, a large number of identified distinct models (12%) occur several times. In average duplicates are spread over 1,88 projects.**

## 3.6 Discussion

Considering our initial expectations we were surprised to find such a big number of projects with UML. Surely, 3 295 projects are still a small number compared to the overall number of GitHub projects. Nonetheless, the identification of 21 316 UML models exceeds by far the expectations that we had based on the numbers of models found so far in open source projects in related work, e.g. 121 models by Langer et al. [43] or 19 projects with UML by Ding et al. [38].

**Data consistency** We want to shortly discuss the type of data that we can get with the presented mining method. The method we applied is not trivial and consist of several steps of data collection. For example, we search for UML candidates using a GHTorrent dump, but accessed the GitHub API to retrieve

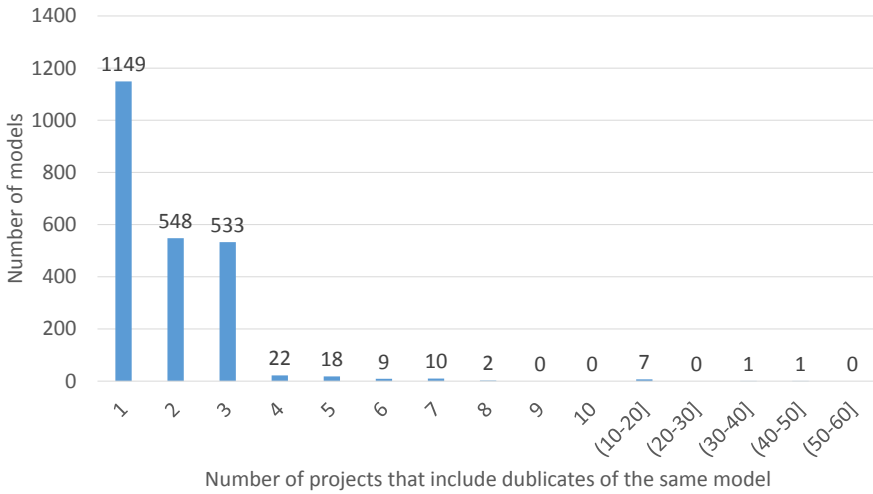


Figure 3.11: Histogram of models that have duplicates in one or more projects. The histogram shows the number of models by number of projects within which occurrences of a model were identified.

further information about model contributors. Due to the difference in time between the creation of the GHTorrent dump and the request to the GitHub API, we had drop outs of identified models/projects during the second step.

In addition, we performed this method for the first time, which had an exploratory component in trying out what kind of data we can (and need to) retrieve. This led to the situation that we accessed the GitHub API several times, leading to different drop-outs in models and projects for the different types of information collected.

A *lessons learned* is that, for the next analysis, we have to make a clear planning of all required data in advance, to ensure that at least the second threat to data consistency can be reduced. For this paper we addressed the problem with a reduction of the finally analyzed data set to models and projects for which we had the data points that are necessary to answer the different research questions.

**Static models** A finding is that many projects use UML only in a very static way. In such projects models are never updated and often all models are introduced at the same point in time. These results confirm findings from smaller studies such as Yatani et al.’s [39] or our own (Osman et al.’s [40]), who both found that updates of models are rare. This can have different reasons. One optimistic interpretation would be that models are just introduced as first architectural plans that are followed and used as documentations, but never changed. Another rather pessimistic interpretation would be that modeling is just “tried-out” at some point in time and then dropped. An observation that at least supports the idea that the optimistic interpretation plays a role is that in most projects the main activities of introducing models happen during the first half of all commit activities.



**Projects with regular model usage** Another number that we consider surprisingly high is the number of projects (or models) with more than 20 updates as well as projects with more than 1 year of active UML creation and modeling. Again, compared to the number of overall GitHub projects the here found number seem small. Nonetheless, it was unexpected to find several projects that seem to use modeling on a regular basis.

It has to be noted that the results we found are in contrast to the study of Langer et al. [43] who found an average model lifespan of 1 247 days, while studying 121 enterprise architecture models in open source. We found much lower lifespans. The difference in the findings might be caused by the fact that enterprise architect is a modeling tool that is rather used in an industrial context. Thus, the probability that the projects studied by Langer et al. [43] have industrial support is very high.

**Model genesis** An aspect that we could not address in this study the source of the models or the reason for model usage. Accordingly, the data set was not filtered to exclude for example student projects. We expect this to influence the findings in this paper, since student projects might show different patterns of model updates, model introduction time, and life span than non-student projects. Addressing this threat will be subject to future work.

**Different populations** A finding that is supported by multiple of the figures shown above is that there seem to be different populations of model usage. A first hint that the data set covers different populations can be seen in Table 3.2. There is a difference in the number of model updates between projects with more than 100 model files and projects with less than 100 model files. One reason for different populations could be the actual form of model usage and creation. Models might be created manually or automatically (e.g. through reverse engineering). They might solve as plans for system design or as description for an already existing system. Model updates might be performed in order to make small corrections after an initial creation (leading to updates within in short span of time) or in order to make a documentation up to date after a longer phase of system change. At the current state we do not know whether these populations can actually be distinguished on their characteristic commit and update pattern. However, a further hint that they might play a role can be seen in the relatively constant distribution models by the amount of commits that were already done within a project (see Figure 3.6). We can see model introductions at all project ages. The in average short time of active UML creation and modification speaks against the idea that these introductions at different points in time happen within the same projects. Thus, it seems that we have to deal with different groups of projects introducing their models at different points in time. In future work we plan to have a closer look at the model usage in order to study whether we can associate pattern to different populations of model use.

**Duplicates** The large number of identified duplicates leads to questions. What are the reasons for duplicates? Missing model versioning techniques alone cannot explain the found results. Furthermore, it is not clear yet whether these duplicates represent a form of model use. E.g. if models are adopted together

with code from other projects, they might be used to understand the alien code that is embedded in a new project.

**Paving the way for future research** Finally, one of our main contributions is that we presented a method to systematically mine for UML models in GitHub and that this leads to an enormously promising set (much larger than any existing set of projects) for future analysis. On the one hand this will help us to address in future question that arise from the findings of this paper. For example, concerning the model updates, it would be interesting to consider following questions:

- Are models updated by their original authors or by other people?
- In how many projects are UML files obsolete?

Further considering the time of model introduction, we would like to address the following question further: Has the time of introduction an influence on the "success" of an open source project, i.e. the question how many developers join a project? And of course we would like to address the question whether different populations of model-usages can be statistically distinguished.

Even more important, the hereby published list of open source projects using UML can help other researchers to progress in their studies. For example:

- What kind of UML diagrams are used most often?
- What coding languages are used most often in combination with UML?
- What files are changed together with changes in architectural models?
- Can UML help to attract and integrate inexperienced developers?

Furthermore, the data can be used to find case studies for other model or architecture related research, such as:

- Does a good architectural design in models help to create a good architecture in the code?
- Tools for traceability management and model merging can benefit from the real case studies.
- Research that integrates models into fault prediction can be evaluated with the help of that data.

Thus, we believe that the identified initial list of open source projects with UML will be of great help for other researchers, too.

### 3.7 Threats to validity

We defined a number of threats to our research's validity. We categorized them by using the validity terminology introduced by Wohlin et.al [126]. We identified three types of threats to validity, they are: Construction Validity, External validity and Conclusion Validity.

### 3.7.1 Threats to construct validity

There were a number of threats that might cause the loss of UML files during data collection phase:

- With regards to the materials that were used to collect data, we used a subset of GHTorrent SQL dump from 2015-06-18 which is out-dated at the current time. Accordingly, newer projects have a higher probability to be dropped out. In addition, the limitation of 5 000 hits per hour of GitHub API made data collection last long. Requests that were done at different points of time during the period could give different outcomes, and probably the loss of potential UML files.
- Our collection method, which made use of a number of heuristic filters, might overlook potential UML files which are not complying with searching terms and file-type list. We noticed some cases where UML files had been named differently such as *act-cartesortir.jpg* and *FrameworkInterface.png*. Further, we restricted the search to file formats for which we had techniques to decide, whether the file includes UML. This excludes a couple of other formats which might include models, such as some formats from modeling or graphic tools (e.g. visio files or enterprise architect files), but also documents that might include models as part of documentations, e.g. pdf and word (docx) files or powerpoint.

The loss of UML files might affect to our analysis in the sense that it could make us underestimate the number of projects with UML models and the number of UML models. Being aware of the above consequences, in this research, we don't use our data to analyze the frequencies of model usage as well as the evolution of model usage in general over \*the years\*. We were focused on getting an overview of various aspects of the use of UML in GitHub projects. We expect no systematic bias concerning the aspects that we investigated!

The applied mechanisms for duplicate detection allow us to identify duplicates within the same file type. However, we cannot identify whether an image and an .xmi file are duplicates. This might lead to an underestimation of the amount of models in this paper. Despite this limitation, our results are already interesting and we consider them a valuable starting point, towards a better understanding of model usage in FOSS.

Kalliamvakou et.al discuss a number of promises and potential risks that researcher might be faced when mining GitHub repository [127]. We found that the threat that many active projects might not conduct all their software development in GitHub could somehow mitigate our analysis.

### 3.7.2 Threats to external validity

During data collection phase, in order to minimize the possibility of incorrectly collect non-UML files, we excluded some tool-specific file types from the search for UML models. This might reduce the generalization of our results with respect to these UML tools. However, most of these tools, e.g. Enterprise Architect, are commercial. It is to be investigated in future work whether they are used in open source projects to a similar degree as non-commercial formats.

Data in this research was only taken from GitHub, but not other OSS hosts/platforms such as SourceForge, Google Code, etc. As they differ to each other in terms of size, functionality, users and user's behaviors, the results of this paper can hardly be generalized to the other platforms. It is possible that UML is used in a different ration within projects at other platforms. However, as GitHub is one of the biggest player in the field, we strongly believe that our investigation gives valuable insights to a majority of the OSS community.

A manual glance at the retrieved list of UML models shows that several project paths include names such as "Assignment" or "master's thesis". While this is no direct threat to our results, it limits the generalizability. For example, it is possible that many of the projects that include single UML files only, actually are result of university teaching.

Last but not least, outcomes of this research can not be generalized to closed source community.

### 3.7.3 Threats to conclusion validity

As described above, the data has some limitations which permit to do analysis of frequencies, since we expect to have only discovered a part of the overall set of UML models and respective projects. In particular we have not considered powerpoint, pdf, and word-formats of documentation in which UML models may be embedded. For that reason we do not do statistical analysis or even predictions, but stay on a descriptive level in this paper. Nonetheless, we are convinced that this descriptive analysis already represents a valuable contribution to the research community.

## 3.8 Conclusions

In this paper we joined forces in repository mining and model identification in order to identify open source projects on GitHub that contain UML models.

As a result we can present a list of 3 295 open source projects which include together 21 316 UML models. This is the first time the modeling community can establish a corpus comparable to collections already exist for source code only, such as *QualitasCorpus*<sup>7</sup>. Furthermore, the relatively low amount of UML projects amongst the investigated GitHub projects (0.28%) reconfirmed that our systematic mining approach was required in order to establish the corpus.

We analyzed the data to gain first descriptive results on UML model usage in open source. One finding is that the majority of models is never updates, but that projects exist that do update their models regularly. Furthermore, we learned that models can be introduced during all possible phases in the lifespan of an open source project. Nonetheless a peak of model introduction is during the first 10% of the duration of projects.

A few projects are active with UML during their whole lifetime. However, most projects work very shortly actively on UML, usually at the beginning. We found that 12% of the distinct models occurred several times. Duplicates are in average spread across 1.88 projects.

---

<sup>7</sup>*QualitasCorpus* <http://qualitascorpus.com/>

---

In the future we plan to further explore the possibilities that arise with the here presented new method to collect data about UML usage in open source projects. For example we plan to analyze the impact of model usage on project dynamics, such as the number of people joining projects. We are planning to proceed with mining GitHub in future work. Based on the now investigated 10% of GitHub we expect that GitHub includes around 34 000 projects with UML and together around 200 000 UML models. Furthermore, we will investigate possibilities to identify UML models that are embedded in other files such as manuals stored in pdf.



# Chapter 4

## Paper C

### Practices and Perceptions of UML Use in Open Source Projects

T. Ho-Quang, R. Hebig, G. Robles, M.R.V. Chaudron, F. Miguel Angel

*In Proceeding of the 39th International Conference on Software Engineering - Software Engineering in Practice Track (ICSE SEIP 2017), Buenos Aires, Argentina, May 20 - May 28, 2017.*





## Abstract

**Context:** Open Source is getting more and more collaborative with industry. At the same time, modeling is today playing a crucial role in development of, e.g., safety critical software.

**Goal:** However, there is a lack of research about the use of modeling in Open Source. Our goal is to shed some light into the motivation and benefits of the use of modeling and its use within project teams.

**Method:** In this study, we perform a survey among Open Source developers. We focus on projects that use the Unified Modeling Language (UML) as a representative for software modeling.

**Results:** We received 485 answers of contributors of 458 different Open Source projects.

**Conclusion:** Collaboration seems to be the most important motivation for using UML. It benefits new contributors and contributors who do not create models. Teams use UML during communication and planning of joint implementation efforts.

**Keywords:** UML; architecture documentation; OSS projects; GitHub; motivation; communication; effectiveness of UML

## 4.1 Introduction

Open Source Software (OSS), which has its roots in the free software movement, started partially as a counter-movement to the software industry in the 80s and 90s [128]. Even though, there was a clear border between OSS and industry, the situation started to change in the late 90s and early 2000s. In those years, some industry started to early adopt the OSS movement practices, collaborating with communities [129], or some companies were created around some communities [130]. Many projects created foundations to serve as an umbrella to collaborate and integrate software industry partners [131].

Thus, we have witnessed a process and technology transfer between OSS and industry that has made the line between both be vague nowadays. Notable contributions from OSS to industry have been technologies, such as git and GitHub, and community-managing practices, although the list of adoptions is much larger [132]. On the other hand, OSS has embraced practices from industry, such as (modern) code review practices and planning and requirements analysis mechanisms [133]. Companies with a large pool of developers try to have an “internal” OSS-like ecosystem, a concept coined as inner source [134]. Many OSS practices are commonly taught at universities, and young graduates start their professional careers with experience in OSS, whether in languages (Python, Perl, Ruby...), products (jQuery, Hadoop...) and tools (GCC compiler tool chain, git and GitHub...) [135]. And the software industry is looking into popular OSS repositories, such as GitHub, to find suitable candidates to fill open development positions [136].

In this regard, we have seen a clash of two worlds, resulting in new practices where industry sometimes has adopted elements from OSS and vice versa. As the trend seems to go on, we would like to draw attention on modeling, specifically on the use of Unified Modeling Language (UML) in OSS. UML has been around as a graphical language for modeling software systems for about 25 years. As far as it is known, UML is not yet frequently used in OSS projects, with a rather marginal use [137]. OSS is known to be programming-driven, with other tasks having room for further improvement [119]. However, modeling is used in major companies [32]. Modeling is, thus, an area where we can find a gap between OSS and industry. Given that the use of UML in OSS is not very well-known, we would like to shed some light into this issue with the aim of discovering how UML is used and whether it is considered useful. We hope that the results will help to understand whether the use of UML in OSS helps these projects and whether industry working with OSS projects should promote its use.

To this end, we used a technique that we developed to find UML use in GitHub projects [137]. This effort showed the feasibility of our approach and triggered us to come up with various research questions addressed in this paper, where we scanned through the majority of non-forked GitHub projects (over 12 million of projects) and identified which of them use UML.

We performed a large scale survey directed at those projects that use UML, with focus on how it is used and impacts development activities. The contributions of this research are: i) the identification of a large set of OSS projects that use UML, and ii) insights from a large scale survey of OSS developers that use UML. Amongst other insights, we have found that UML is

used to coordinate the development. Furthermore the use of UML seems to help new contributors to get started, although it does not seem to attract new contributors. The set of projects we identify are a valuable resource for future empirical studies regarding UML.

The rest of this paper is constructed as follow: We formulate a number of research questions in Section 4.2, then introduce related work in Section 4.3 and describe our research method in Section 4.4. Section 4.5 presents our findings. Our findings, possible threats to validity and implications of our research are discussed in Section 4.6. Conclusions can be found in Section 4.7.

## 4.2 Research Question

To better understand the use of UML in OSS, we formulate the following three main research questions:

**RQ<sub>1</sub>:** *Why is UML used in OSS projects?*

To get an impression of the role of UML models in OSS projects, we formulate this first question as the following.

- *SQ<sub>1.1</sub>*: What are the motivations to use UML modeling?
- *SQ<sub>1.2</sub>*: What are the reasons not to use UML in projects?

**RQ<sub>2</sub>:** *Is UML part of the interaction of (a team of) contributors?*

Teams and interaction between developers play an important role within today's software intensive industry [36]. Models are used as basis for planning and work coordination. However, it is an open question, whether UML models fulfill a similar role in OSS projects. We approach this question from three aspects: 1) awareness of developers about the existence of UML models within the project, 2) the use of UML during project planning and communication, and 3) the role of UML during joined implementation efforts. These three sub-research questions are structured as follows:

- *SQ<sub>2.1</sub>*: Are developers aware of the existence of UML in their projects?
- *SQ<sub>2.2</sub>*: Are UML models used during communication and team decision making?
- *SQ<sub>2.3</sub>*: Are modeled designs adopted afterward during the implementation phase by teams of OSS contributors?

**RQ<sub>3</sub>:** *What is impact/benefit of UML?* Much research has been performed to identify benefits of UML usage in industry. However, it is not yet clear whether UML usage impacts or even benefits development in OSS. Again, we consider three different perspectives: 1) the role of UML for novice contributors, 2) the impact of UML on the working routine, and 3) the impact of UML on the attractiveness of a project for potential contributors. The following sub-research questions are structured:

- *SQ<sub>3.1</sub>*: Can UML models support new contributors?
- *SQ<sub>3.2</sub>*: What are the impacts of using UML in OSS projects?
- *SQ<sub>3.3</sub>*: Can UML models help to attract new contributors?

## 4.3 Related work

In the following we discuss related studies about UML or modeling in industry and OSS.

### 4.3.1 Modeling in Industry

Modeling has been widely studied in industry, in particular in several surveys. Torchiano et al. found that models help to improve design and documentation [32]. However, they also found that model usage is connected to extra effort, especially due to a lack of supporting tooling. Forward et al. find that models are primarily used for design and documentation, while code generation is rather seldom [33]. Gorschek et al. focused on a different population, which are programmers, partially working in industry and OSS [34]. Within their sample design models are not use very extensively. However, models and UML are found to be used mainly for communication purposes. Further, they report on a higher use of models for less experienced programmers.

Case studies have also been performed in order to investigate the impact of modeling/UML usage. For example, Baker et al. found an increase of productivity when using UML in Motorola [7]. Nugroho et al. investigated an industrial case study and found that UML usage has the potential to reduce the defect density and, thus, increase the quality of software [35]. Just as in the case described by Kuhn et al., most of the case studies draw a picture of model use, where models are actually artifacts that are produced and consumed by different people [36].

### 4.3.2 Modeling in Open Source Software

Much less work has been done on UML use in OSS. One reason for this is the challenge to actually find cases that can be studied. For example, Badreddin et al. studied 20 projects without finding UML, and concluded that it is barely used in OSS [37]. Similarly, Ding et al. found only 19 projects with UML when manually studying 2,000 OSS projects [38]. However, in our previous work, we presented an approach that allows to find thousands of projects with UML by mining GitHub [137]. There are several investigations of single or very small numbers of cases of OSS projects that use UML, e.g. by Yatani et al., who found that models are used to describe system designs, but are rarely updated [39]. Osman et al. studied to what extent classes in the diagrams are implemented in the code [40]. Finally, Kazman et al. investigate the Hadoop Distributed File System to learn how documentation impacts communication and commit behavior in the open source system [41]. There are some studies that approach the use of models in OSS with a quantitative perspective, studying a large number of projects. For example, to study the use of sketches, Chung et al. collected insights from 230 persons contributing to 40 OSS projects [42]. Finally, Langer et al. studied the lifespan of 121 enterprise architect models in OSS projects [43].

However, to the best of our knowledge there is so far no quantitative study targeting the use of UML within the team communication and its effects.

## 4.4 Research Methodology

In this section, we describe our study method in detail. The overall process is shown in Fig. 4.1.

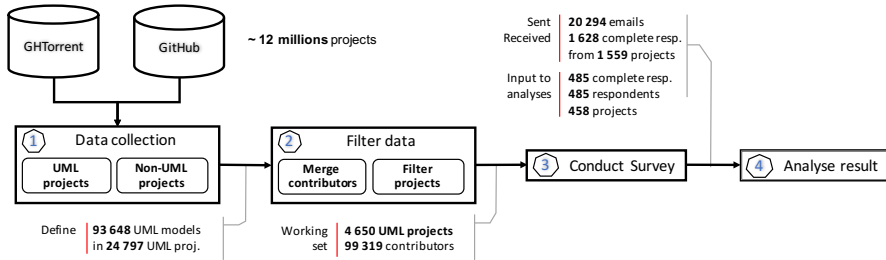


Figure 4.1: Overall process

### 4.4.1 Data Collection

The first step is to identify UML files in GitHub repositories. In our previous work, we analyzed 1.2 million GitHub repositories to identify UML files in them [137]. In this study, we have extended the data collection to the whole GitHub database. A number of changes have been made in order to adapt our method to the retrieval and analysis of such a big dataset. In this section, we briefly summarize the data collection steps and the changes that were made.

#### 4.4.1.1 Obtaining the full list of GitHub projects

To obtain the list of projects, we used the data from the February 1st 2015 dump of GHTorrent [45]. From this dataset we identified a list of projects that were not deleted and non-forks. As GHTorrent does not contain information on the files in the repositories, we made use of the GitHub API to retrieve the list of files, for a total number of 12,847,555 repositories. The result is a JSON file per repository with information on the files hosted in the *master* (or *default*) branch of the repository.

#### 4.4.1.2 Identifying UML files

The next step was to identify UML files from the file list. First, potential UML files were collected using several heuristic filters based on the creation and storage nature of UML files. After that, an automated process was applied to examine the existence of UML notation in the obtained files. A manual validation was made to consolidate the results. Details about the identification procedure are described in Section 4 in [137]. At the end of this step, we had 93,648 UML files from 24,797 repositories.

#### 4.4.1.3 Extracting meta-data

For all projects that contain a UML file, development meta-data from the repositories has been retrieved. Therefore, we use `perceval`, an evolution of the well-known `CVSanaLY` software [124], that allows to obtain these data in JSON files, allowing to perform the data extraction process in parallel. It took

the five instances of the tool over 4 weeks to complete this task. At the end, and after removing 240 JSON files that contained 404 Not Found responses, we had 24,125 JSON files that were parsed and normalized, and finally converted into SQL.

## 4.4.2 Filtering the obtained projects and contributors

In this phase, we aimed at mitigating a number of known threats to validity when mining GitHub, i.e., sample/short-time projects [127] or identification of contributors [138].

### 4.4.2.1 Filtering short-time projects

For this paper we aim at projects that are interesting from an industry perspective. Thus, we focus on projects that are not short-term and that do not consist of a single contributor. We define short-time projects as those projects that have: i) active time (time between the first and the latest commits) less than 6 months, OR ii) less than 2 contributors, OR iii) less than 10 commits. After classifying and filtering short-time projects, 4,650 UML-projects (out of 24,125, we use the term *UML project* to refer to GitHub projects that contain UML file(s)) and 2,701 (out of 17,101) non-UML projects met our requirements. The final list of the projects is shared in our replication package<sup>1</sup>.

### 4.4.2.2 Merging duplicate contributors

A contributor can use different emails or usernames during the project time, and thus a procedure has to be applied to merge all the identities into a unique one. In our work, developers who have different identities are merged when they have the same e-mail address or the same full name. In the case of the full name, we consider them to be the same if at the full name is composed of at least two words or of only a word and numbers, e.g., "arg123". This is a rather conservative approach, but it minimizes the number of false positives [138]. After running the script, the original 129,276 contributors result in 99,319 distinct ones.

## 4.4.3 Conducting the survey

In the following, we give a short overview about how we conducted the survey.

### 4.4.3.1 Participant

To ensure that we obtain a balanced picture, we had to consider the role that contributors play within the OSS projects with UML. Two dimensions of roles are important (each questioned person would fulfill a combination of roles in these two dimensions):

- Founder (F) vs. non-founder (NF)
- Non-UML Contributor (NUC) vs. first UML Contributor (1UC) vs. UML updater (non-1st contributor) (UC)

---

<sup>1</sup>The replication package for this paper can be found at <http://oss.models-db.com/2017-icse-seip-uml/>

Consequently, each interviewed participant fulfills one of the following six roles: F-1UC, F-NUC, F-UC, NF-1UC, NF-NUC, NF-UC. For each project, we randomly selected three contributors, to whom we sent the questionnaire. The selected three contributors had to fulfill one of the following three constellations of roles.

- F-NUC, NF-1UC, NF-UC
- F-1UC, NF-UC, NF-NUC
- F-UC, NF-1UC, NF-NUC

For those projects where we could not identify any NUC or UC (e.g., projects that have only one UML contributor), we contacted less contributors.

#### 4.4.3.2 Questionnaire

The questionnaire has been designed to meet the following requirements:

**Multiple roles** We send different question sets depending on the role of the contributor. For example, NUCs are asked whether they are aware that UML models exist in the project, while UCs are asked if they think that NUCs are aware of them. Thus, depending on the role, participants received between 5 (NF-NUC) and 19 (F-1UC) questions.

**Exploration** We use a funneling approach (from broad to narrow) when designing the survey. For example, if a UC uses a UML model for architecture/design purpose, we would ask if the model is adopted, and eventually, who implemented the model. Accordingly, the number of questions will not only differ among different roles, but also among respondents who have the same role. In addition, to gain more insights, we use a mix of close-ended and open-ended questions in the survey.

**Personalized Contact** To ensure that participants know what projects and UML models we are referring to, we personalized the email with which we contacted potential survey participants by concretely referring to his/her GitHub identification, the name of project of interest, and (if applicable) an URL to his (first) UML commit or to a UML file committed by someone else. By following the URL (e.g., <https://github.com/rvs-fluid-it/wizard-in-a-box/blob/master/src/doc/wizard-in-a-box-design.png>), participants could get further contextual information about the UML models, for example commit messages, commit date, etc.

We used the Lime Survey tool<sup>2</sup> as it offers the possibility to perform on-line surveys. Our Lime Survey server is hosted at <http://survey.models-db.com/>. Details about survey settings (questionnaire and its data-flow diagram) and email templates can be found in the replication package.

---

<sup>2</sup>LimeSurvey homepage: <https://www.limesurvey.org/>

### 4.4.3.3 Sending out the survey

We sent 20,294 survey emails to OSS contributors in 6 days, from July 21 to July 26, 2016. More than 1,000 emails were not sent because of various problems, including out-dated email addresses, etc. We sent reminder emails after one week, and finally closed the survey in August 4, 2016. Altogether, we received 2,230 responses, being 1,628 completed. After filtering responses that belonged to short-term projects, we had 485 survey responses of respondents from 458 projects.

Table 4.1: Number of emails sent, number of responses and number of responses after filtering by participant categories

	Founder			Non-Founder			SUM
	<i>1UC</i>	<i>NUC</i>	<i>UC</i>	<i>1UC</i>	<i>NUC</i>	<i>UC</i>	
<b>Sent emails</b>	4509	3891	713	6737	3221	1223	20294
<b>#full resp.</b>	373	293	68	564	210	120	1628
<b>#inc. resp.</b>	167	105	24	214	56	36	602
<b>#fil. resp.</b>	<i>84</i>	<i>79</i>	<i>27</i>	<i>176</i>	<i>80</i>	<i>39</i>	<i>485</i>
<b>Percent(%)</b>	17.3	16.3	5.6	36.3	16.5	8.0	100

## 4.4.4 Data Analysis

First, we take into account completed responses only. Second, we do not consider short-time projects.

Part of the questionnaire are free-text questions. We use these questions to learn about phenomena for which we do not know a fixed set of answers yet. The goal of analyzing the data is to identify re-occurring themes. Therefore, we used a coding technique, following the constant comparison method as described by Seaman [139]. We decided to use an empty starting set of codes and develop them during the coding. For each of the question two of the authors coded the answers independently. In a second step we inspected the codes together to identify and if necessary resolve differences in the selected codes and application of the coding. Afterward, we went a second time through the data in order to ensure that the now fixed set of codes was assigned consistently. We did this i) to increase the quality of the coding and ii) to decrease the probability that we miss interesting aspects. As a final step we checked whether codes occurred for more than one project, in order to prioritize those themes that are of greater relevance.

Furthermore, we took those cases where we got multiple responses for the same project and aggregated them. This aggregation was done as follows: we interpret observation based questions (i.e., whether UML is used for communication) as reports about a project. Thus, aggregating a “yes” and a “no” answer for the same project to a “yes” to indicate that there is a report about a phenomenon for that project. Similarly, we prioritized “no” over “I have no opinion”. “I do” and “I have seen other people doing” are merged to “I do”.



## 4.5 Results/Findings

### 4.5.1 Respondent Demographics

A total of 2,230 respondents from 91 countries began the survey, with 1,628 completed compulsory questions of the survey. After filtering out survey responses from short-time project participants, we ended up with 485 survey responses of respondents from 458 projects.

Table 4.2 (Appendix 1) and Figure 4.16 (Appendix 2) show the distribution of the respondents by country and continent, indicating the majority of the responses originating from Europe with 57.52%, followed by North America and South America.

Among the 485 respondents, 190 (about 40%) are founders of an OSS project and 159 (32.8%) are non-UML contributors (Table 4.1). Regarding the educational background (as shown in Fig. 4.2), 37.73% of respondents had a Master's degree, 30.31% a Bachelors, 16.29% a Ph.D., and 11.75% were still in education. About 4% of the respondents identified themselves as autodidacts. A vast majority of the respondents reported to be familiar with architecture documentation in different formats, mostly UML (90.31%), then auto-generated code documentation and software models in generic formats (78%) (Fig. 4.3). Only a half of them (45%) were familiar with architectural notations on white papers. There are programming languages where UML is more frequently found (Smalltalk, Java, C# and C++). On the other side, UML has not that much impact in the Objective-C and the Ruby community.

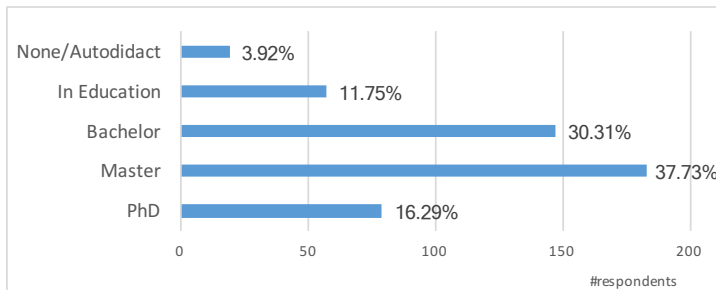


Figure 4.2: Distribution of respondents based on their highest educational background

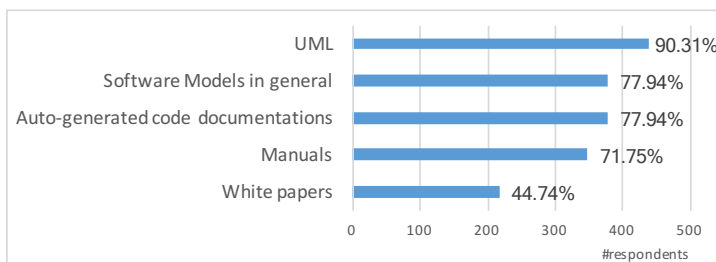


Figure 4.3: Familiar architecture document formats (multiple choices were allowed)

## 4.5.2 Why is UML used?

### 4.5.2.1 What are the motivations to use UML modeling?

Fig. 4.4 shows the answers from 326 UCs (from 319 projects) about the *intent* of UML files they added/updated. Most of UML files served for design/architecture and documentation purposes, with 70% and 71% of votes, respectively. For about 18% of the projects, software verification was mentioned as one of the main purposes. Refactoring and code generation was less usual (14.11% and 12.85% of the projects).

Among 125 NUCs that claimed to be aware of the existence of UML models, 109 people (from 109 projects) reported to find UML helpful (Fig. 4.5). 79% of the respondents found UML useful for understanding the OSS systems. They also found UML models helpful as the models assisted in improving communication within their project, guiding implementation and managing quality of the project.

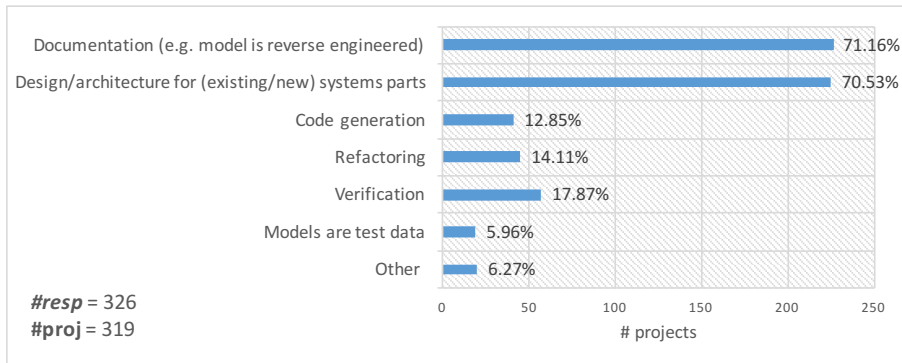


Figure 4.4: Intent of UML models that were added/updated

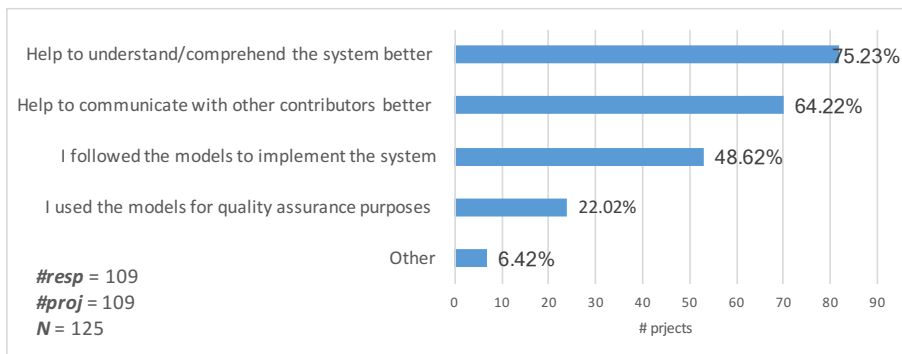
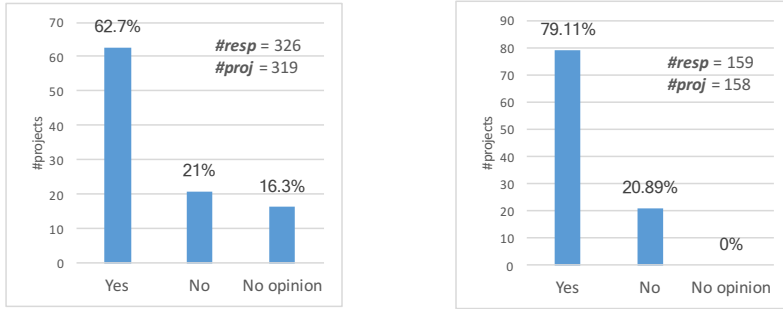


Figure 4.5: How did UML help non-UML contributors?

*Results for SQ<sub>1.1</sub>: The majority of models are intended for creating software designs and documenting software systems. Non-UML Contributors (NUCs) benefit from UML models when it comes to understand a system and to communication.*



(a) Do UCs think that other contributors are aware of UML?

(b) Are NUCs aware of the existence of the UML models?

Figure 4.6: Awareness of developers about the existence of UML in their projects (by project)

#### 4.5.2.2 What are the reasons not to use UML in projects?

To complement our finding on the motivations to introduce/use UML, we asked the 16 NUCs who did not find UML models useful the reasons for this. Respondents from 6 projects actually had not used models, finding themselves not required to learn/use UML (e.g., “there was no demand to do so”). Interestingly, in no case license problems for modeling tools were a problem.

In 4 cases, the UML files were outdated. Other reasons that were brought up in free-texts are: missing support for versioning models, a failed attempt to understand the models, a preference for other means of communication (face to face), a preference for other forms of modeling/sketching, a preference for reading code rather than spending time for UML models, and the dislike of UML (anti-UML attitude).

*Results for SQ<sub>1.2</sub>: Only a small number of respondents found UML not useful.*

### 4.5.3 Is UML part of the interaction of contributors?

#### 4.5.3.1 Developer’s awareness about the existence of UML in their projects

To answer this question, we first asked creators/maintainers of UML models whether they think that the models are known by developers of the projects (summarized in Fig. 4.6a). In 62.7% of the 319 projects with responses, the UCs/1UCs believed that UML models are known by the developers of the projects. Second, we asked NUCs of projects that use UML if they are aware of the existence of UML models in their projects (Fig. 4.6b). Surprisingly, for the vast majority of projects (80%) NUCs stated that they are aware of UML models.

To better understand the difference between the answers of UCs and NUCs, we looked in detail into the 24 projects for which we received responses from NUCs and UCs. In 10 out of 24 projects, NUCs and UCs differed. Interestingly,

UC(s) did not expect their UML to be known by other developers although NUCs were aware of it in 8 of them. It seems that model creators tend to underestimate the spread of their models.

*Results for SQ<sub>2.1</sub>: A majority of non-UML contributors are aware of the UML models in their projects. Awareness is higher than the one expected by the authors of the models.*

#### 4.5.3.2 Are UML models used during communication and team decision making?

In a first step we asked founders and UCs whether UML models are considered in the communication between contributors. Fig. 4.7 summarizes the 405 individual responses from 388 projects. According to the responses, UML models were considered in communications in a large majority of the participated projects (60%).

As a step further, we asked whether UML models were used as a basis for architectural decision making or mentoring activities. Respondents from a majority of the projects recalled that they had used the UML models for making architectural decisions (58.7%) and to explain each other different aspects of the system (58.25%) (Fig. 4.8).

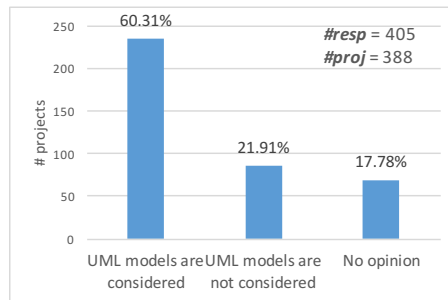


Figure 4.7: Are the UML model(s) considered in the communication between contributors? (per project)

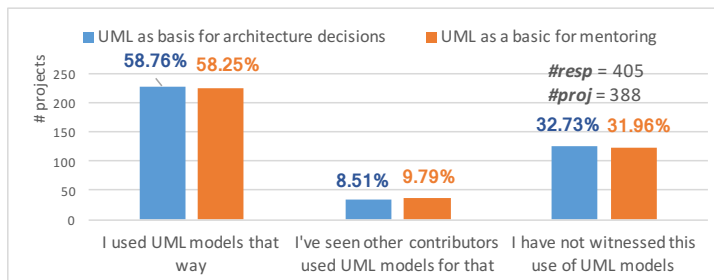


Figure 4.8: Is UML a basis for architectural decisions or mentoring activities? (per project)

*Results for SQ<sub>2.2</sub>: UML models were considered as a mean of communication, as a basis for architectural decisions, and for mentoring in a majority of the projects.*

### 4.5.3.3 Are modeled designs adopted afterwards, during the implementation phase by teams of OSS contributors?

For those projects that claimed to have design models, we asked the question “Was the UML model adopted during the implementation phase?”. Fig. 4.9 summarizes the answers of the 231 respondents from 225 projects. In most cases UML models were adopted partly or completely during the implementation phase (about 92%).

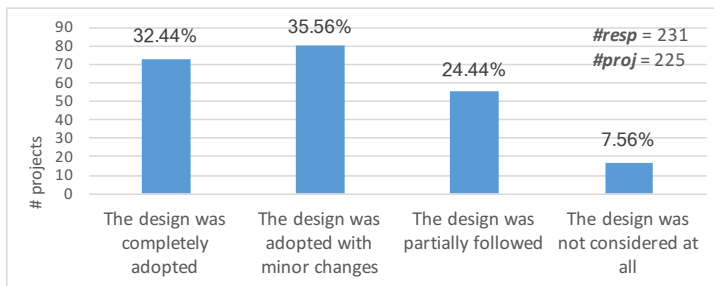


Figure 4.9: Was the UML model adopted during the implementation phase? (by project)

If the answers were that UML models were at least partially adopted, we asked further questions to find out who and how many contributors implemented the modeled designs. Fig. 4.10 and Fig. 4.11 summarise the responses per project (based on 214 individual responses for 208 projects).

Creators of UML models are greatly involved in implementing the modeled designs (in 88.5% of the projects). Experienced contributors helped in 35.5% of the cases and novice contributors helped in around 13% of the cases.

In the majority of the projects (around 66%) more than one person participated in the implementation of previously modeled designs. However, only 7% of the projects reported to have more than 5 contributors involved in such joint implementation efforts.

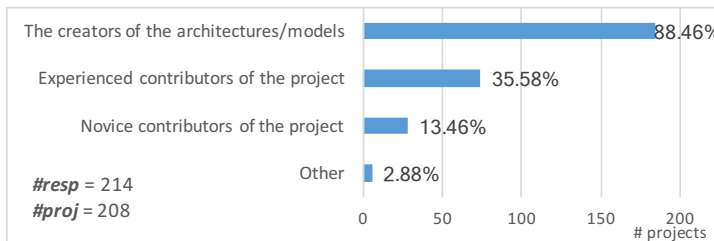


Figure 4.10: Who implemented the UML models? (by project)

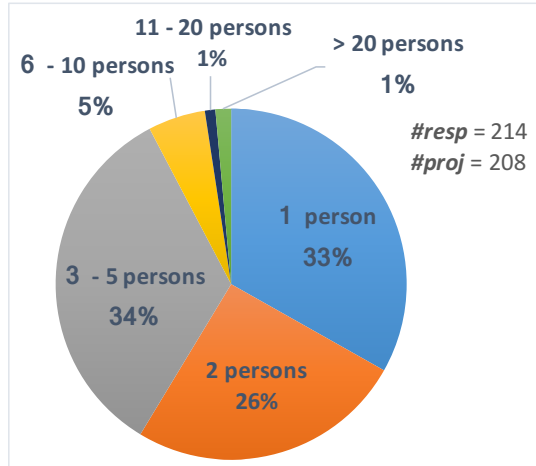
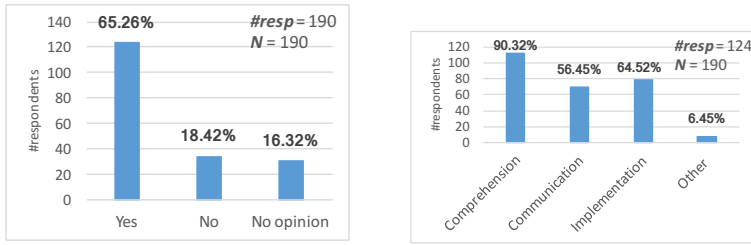


Figure 4.11: Number of contributors who implemented UML models in a project



(a) Do UML models help new contributors? (b) For what tasks do models help?

Figure 4.12: Responses for the questions whether UML models help new contributors to join a project.

*Results for SQ<sub>2.3</sub>: Designs introduced with UML are in most cases adopted during the implementation phase (fully or with slight changes). Most often these designs are implemented by groups of 2-5 developers.*

## 4.5.4 What is the impact/benefit of UML?

### 4.5.4.1 Can UML models support new contributors?

We used two perspectives to approach the question whether UML models support new contributors.

First, we ask founders if they think that UML models help new contributors to join their projects. We received 190 responses from 84 F-1UCs, 79 F-NUCs and 27 F-UCs. For those who agreed, we further asked with what tasks models help. Fig. 4.12 shows the responses in detail. 124 out of 190 respondents (65.26%) agreed that UML models can help new contributors when joining projects. They expected models to assist new contributors in comprehending the system (90%), during implementation phases (65%), and when communicating with other contributors (56.5%).

Second, we asked each contributor what software artifacts he/she used when they got started with the project. 485 contributors answered this question.

Despite the fact that most of respondents were familiar with architectural documents (as shown in Section 4.5.1), source code still remains their first choice to start working with an OSS project (81%) - see Fig. 4.13. Remarkably, UML and software models in general were reported to be starting points for 55% and 43.5% of the respondents, respectively. This is more than the proportion of contributors who started using wikis, issues, manuals, and auto-generated code documentation. This conforms with the answers given by the founders about new contributors.

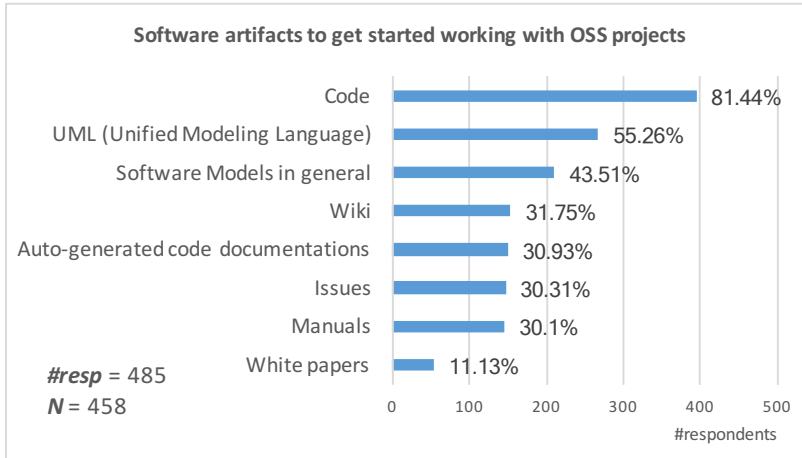


Figure 4.13: Software artifacts used by respondents to start working in their OSS project (multiple choices were allowed).

*Results for SQ<sub>3.1</sub>: The results suggest that UML is helpful for new contributors to get up to speed.*

#### 4.5.4.2 What are the impacts of using UML in OSS projects?

Because of their overview about the projects, we asked founders for their impression about the impacts of introducing UML into their project. Fig. 4.14a and Fig. 4.14b summarize the 190 answers for the two questions. A majority of respondents (65.79%) reported positive impacts, while only a few founders (<2%) encountered negative impacts. Only, 34% of the founders saw changes in the way the contributors worked after UML was introduced.

To find out more about the changes, we asked those who observed changes to describe the way the working routine had changed. We received 31 responses to the open ended question. Comments positive to UML can be summarized in following groups: i) Hiding complexity/improved overview (mentioned 18 times); ii) Improved communication/ reduced ambiguity (6 times); iii) Prevention of sub-standard implementations (5 times); iv) Improved scoping and partitioning of work (3 times); v) Improved/easier to implement designs (9 times); vi) Improved quality assurance (1 time); vii) Reduced architecture degradation (1 time).

We also received two answers describing negative changes, complaining about more work and the need for developers to learn UML notation.

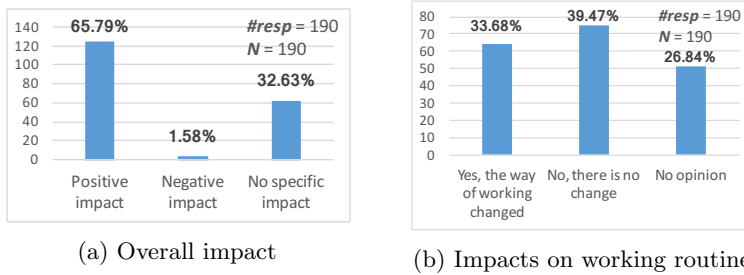


Figure 4.14: Impacts of introducing UML in OSS projects

*Results for SQ<sub>3.2</sub>: One third of respondents reported changes of the working routine due to UML, mainly in the planning phase, the development process and in communication. Most of the reported changes can be considered positive.*

#### 4.5.4.3 Can UML models help to attract new contributors?

We ask founders if they think that UML models help to attract new contributors to their projects. 190 founders answered this question. Fig. 4.15 shows the responses in detail. Only a few of the respondents (21.58%) believe that UML models can attract new contributors, while most of them think UML is not an attractive factor (47.37%).

We asked those who think UML models attract new contributors for reasons behind their thoughts. We received only 25 answers, including following arguments: a) UML models make the project and its goals easier to understand (mentioned 13 times), b) the potential of UML to help new contributors (by code comprehension) (7 times), c) visual documentation is considered attractive (3 times), and d) UML can support communication between old and new members (2 times).

It is worth mentioning that two of the projects have been based on executable UML diagrams (xtUML), therefore the diagrams were considered a magnet to contributors.

Two of the respondents who answered previously that UML is an attracting factor, mentioned additional factors, i.e., the personality, the quality of the model, and complexity of the project, e.g., “*I feel that it depends on two things: how perceptive the contributors are, and how elegantly and interesting the models [were] structured*”.

*Results for SQ<sub>3.3</sub>: Few founders think UML models attract new contributors to their projects.*

## 4.6 Discussions

In the following we discuss our insights in context of related works and implications of our results. Furthermore, we present the threats to validity.



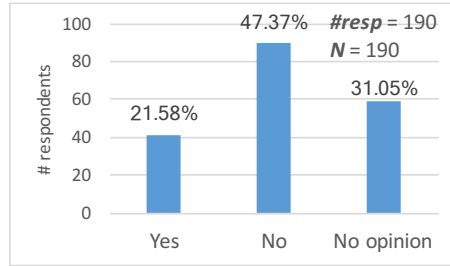


Figure 4.15: Do UML models attract new contributors to the project?

## 4.6.1 Comparison to Insights to Related Works

In this section, our observations are compared with findings from related works.

*Communication:* The finding that UML is used for communication purposes within OSS fits with observations that were already made about the use of documentation by Kazman et al. [41] and sketches Chung et al. [42]. Furthermore, the results fit with the insights of Gorschek et al. [34], who also observed a use for communication within industrial and OSS programmers.

*New contributors:* The observation that new contributors seem to benefit from the use of UML confirms the first anecdotal evidence that Chung et al. collected [42]. Gorschek et al. found similar tendencies in their survey, where the use of models was found to be higher for novices [34].

*Design and documentation:* We could uncover a main similarity in the use of UML in OSS and industry, as we observed that UML is mainly used for design and documentation, and less for code generation within OSS. Similar observations had been made for industrial usage by Torchiano et al. [32] and Forward et al. [33].

*Role splits:* However, we also found a hint of a contrast in the use of UML. While we observed that the architectures defined within UML models are often implemented by multiple developers, as it happens within industry, we also observed that in most cases all these contributors had participated in the model creation. This seems to be in contrast to the practice in many industrial cases, where those who create the models are not necessarily the ones who create the code, as, e.g., observed by Kuhn et al. [36].

Finally, we made two observations that should be further studied, also in industry. *Passive benefits:* Many participants who do not create UML models consider its existence in the project beneficial. *Partial adoption:* Many models are only partially adopted during implementation. It would be interesting to see whether this conforms or is in contrast to industrial practice.

## 4.6.2 Implications

### 4.6.2.1 OSS practitioners

*Use UML to coordinate team work!* We know that UML is used in industry within teams - communicating and coordinating their work [8]. The insights from this paper indicate that this practice might actually also work to coordinate joint efforts within OSS teams with often remotely located developers.

#### 4.6.2.2 OSS seniors

*Provide UML to support your junior peers!* In most investigated aspects the answers given by NUCs showed a slight tendency to be more positive about UML than the answers of UML contributors. Thus, it seems that models have an impact on teams that affects not just the model creators positively. We hope that OSS contributors feel motivated by these results to contributing more models. Furthermore, it seems that the usage of UML helps new contributors to get productive. This might be seen as an incentive for the introduction of UML.

#### 4.6.2.3 Industrial companies

*Adopt team-modeling!* The observed contrast that most people implementing a model also participated in its creation, might be an interesting option for industrial practice, too. Especially, when agile practices are applied, models can be taken into the loop, e.g., as part of planing during Scrum meetings.

#### 4.6.2.4 University teachers

*Promote consumption as first experience when learning UML!* Again, the mentioned slight tendency of NUCs to be more positive about UML is worth noting. It seems that the benefits of UML are more positive for consumers than for creators. This is to be confirmed in future studies. It can have today an impact on the way we teach modeling. Students still tend to learn modeling by creating models. Our results imply that it might be a good idea to let them consume models first.

### 4.6.3 Threats to Validity

In the following, we discuss internal and external threats to validity of our study as introduced by Marczyk et al. [140].

**Internal validity** Some threats that are generic to research that use GitHub data, as discussed by Kalliamvakou et al. [127], concern our study, too: First, a large amount of GitHub projects are not software development projects or have very few commits, only. Furthermore most GitHub projects are inactive (Kalliamvakou et al. guess that the amount of active projects is around 22%). To mitigate the impact of these threats on our study, we filtered the projects based on the number of commits and size. Since such filters are always just heuristics, it is probable that some of the remaining projects still are toy or educational projects. However, we consider the remaining threat acceptable, since we can assume that the vast majority of the here studied projects are *real* software development projects.

We focus on projects that do use UML only, to ensure that questioned developers have the experience of working in a project with UML. To ensure nonetheless that persons that prefer to not use UML are not underrepresented, we sent the questionnaire not just to persons who manipulated UML, but also to contributors who did not change or introduce UML files (NUCs). Therefore, we believe that our results still provide valuable insights.

**External validity** Our study focuses on OSS projects in GitHub. While we do not expect a direct generalization of our results to closed source projects, we expect them to be mostly generalizable to OSS projects. 16.29% of the survey respondents had a PhD degree. This rate is higher than industry average. We expect them to be more positive about UML, making them more likely to have answered our questionnaire. Thus, there might be a selection bias towards projects that have PhDs as contributors. We do not know whether these projects are different in nature concerning our results. However, since this concerns only 16.29% of our data points, we believe that our results are nonetheless representative.

We did not limit the domain. However, there might be a bias towards the domain that comes with the use of UML. Since we study the impact of UML, when it is used, we consider our results valuable despite the possible bias in study domains. We only have a look at UML models that are stored as specific file formats. Although, it would be better to have a look at all possible representations of UML models that exist, the selected set of formats comprehends the standard ones (.uml and .xmi) and image files, being already broad and allows a first valuable insight. Finally, in this study, we do not distinguish between UML diagram types. We therefore do not conclude for single UML types but for UML in general.

## 4.7 Conclusion and Future work

In this paper we study the use of UML in open source, in order to identify commonalities and differences to the use of UML in industry. Therefore, we performed a survey with contributors from 458 GitHub projects that include UML files. Our study delivers some first insights that might help companies to decide whether to promote UML usage in open source projects. In favor of UML are the observations that UML actually helps new contributors and is generally perceived as supportive. However, UML does not seem to have the potential to attract new contributors. Further, we found that the use of UML in open source projects is partially similar to industrial use. However, there are also differences that should be considered when joining industrial projects with open source efforts. For example, the fact that there seems to be barely a split of roles between model creator and person implementing the modeled system. Furthermore, we found that many modeled designs are only partially followed during implementation.

**Future works** We only use a part of survey responses in this study (ignoring responses of short-time projects). In the future, we plan to compare whether the results for these projects are different from the ones we found. Furthermore, we plan to use meta data to investigate whether different aspects such as size, active time, and number of contributors of a project affect the use of models and the perception of developers within the projects. Nonetheless, our findings from this study are drawn for UML in general. We would love to enrich our dataset by classifying UML diagrams by diagram type. This will enable to see whether diagram types affect the use of UML, and what UML diagrams are in widest use.

## 4.8 Appendix 1. Distribution of survey respondents by countries

Table 4.2: Respondents by countries (Top 26)

Country	No. responses	Country	No. responses
United States	72	Russia	9
Germany	49	Austria	8
France	46	China, People's Republic of	8
Brazil	35	Czech Republic	8
Spain	26	India	8
United Kingdom	21	Belgium	7
Switzerland	20	Colombia	6
unknown	15	Slovakia	6
Canada	14	Sweden	6
Italy	12	Bulgaria	5
Netherlands	11	China, Republic of (Taiwan)	5
Argentina	9	Denmark	5
Poland	9	Finland	4

## 4.9 Appendix 2. Distribution of survey respondents by continents

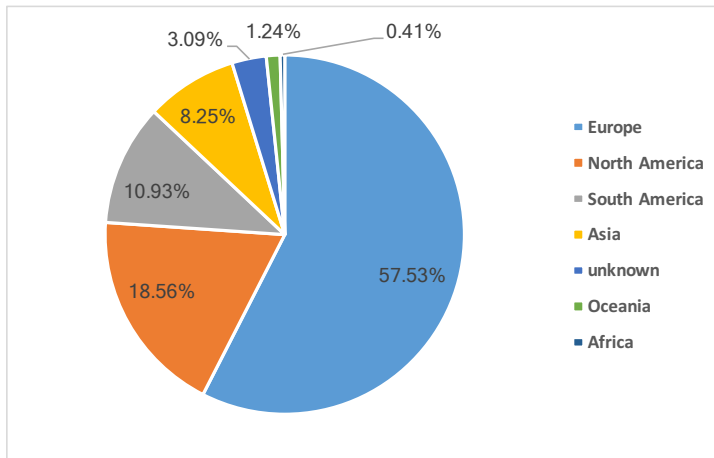


Figure 4.16: Distribution of survey respondents by continents

# Chapter 5

## Paper D

**An Automated Approach for Classifying Reverse-engineered and Forward-engineered UML Class Diagrams**

M.H. Osman, T. Ho-Quang, M.R.V. Chaudron

*In Proceeding of the 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA) (pp. 396-399), Prague, Czech Republic, August 29 - August 31, 2018.*



## Abstract

UML Class diagrams are commonly used to describe the designs of systems. Such designs can be used to guide the construction of software. However, recent studies show that UML models are at least as important in the maintenance of software: these diagrams make it easier for maintenance engineers to understand the system and plan for corrections and extensions. In practice we have identified two main types of using UML: i) FwCD: here diagrams are hand-made as part of the forward-looking development process; typical uses of these models are in communicating and guiding the design; ii) RECD: these diagrams are reverse engineered from the source code; hence these diagrams follow the construction of the implementation and mostly serve as after-the-facts documentation.

Our research is aimed at studying the effects of using UML modeling in software development. Recently, empirical studies in Software Engineering have started looking at open source projects. This enables the automated extraction and analysis of large sets of project-data. For researching the effects of UML modeling in open source projects, we need a way to automatically determine the type of UML use in a project. To support this, we propose in this paper an automated classifier for deciding whether a diagram is an FWCD or an RECD. We present the construction of such a classifier by means of (supervised) machine learning algorithms. As part of its construction, we analyse which features are useful in classifying FWCD and RECD. By comparing different machine learning algorithms, we find that the Random Forest algorithm is the most suitable algorithm for our purpose. We evaluate the performance of the classifier on a test set of almost 1000 class diagrams obtained from open source projects.

## 5.1 Introduction

Diagramming is used throughout software development lifecycle (SDLC) due to the fact that diagrams may capture diverse types of information. In the early stages of the SDLC, class diagrams may be used to represent the architectural software design. As development progresses, class diagrams can be used to represent information that is closer to the construction of the system (design level class diagram). During or after the implementation of source code, a class diagram may be recovered using reverse engineering techniques. Such a reverse engineered class diagram is closely based on the source code and reflects the fine-grain implementation structure of software systems [141].

Hebig et.al. [137] present Lindholmen dataset which is a repository of UML diagrams built to serve as an informative collection of UML models. This repository contains a large amount of UML models that are gathered from the open source software community. This repository holds more than 24,000 UML class diagrams and includes links to the projects on GitHub where the diagrams were found. As such it forms a valuable resource for empirical studies on projects that use some forms of UML modeling. The classification of these diagrams is needed to assisting the research of diagram for various purposes.

**Goal:** In this paper, we focus on providing information on type and purposes of each class diagram. This study aims at providing an automated classification model for classifying Forward Engineered Class Diagram (FwCD) and Reverse Engineered Class Diagram (RECD). This study use the following definitions for the FwCD and the RECD:

**Definition 1.** Forward Engineered Class Diagram (FwCD): *“Forward engineering is the traditional process of moving from high-level abstractions and logical, implementation-independent designs to the physical implementation of a system.”* [53]

**Definition 2.** Reverse Engineered Class Diagram (RECD): *“Reverse engineering is the process of analyzing a subject system (especially its implementation) to identify the system’s components and their interrelationships, and create representations of the system in another form or at a higher level abstraction.”* [53]

These diagrams have different perspectives and purposes in software development: The FwCD are mainly for describing the high level design structures of a system (as illustrated in figure 5.1). Meanwhile, the RECD purpose is more oriented towards describing the structure of the implementation (as illustrated in figure 5.2). We apply supervised machine learning techniques as the method for the classification of diagrams. We use a dataset of 999 class diagrams that are collected from Lindholmen dataset. In order to obtain a ground truth, this dataset is labelled by experts that have experience in working with UML diagrams. The classification features are extracted based on (i) characteristics of the models that were mentioned by the expert while classifying into FwCD and RECD and, (ii) the work by Hafeez and Chaudron [142], and (iii) Nugroho and Chaudron [143]. We evaluate 11 classification algorithms that cover diverse type of algorithms in supervised machine learning, in order to select the best classification algorithm for this problem.

**Contribution.** The contributions of this study are the following:



- Identification of features that can be used to classify FwCD and RECD diagrams
- A suitable machine learning algorithm for classifying FwCD and RECD
- A dataset with ground truth for classifying FwCD and RECD
- A comparative analysis of the performance of various machine learning algorithms for our problem

This paper is structured as follows. Section 5.2 discusses the related work. Section 5.3 describes the research questions and Section 5.4 explains the approach. We present the analysis of results in Section 5.5. The discussion and future work are presented in Section 5.6. This followed by conclusions in Section 5.7.

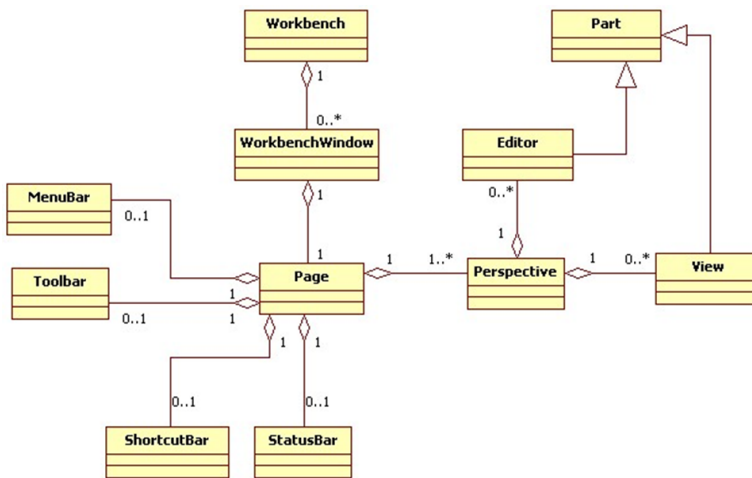


Figure 5.1: FwCD Example

## 5.2 Related Work

To our best knowledge, there is no work that is directly targeted at the automatic classification of FwCD and RECD. Therefore, we broaden our discussion to works that have used class diagram information (e.g. metrics, measures) and machine learning classification algorithms for classification or prediction purposes.

Maneerat and Muenchaisri [144] proposed a method for predicting bad-smell from software design model. 27 software metrics were used in this study that consists of Basic Class Employment (basic class information), Complexity, Diagrams, Inheritance, MOOD [145], model size and relationship. They used seven (7) datasets that were created by extracting these metrics from reverse engineered class diagrams. Cross-validation was used to assess the prediction performance and for preventing over-fitting.

Halim [146] proposed a method to Predict fault-prone classes using the complexity metrics of UML class diagram. The prediction models were built

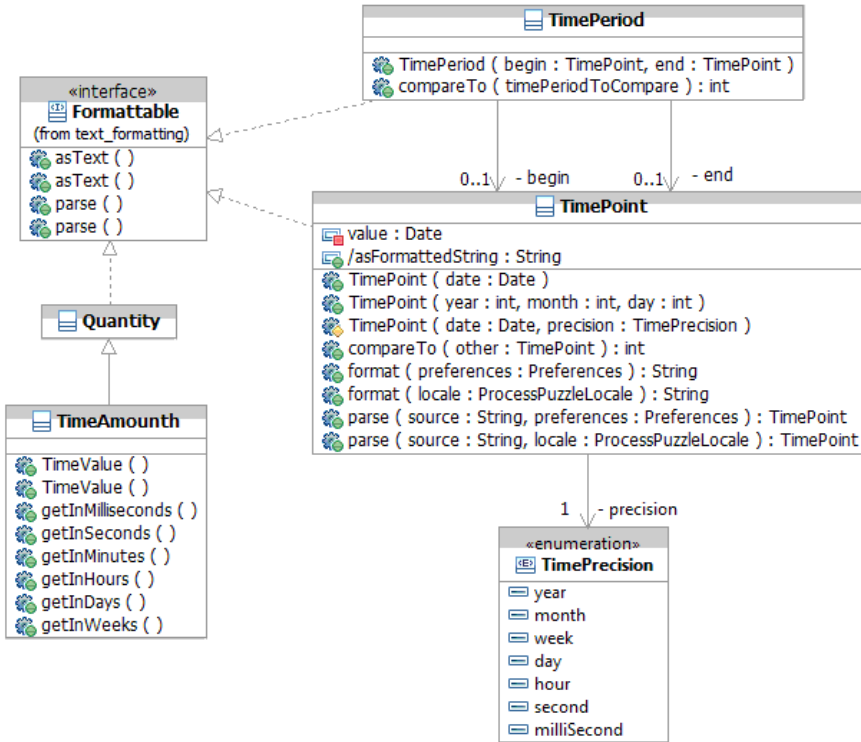


Figure 5.2: RECD Example

using two classification algorithm i.e. Naive Bayes and k-Nearest Neighbors. The models were validated by using 10-fold cross-validation and the performance was assessed by using Receiver Operating Characteristics (ROC) curve analysis.

Bagheri and Gasevic [147] investigated through controlled experimentation whether a set of structural metrics can be good predictors (early indicators) of the three main sub-characteristics of maintainability: analyzability, changeability, and understandability. The analyzed measures for software product line features model consisted of size measures (e.g. Number of Features (NF), Number of leaf features (NLeaf)), Structural Complexity Measures (e.g. Cyclomatic Complexity (CC), Flexibility of Configuration (FoC), Number of valid configuration (NVC)) and Length Measure (Depth of tree (DT)). They built four (4) prediction model by using J48, ID3, CART and Logistic Regression. Bagheri and Gasevic conclude that NLeaf, NVC, CC and FoC are the most suitable features for predicting the aforementioned sub-characteristics of maintainability. Nugroho also performed a study on predicting defects based on diagram metrics [143]. He however, used metrics that are indicators of the level of detail used in the diagram. His study showed that higher level of detail in UML models correlates with fewer defects in source code. This suggests that when our classifier uses level of detail metrics, it might also be used in quality assurance of models.

Osman et. al. [148] proposed an approach to condensed reverse engineered class diagram by using machine learning technique. They used object-oriented design metrics i.e. size measures (e.g. number of classes, number of operations) and coupling measures (e.g. Import Coupling Attribute, Dependency out). The datasets were collected from open source java projects. Nine (9) classification algorithms involved in their experiments in order to find the best model for classifying key classes in reverse engineered class diagram. Based on this work, Thung et. al. [149] improved the classification result by adding Network measure (e.g. Baycenter, PageRank). Recently, Yang et. al. [150] improves the classification performance as well as the effort of learning.

These related works show the usage of class diagram information or object-oriented design metrics for assessing (which is a form of classification) UML diagrams.

## 5.3 Research Questions

This section describes the research questions of this study that will be answered in section 5.5.

**RQ1:** Which features of UML diagrams are influential predictors for classifying diagrams into FwCD and RECD?

**RQ2:** What are suitable classification algorithms for classifying FwCD and RECD?

## 5.4 Approach

This section describes our overall approach (illustrated in Figure 5.3) that consists of (i) Data Collection (ii) Features Extraction & Creating Ground truth, (iii) Model learning, and (iv) Evaluation of Results.

### 5.4.1 Data Collection

Data preparation process consists of two (2) main activities: UML Class Diagrams (images) Collection and Image to XMI Conversion.

#### 5.4.1.1 UML Class Diagrams Collection

The main input for this study is UML class diagram images. These class diagram images were collected from Lindholmen dataset [137]. It is noted that at the time of collecting data for this study, the Lindholmen dataset were being built. Therefore, we were able to scan 4443 projects and collect 2000 class diagrams that are stored in various image file formats. In the next step, we refined the dataset by applying a number of filters. Firstly, the images should have a reasonable quality, as they serve as input for an image-processing conversion tool (see section 5.4.1.2) and the tool expects reasonable-quality images in order to generate reliable result. Therefore, we had to exclude images

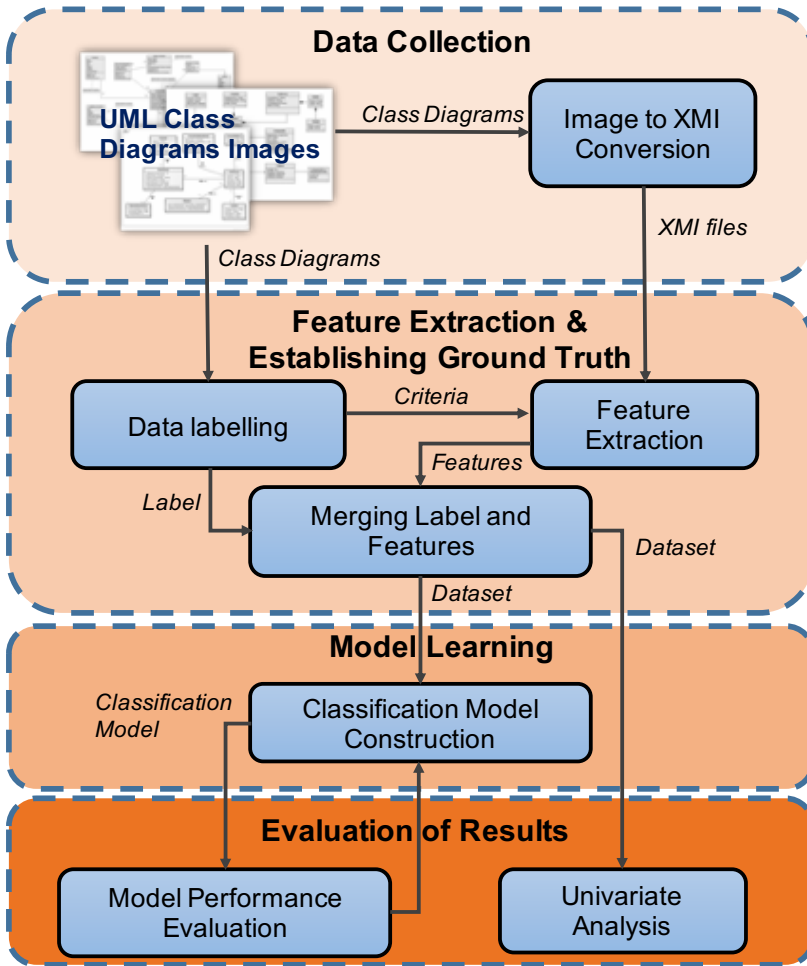


Figure 5.3: Overall Approach

that have a small-size and a low resolution. Secondly, to avoid data redundancy, we removed duplicate images from the dataset. In the end, we finalized a list of 999 class diagram images as the dataset for this study. The list of the selected class diagram images can be found at [151].

#### 5.4.1.2 Class Diagram Image to XMI Conversion

Since the collected class diagrams are in image formats, it is necessary to extract and store the content of the diagrams in a standard UML file format which is XMI (XML Metadata Interchange). The conversion of UML class diagrams in image formats into XMI format is done by using the Image2UML tool [95]. As the tool does not extract methods parameters, we modify the source code (by adding new code) in order to capture that information. Integration tests are performed to ensure the new feature works well. Furthermore, we improve the Image2UML tool's XMI file structure for better representation and also to suit our purposes.

## 5.4.2 Feature Extraction & Establishing Ground Truth

This subsection consists of three (3) activities: Data Labeling, Feature Extraction, and Merging Label and Features.

### 5.4.2.1 Data Labelling

Supervised machine learning needs a labelled data (for learning purposes). Thus, the dataset of 999 class diagrams should be labelled either FwCD or RECD. Since there is no explicit rule on how to distinguish those diagrams, the labelling activity was done manually by three (3) selected UML experts who have at least five (5) years using UML class diagrams for different purposes. The labelling process consists of two (2) phases:

[a] *Define FwCD and RECD characteristic*

All the experts gathered together in a brainstorming session to outline the characteristics of FwCD and RECD. This session aims at synchronizing the general characteristics of FwCD and RECD among the experts and at finding information to formulate the classification features. After the experts defines the FwCD and RECD characteristics, we randomly select 30 class diagrams images from the dataset and let the experts apply the defined characteristics to classify the diagrams. In this way, we can investigate whether (i) the defined characteristics could actually help on classifying FwCD and RECD, and (ii) the experts reaching agreement on their views toward FwCD and ReCD.

[b] *Manual Classification*

Each expert is randomly assigned a set of 323 class diagrams. A simple webpage ([151]) is created to assist the experts when performing classification. The webpage displays class diagram images together with the defined FwCD and RECD characteristics. The experts are also asked to provide the rationale behind their decision by choosing among the criteria and/or using a free text box to indicate their thought if the characteristics do not match. For each diagram, every expert needs to classify it into different categories: “Forward Design”, “RE Design”, “For Discussion” and “Skip For Now”. Diagrams in different categories than “RE Design” and “Forward Design” are discussed in follow-up group meetings. The loop of individual classification and group discussion is continued until all class diagram images are classified into FwCD and RECD.

### 5.4.2.2 Feature Extraction

Based on Osman et. al. [142], there are several weaknesses of RECD produced by commercial CASE tool, e.g. high number amount of information, unable to detect several types of relationship and etc. These weaknesses are our basis for selecting our classification features. Furthermore, we also used the expert judgment on FwCD and RECD that we observed during the data labelling activity as the basis of formulating the classification features. The detail explanations on selected classification features are illustrated in Table 5.1.

Table 5.1: List of Features

No	Features	Data Type	Description
1	noCls	Numeric	Count number of class(es) in the class diagram
2	noOper	Numeric	Count total number of classes in the class diagram
3	noAttr	Numeric	Count the number of attribute in the class diagram
4	noPara	Numeric	Count the number of operation in the class diagram
5	extOperPara	Nominal (Binary)	“true” if the operation parameter exist and “false” if it is not exist
6	noAssociation	Numeric	Count the number of operation in the class diagram
7	noAssocType	Numeric	Count the number of type of as- socation type exist in the class diagram
8	extOrpCls	Nominal (Binary)	“true” if the orphan classes exist and the “false” if it is not exist
9	noOrpCls	Numeric	Number of orphan classes in a class diagram
10	avgAttrCls	Numeric	Average attribute per class
11	avgOperCls	Numeric	Average operation per class
12	avgAssocCls	Numeric	Average operation per class
13	avgParaOper	Numeric	Average parameter per operation
14	avgOrpCls	Numeric	Average orphan classes
15	maxAttrCls	Numeric	Select the highest number of at- tribute in a class in the class di- agram
16	maxOperCls	Numeric	Select the highest number of op- eration in a class in the class di- agram
17	isFWD	Nominal (Binary) - Class Label	“Yes” if the class diagram is for- ward design and “No” if it is a reverse engineered class diagram

### 5.4.2.3 Merging Labels and Features

In this activity, we extract all the features (listed in Table 5.1) from the class diagrams (from its XMI files) into a data file (csv format). Then, we merge the data file with the label information (isFwd - “Yes” or “No”). The data file that consists of 999 class diagrams classification features and its label is the dataset for this study.

### 5.4.3 Model Learning

In this section, we explain the classification model construction. This activity is supported by Waikato Environment for Knowledge Analysis [WEKA] tool [152].

#### 5.4.3.1 Classification Model Construction

Mining data is experimental. There is no algorithm that fits all situations and all purposes. Therefore, at first, we need to find the suitable machine learning algorithm(s) for our purposes and our dataset.

We start the selection of machine learning algorithm by conducting an exploratory experiment on a range of machine learning algorithms. In this exploratory experiment, we are interested to find not one suitable classification algorithm but a set of classification algorithms that are suitable for the dataset. Therefore, this experiment looking at the various type of machine learning classification algorithm.

The algorithms in this experiment are selected from the different set of algorithms representative for different approaches. For example, Decision Trees, Stumps, Tables and Random Trees or Forests all divide the input space up into disjoint smaller sub-spaces and make a prediction based on the occurrence of positive classes in those sub-spaces. K- Nearest Neighbour (k-NN) and Radial Basis Functions (RBF) Networks are similar local approaches, but the sub-spaces here are overlapping. In contrast, Logistic Regression and Naive Bayes model parameters are estimated based on potentially large numbers of instances and can thus be seen as more global models [148]. According to Holte [153], there is a possibility that a simple algorithm works well in a dataset. Hence, OneR is selected to represent the simplest classification algorithm compared to the algorithms mentioned above. The detail explanations of the aforementioned algorithms can be found at [152]. We use the result of ZeroR as the baseline (only on the accuracy measurement). ZeroR shows the probability of a guess of whether a class diagram is ReCD or Fwd. If the result of a classification algorithm approaching this baseline, it means that the classifier does not make a significant contribution in classifying the ReCD class diagram.

The 10-fold cross-validation is used for evaluate the classification model performance. To ensure a more accurate validation result, the 10-fold cross-validation is repeated ten (10) times for every classification model. The average value will be used as the final result.

### 5.4.4 Evaluation of Results

The process consists of two (2) activities: Univariate Analysis and Classification Performance Evaluation.

#### 5.4.4.1 Univariate Analysis

Prior to processing the data, we explore the behaviour of the data by measuring the predictive power of the predictor (classification features). To measure predictive power of predictors, we used the information gain with respect to the class [152]. Univariate predictive power means measuring how influential a single predictor is in prediction performance. Normally, the result of this analysis is used to select the suitable predictor(s), however, this study uses this analysis for the data exploration. We aim at discovering the most influential predictor in the dataset. We use WEKA's Information Gain Attribute Evaluator (InfogainAttrEval) in conducting this experiment. The WEKA InfoGain Attribute Evaluator produces a value from 0 to 1. The higher value of InfoGain (close to 1) denotes a stronger influence of the predictor.

#### 5.4.4.2 Classification Performance Evaluation

In general, we use three (3) evaluation measures to evaluate the classification algorithms performance i.e. (i) Percentage of correct (accuracy), (ii) Precision and, (iii) Recall.

- [a] *Correctness (or Accuracy)* is defined as the ratio of the number of correctly classified items to the total number of items [154].
- [b] *Precision* is a function of true positives and examples misclassified as positives (false positives) [155].
- [c] *Recall*: is a function of its correctly classified examples (true positives) and its misclassified examples (false negatives) [155].

The aforementioned measurements are the basic evaluation measures that are used in this study. If required, we extend this evaluation into more detail measures such as F-Measure [156] and Area Under Receiver Operating Characteristic (ROC) Curve (a.k.a AUC) [157].

## 5.5 Result and Findings

This section evaluates the performance of the selected features and the classification algorithms. Each of the following subsection answers the research questions.

### 5.5.1 RQ1: Analysis of Selected Features

The InfoGain measure is an indicator of how significant a feature is for performing a classification-task. A 0-value indicates no significance, and a higher value indicates a higher significance. Table 5.2 shows InfoGain results for our 16 features. (predictors) produce InfoGain score  $>0$ . This result shows that every single feature used in this study has some predictive power. In order words, every single feature influences the classification model. The average number of operation parameters (*avgParaOper*) is the most influential predictor. This is followed by the number of parameters (*NoPara*) and the existence of parameter (*extOperPara*). These three (3) most influential features are related to the



Table 5.2: InformationGain Attribute Evaluator Results

No	Predictor	InfoGain Value
1	avgParaOper	0.3191
2	noPara	0.2897
3	extOperPara	0.2371
4	avgOperCls	0.2249
5	maxOperCls	0.1602
6	avgAssocCls	0.1597
7	noCls	0.1319
8	noAssociation	0.1304
9	noOper	0.1265
10	noOrpCls	0.0858
11	avgOrpCls	0.0668
12	avgAttrCls	0.0605
13	noAttr	0.0551
14	maxAttrCls	0.0377
15	extOrpCls	0.0238
16	noAssocType	0.0118

operation parameters. Meanwhile, the other most influential features are the average number of operations per class (*avgOperCls*) and the maximum number of operation per class (*maxOperCls*). Both features are related to the operations in class diagrams. Thus, this result indicates that the class diagram operations plays a major role in classifying the RECD and FwCD. More specifically, the information about operation parameters is the best indicator to classify UML diagrams into FwCD and ReCD.

On the other hand, not all features related to class relationships strongly influence the classification performance. The average association relationship per class (*avgAssocCls*) and the number of association (*noAssociation*) have a high InfoGain value. Meanwhile, the number of orphan class (*noOrpCls*) and the average of orphan class (*avgOrpCls*) moderately influence the classification performance. The existence of orphan class (*extOrpCls*) and the number of association types show the weakest influence in classification performance from the class relationship category.

This result also shows that the features related to attributes in class diagrams have a weak influence in classification performance. The number of attributes (*noAttr*) and the maximum number of attribute (*maxAttrCls*) have a rather low InfoGain value.

## 5.5.2 RQ2: Classification Model Performance

This subsection shows the evaluation of the classification model performance. The evaluation is based on the classification performance score illustrated in Table 5.3.

In this evaluation, we use two (2) baselines measurements: (i) ZeroR and (ii) OneR. The first baseline is ZeroR. For ZeroR, the benchmark only uses the accuracy measures. The accuracy measure for ZeroR means a probability of a random guess (in terms of percentage) for the majority of classes in the dataset.

Table 5.3: Classification Performance

<b>Performance Measure</b>	<b>Acc.</b>	<b>Prec.</b>	<b>Recall</b>	<b>F-Measure</b>	<b>AUC</b>
OneR	88.01	0.94	0.91	0.92	0.84
Decision Table	88.33	0.93	0.93	0.93	0.93
Naive Bayes	88.10	0.97	0.88	0.92	0.91
RBFNetwork	89.32	0.95	0.91	0.93	0.92
Logistic Reg.	88.90	0.94	0.93	0.93	0.94
SVM	87.28	0.88	0.97	0.93	0.71
K-NN (1)	89.16	0.93	0.93	0.93	0.88
K-NN (5)	87.89	0.93	0.92	0.92	0.93
Decision Stump	87.69	0.98	0.87	0.92	0.89
J48	88.59	0.94	0.91	0.93	0.85
Random Tree	87.89	0.92	0.93	0.93	0.81
Random Forest	90.74	0.95	0.93	0.94	0.96

Our dataset is imbalanced: the majority of the class diagram in the dataset is RECD. The accuracy value for ZeroR is 80.68% which means without taking any features for prediction, the probability of correct prediction is 80.68%. Thus, classification algorithms should perform better than ZeroR because otherwise they would not make any significant improvement. The results show that all classification models produce a significant improvement compared to ZeroR. The relative improvement of accuracy values ranges from 7% to 10%.

The second baseline is OneR. The OneR classification algorithm uses only one (most influential) feature to construct the classification model. This benchmark experiment is conducted because according to Holte [153], there is a possibility that a simple algorithm works well in a dataset. OneR represents the simplest classification model. Thus, complex classification algorithms should perform better to produce a significant improvement because they require greater computational effort (e.g. for extracting all features). The results show that the performance of SVM, k-NN (5), Decision Stump and Random Tree are slightly lower than OneR (based on accuracy value). Thus, we exclude these algorithms from further evaluation.

After benchmarking the classification algorithms against the baselines, we compare the classification algorithms' performance based on precision and recall. This evaluation only involved the classification model constructed using Decision Table, Naive Bayes, RBF Network, Logistic Regression, k-NN(1), J48 and Random Forest. The results show that Naive Bayes and Decision Stump produce a high precision score. However, both classification algorithms scored relatively low on recall. Decision Table, Logistic Regression, k-NN (1) and Random Forest show a balance score between precision and recall. Hence, based on this result, Decision Table, Logistic Regression, k-NN (1) and Random Forest are suitable algorithms for our problem and dataset. In order to find the best classification algorithms for the dataset, we compare the classification performance score using the F-Measure. Based on the F-Measure result, the Random Forest is the best performing classification algorithm for our purpose.

## 5.6 Discussion and Future Work

In this section, we reflect on our findings and outline future work based on (i) the feature selection, (ii) the dataset and, (iii) the classification algorithms. We discuss the threats to validity at the end of this section.

### 5.6.1 Feature Selection

In this study, we have selected 16 features that we believe influential in classifying RECD and FwCD. These features covered a high-level structural information as well as information about level of detail of class diagrams. For example, we use the number of classes (*noCls*), the number of attributes (*noAttr*) and the number of operations (*noOper*). This type of information can be classified as the high-level structural information of class diagrams. Meanwhile, the average association per class (*avgAssocCls*), average parameter per operation (*avgParaOper*) and average attribute per class (*avgOperCls*) are the examples of features that related to the level-of-detail of class diagrams.

For future work, we would like to enhance these classification features by exploring features such as information about types of class relationship, complexity metrics, use of inheritance (e.g. number of child, depth in inheritance tree), attribute and operation visibility (e.g. public, private), and possibly layout-related feature and etc.

### 5.6.2 Dataset

From the result, it is obviously shown that the information about the operation in class is the most influential feature in classifying RECD and FwCD. The top five (5) most influential features i.e. average parameter operation (*avgParaOper*), number of parameter (*noPara*), existence of operation's parameter (*extOperPara*), average operation per class (*avgOperCls*) and maximum number of operation per class (*maxOperCls*) are features related to the operations of classes.

Initial observations of the dataset showed that most of the class diagrams that were labelled as RECD had a higher number of parameters and level of detail for class operation. This points in the direction that a high level of detail of the operations in a class diagram is a strong indicator that the class diagram is an RECD diagram. However, the InfoGain Attribute Evaluator (InfoGainAttrEval) only evaluates the influence of individual features in isolation, and InfoGain does not add up linearly when combining features. InfoGain does not count the influence of a group of features. In the future, we would like to evaluate the influence of a group of features in classifying FwCD and RECD. We also plan to analyze the correlation between features and come out with better predictor/feature set.

### 5.6.3 Classification Algorithm

We have selected 11 classification algorithms as candidates to be used in our automated classification of FwCD and RECD. These classification algorithms represent diverse type of supervised machine learning approaches. In this study, we found that Decision Table, Logistic Regression, Nearest-Neighbour (k-NN(1))

and Random Forest are suitable classification algorithms for the dataset. We conclude that the Random Forest is the best performing classification algorithm for this purpose. Random Forest scored 0.95 (precision), 0.93 (recall) and 0.94 (F-Measure). Furthermore, we compared this classification algorithm with other algorithms using AUC score. Based on the AUC score, the Random Forest still scores the highest (0.96) compared to Decision Table (0.93), Logistic Regression (0.94) and k-NN(1)(0.88).

We took a step forward to find out why the classification algorithm (we focused on Random Forest) failed to classify several class diagrams. Figure 5.4 shows an example of an FwCD was classified as RECD (False Negative). We found that the class operation's level of detail have a high influence in classifying FwCD and RECD. Hence, the improvement on classification features as mentioned in section 5.6.1 is required to overcome this issue.

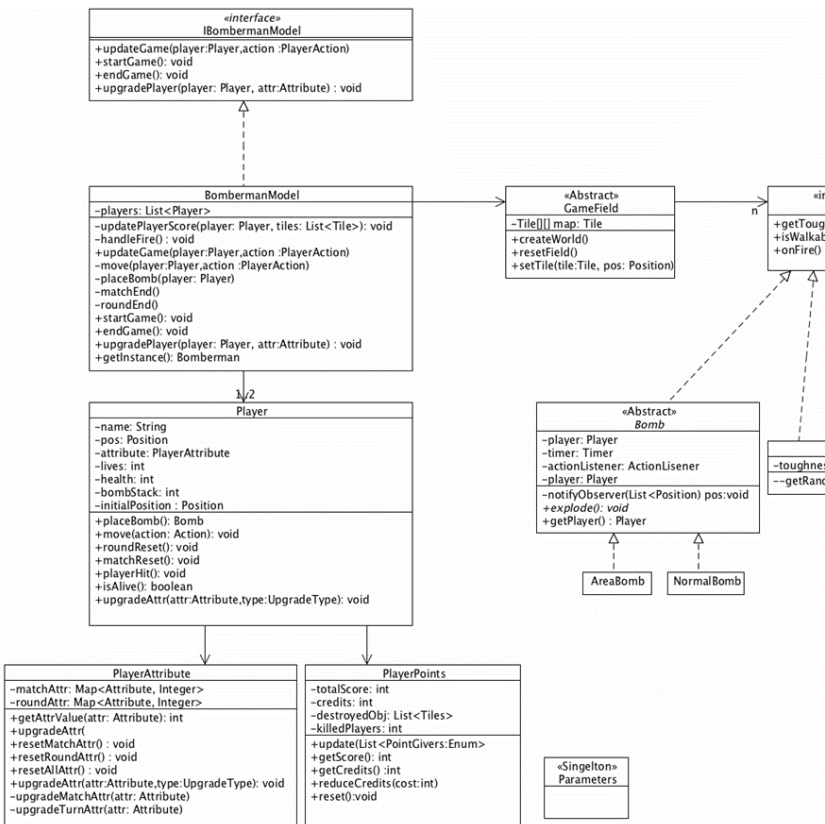


Figure 5.4: An Example of False Negative - FwCD that is classified as RECD

Even though the classification model can be considered to produce a reasonably good classification performance, we see there is a possibility to enhance the classification performance by the following:

- *Reconfigure the classification parameter*

In our study, we only used the default configuration that is provided by WEKA. We believe it is possible to enhance the classification performance

by changing the classification algorithms' parameter configuration.

- *Combination of classification algorithms*

Since there are several classification algorithms that seem suitable for our purpose, in the future, we would like to combine or stack several classifiers.

Other future works are as follows : (i) enhance the practical usability of a tool based on our classification model, and (ii) extend our approach to larger datasets. Moreover, we would like to explore (iii) classifying class diagrams into application domains by using text mining techniques.

## 5.6.4 Threats to Validity

This subsection describes the threats to validity of this study. We consider Internal validity, External validity and Construct validity.

### 5.6.4.1 Threats to Internal validity

Our raw data (class diagrams) is collected from image-type diagrams. We used previously developed image-recognition techniques [95] to convert these diagram into XMI format to enable our feature extraction. There is a possibility of inaccuracy of the extracted information. A random check has been performed on a sample of the extracted models to make sure the converted diagrams are correct. If the accuracy of the image recognition would be improved, then this most likely would allow some improvements to the performance of the classifier, because the image recognizer currently extracts a large amount of information from the diagrams.

### 5.6.4.2 Threat to External Validity

In our study, class diagrams are collected from 4443 Github projects which is a very small portion of more than 12 million projects on Github. Therefore, we cannot claim that our dataset is representatitive for all open source projects (on GitHub). The generalization of our approach would benefit from extending the training and testing datasets.

### 5.6.4.3 Threat to Construct Validity

The dataset is quite imbalanced i.e. the percentage of RECD diagrams is much more than the FwCD ones. Thus, is a possibility that the classification model has a bias towards RECD. We minimized this threat by performing 10-folds cross-validation ten (10) times (repetition). For each round, the data is randomized. We take the average of all 10 rounds as the final result.

## 5.7 Conclusions

This work presented the construction and evaluation of an automated classifier for differentiating forward-designed- (FwCD) from reverse-engineered (RECD)

class diagrams. This classifier was constructed using machine learning algorithms, We investigated various properties of class diagrams as features of our classifier. Our features covered structural information of class diagrams as well as a features that relate to the level of detail of class diagrams.

In this study, we have shown that each of the feature that we consider is influential in classifying FwCD and RECD. The features that relate to parameters of operations are the most influential features for this classification purposes. In terms of classification algorithms, we found that out of the 11 classification algorithms that were used in our experiments, only four (4) algorithms are suitable for our dataset (i.e. Decision Table, Logistic Regression, k-NN (1) and Random Forest). Random Forest is the most suitable classification algorithms for our classification model. The classification model scored 90.74 (accuracy), 0.95 (precision), 0.93 (recall), 0.94 (F-Measure) and 0.96 (AUC).

As for the conclusion, this study has formulated an automated classification model to classify FwCD and RECD. The classification model performed reasonably well based on the scores benchmark. As part of our future work, we would like to apply this model to our recently collected corpus of class diagrams (in total 24000+) from Lindholmen dataset. Through this research, we expect to get an understanding of the different ways in which UML diagrams are used in open source projects and ultimately an understanding of the effectiveness of various modeling and documentation practices.

# Chapter 6

## Paper E

Challenges and Directions for a Community Infrastructure  
for Big Data-driven Research in Software Architecture

T. Ho-Quang, M.R.V. Chaudron, Regina Hebig, G. Robles

*Accepted as a chapter in the book “Model Management and Analytics for Large Scale Systems”, To be published by Elsevier (Expected release date: November 1, 2019), Edited by Bedir Tekinerdogan, Onder Babur, Loek Cleophas, Mark van den Brand and Mehmet Aksit, 2019.*





## Abstract

Research into software architecture and design has become more and more prominent since the 1990's. Since then, companies started reporting how software architecting helped them to tackle various challenges in system-design, especially related to system-level quality properties such as scalability and maintainability. Academic research in software architecture has focused on several areas, including architecture description through views and architecture description languages, and on methods for evaluating architectural designs. While much of the contributions of research in software architecture was inspired by industrial experiences, little of the research was validated beyond individual case studies. Many scientific disciplines are currently harvesting fruits from large scale data collection about their subjects of study. Therefore, this chapter contributes a discussion of challenges and directions for big-data driven studies of software architecture. Given the large amount of effort that is needed for this type of research, a promising direction is to look into a community-based infrastructure for enabling and supporting this type of research. We share lessons learned through building various tools that could form building-blocks in such an infrastructure. Based on these, we synthesize a reference architecture for creating such a community-wide infrastructure for big-data-based research in software architecture.

## 6.1 Introduction

Research into software architecture and design blossomed in the 1990's. At that point in time, many organisations were experiencing the exponential increase in the size of their software systems. Almost at the same time, projects were struggling with the increasing amount of changes to the software that needed to be handled. The practices of software architecting were proposed as one of the main tools for addressing both the challenges of scale and evolvability. Academic research in software architecture has focused on several areas, including architecture description through views and architecture description languages, and on methods for evaluating architectural designs. While much of the contribution of research in software architecture was inspired by industrial experiences, little of the research was validated beyond individual case studies. Many scientific disciplines are currently harvesting fruits from large scale data collection about their subjects of study. Indeed, such 'big data' promises insights by finding patterns by analysing large data sets. Therefore, this chapter contributes a discussion of challenges and directions for big-data driven studies of software architecture. We discuss lessons from various projects that focus on particular questions that are building blocks in the overall landscape of big data for empirical software architecture research. Based on these lessons, we synthesize a proposal for a reference architecture for a community-wide infrastructure for evidence-based research in software architecture and design.

The structure of this chapter is as follow: In Section 6.2, we discuss existing work related to our research topic. Then, we present our experiences on building, maintaining and sharing a big corpus of models (Section 6.3). This is followed by a discussion on the challenges when conducting empirical studies in software architecture (Section 6.4). The discussion reflects our observations on research in the field as well as our experience building the Lindholmen dataset of UML software designs. In Section 6.5, we list nine requirements for building such an infrastructure. Lastly, we propose a reference architecture for such an infrastructure (called *CoSARI*), and our on-going efforts on building this (Section 6.6).

## 6.2 Related Work

In this section, we discuss works that are in various ways related to the topic of this chapter. Empirical data of software architecture serves as basis for any evidence-based research in the field. Therefore, at first, we summarize existing corpora of software architecture artefacts. The software architecture artefacts/documentation (SAD) can be split into software modeling artefacts (such as UML models, DSLs, etc.) and textual-based artefacts (such as software architecture specification, etc.).

The desired infrastructure should ultimately support researchers with not only empirical data on software architecture but also with means for analysing the data and sharing the analyses. Therefore, we discuss existing work on discovering architecture knowledge and review some scientific workflow systems as a reference for building the infrastructure.

### 6.2.1 Existing Corpora of Software Modelling Artefacts

Störrle et al. introduced the Software Engineering Model Index (SEMI) which contains a list of contemporary model repositories [24]. We take this as a starting point for our search of software modeling corpus. In fact, 3 out of 8 corpora to be reviewed in this section are listed in SEMI. In the paper, the authors also outline four main challenges when building a successful model repository: i) Archiving (“How to archive data with very high reliability, for very long time, yet readily accessible, and economically viable?”), ii) Access Support (“How to search for models?”), iii) Intellectual Property (“How to manage intellectual property such as models?”), and iv) Incentives (“How to motivate researchers/practitioners to publish their models?”).

The *BPM Academic Initiative (BPM AI)* is a platform where business process models are shared for teaching purposes [30]. A business process model is defined as a set of business activities and execution constraints between these activities [31]. It can be used to describe complex interactions between business partners and to indicate related business requirements on an abstract level. Currently, BPM AI claims to host 29,285 business process models in various machine-readable formats. The dataset has however not been updated since 2012. The process of collecting models is not clearly mentioned; apparently, most of the models in the dataset derive from students as part of modeling assignments.

The *Repository for Model Driven Development (ReMoDD)* is created to support researchers and practitioners in sharing exemplar models and other modeling practices [25]. Currently, it contains around 90 modeling artefacts, including models in different modeling languages and artefacts of some MDD conferences. Models are stored in various formats, mostly PDF but also some in XMI.

The *Open Models Initiative (OMI)*<sup>1</sup>, similar to ReMoDD, offers a platform that allows researchers and practitioners to share models. It is currently hosting around 70 models stored mostly in image-formats. There is no report on whether the models are derived from industrial or academic contexts.

Karasneh et al. used a crawling approach to automatically fill an online repository with so far more than 700 model images<sup>2</sup> from Google Image Search [26]. Registration is not required in order to get access to the repository. The repository also provides a comprehensive search which could be used to form and share subset of the data.

Mengerink et al. collected a data set of 9,188 OCL expressions derived from 504 EMF meta-models in 245 GitHub repositories [27]. To this end, the authors firstly performed a couple of GitHub searches, then downloaded all *.ecore* and *.ocl* files in the result list, then removed all duplicated files and finally parsed all the unique files to extract OCL expressions.

Basciani et al. built MDEForge as a web-based modeling platform which aims at fostering a community-based modeling repository [28]. The number of meta-models hosted in this platform is not available.

GenMyModel<sup>3</sup> is a web-based online tool that supports collaborative mod-

---

<sup>1</sup><http://openmodels.org>

<sup>2</sup><http://models-db.com/>

<sup>3</sup><https://www.genmymodel.com/>

eling for UML, BPMN, RDS, and flowcharts [29]. At the time of writing, GenMyModels claims to host about 777,000 diagrams. However, it is not clear how many of these diagrams are open to public access and how many are private.

The Lindholmen dataset<sup>4</sup> contains more than 93,000 UML models from more than 24,000 GitHub repositories [137]. Different from the above-mentioned corpora, the Lindholmen dataset also includes meta data of the projects where UML models are used. This enables researcher to study the use of UML models in their context, e.g. How frequent and in which phase of the project are the models updated? etc. The UML models are collected from GitHub using complex settings of tools and technologies (such as image processing). The models are provided in various formats, mostly in .uml, .xmi and image files. This is currently the biggest data set of UML models.

## 6.2.2 Other Software Architecture Collections

Ding et al. present the retrieval and analysis of a collection of SADs obtained from 108 open source projects [38]. We have reviewed this document and have run into two issues. Firstly, many SAD links expired, resulting in 404-browser errors. This means that either the document was moved or deleted, thus, the URL pointing to it was faulty. Secondly, for the SAD documents that we could find, we found they were of ‘mixed quality’: some documents were identified as SAD but are almost empty or contain only little text.

The well-known book “The Architecture of Open Source Applications” is a collection of SAD of 48 open source applications [158]. In particular, each chapter describes the architecture of an open source application: how it is structured, how its parts interact, etc. It is noted that there is no uniform representation of the architectures. In fact, every chapter in the book has its own structure and uses different kinds of diagramming notations.

## 6.2.3 Mining Architectural Knowledge

Another relevant area of empirical studies in software architecture focuses on the notion of ‘mining’ artefacts for architectural knowledge.

The work by Soliman et al. aims to mine architectural knowledge from natural language sources, in particular from StackOverflow [159]. They apply advanced natural language processing and machine learning algorithms for the recognition and classification of sentences.

Musil et al. describe a novel architecture knowledge management approach with similar objectives: use information retrieval and natural language processing to extract architectural knowledge about systems from all documents available in project repositories [160]. Their approach *CAKI* (stands for Continuous Architectural Knowledge Integration) consists of 4 stages:

- Information acquisition - where relevant data is to be collected.
- Architecture knowledge synthesis - where architecture knowledge is automatically synthesized from raw data.

---

<sup>4</sup><http://oss.models-db.com/>

- Architecture knowledge dissemination - the stage at which architecture knowledge is represented to relevant stakeholders in various ways.
- Feedback - where inputs from user and experts are used to improve the learning model.

CAKI utilizes ontology-based models as a basis for personalization mechanisms and exploratory search. However, in the paper, the ontology models are not presented. This work involved industrial collaboration with Siemens.

Lin et al. propose a system (called *IntelliDE*) in which software big data could be aggregated, mined and analysed to provide meaningful assistance for developers across the software development processes [161]. Similar to CAKI, IntelliDE follows a 3-stage knowledge discovery approach, including Data Aggregation, Knowledge Acquisition and Intelligent Assistant. For each stage, a number of key research issues and challenges are listed. Unlike CAKI, IntelliDE uses a so called Software Knowledge Graph as for “storing knowledge in software domains, projects and systems”. In order to construct such graphs from various knowledge sources, a process of parsing and extracting knowledge entities is proposed. However, no extra review or feedback steps are undertaken in order to validate the knowledge to be added to the graph.

#### 6.2.4 Scientific Workflow Systems

In various scientific disciplines, software tools have emerged for automating sequences of (typically data-intensive) steps of a scientific analysis or experimental procedures. Such system are called ‘scientific workflow systems’ and these build on the approaches of general workflow management tools. Automation of workflows enables the reuse, replication and incremental improvement by making changes to the details of the study. In a survey of existing scientific workflow technology, Barker et al. listed 14 frameworks from the business and scientific domain [162]. Among the listed frameworks, some specialised in particular fields of science, while others aim to be more generic. In the paper, the authors suggest 6 key factors to consider when developing scientific workflow systems, including: i) collaboration is key (to avoid overlapping requirements and reinvention of any wheels); ii) use a conventional scripting language; iii) reuse existing workflow language; iv) research your domain (before building the systems); v) stick to standards; and vi) have a portal-based access.

In the Software Engineering domain, we find eSEE - a novel framework to support large-scale experimentation and scientific knowledge management in Software Engineering proposed by Travassos et al. [163] - the most relevant. Four main requirements of eSEE are: i) having integrated experimentation support tools; ii) being a Web System; iii) using of e-services based paradigm; and iv) providing knowledge management mechanisms. The architecture of eSEE consists of three three distributed macro-components, i.e. Meta-Configurator (MC), Instantiation Environment (IE) and Execution Environment (EE). The framework, however, does not support automated execution and data analysis from the experiment specification for technology-oriented experiments.

## 6.3 Experiences in Creating & Sharing a Collection of UML Software Design Models

In this section, we summarize our experiences in building, curating and sharing the Lindholmen Dataset which is an extensive dataset of more than 93,000 UML models from more than 24,000 GitHub projects [137]. This dataset also contains meta-data of the projects (such as commits, commit messages, committers, etc.) which enables researchers to study UML models in their usage context.

### 6.3.1 Models Extraction from GitHub

The data extraction process is described in detail in [164]. In this section, we provide a brief summary of the extraction steps and highlight the challenges when extracting the dataset. In general, the data extraction comprises four steps, including: (i) Retrieving the file list of all GitHub repositories, (ii) identifying potential UML files, (iii) examining (and manually evaluating) the existence of UML notation in the obtained files, and finally (iv) collecting the meta-data of the repositories where a UML file has been identified.

In step (i) and (iv), we used the GitHub API<sup>5</sup> to retrieve the list of files from GitHub projects as well as to query meta-data from the projects that contain UML models, respectively. At these stages, we faced two big data retrieval challenges. Firstly, the GitHub API limitation (of 5000 requests/hour) could hugely affect the crawling time. In particular, with up to three GitHub calls for each repository, given the limit of 5000 requests/hour, it would take around 14 months to perform the retrieval of data in step (i). We worked around this by downloading the JSON files in parallel with over 20 active GitHub accounts, which were donated from fellow colleagues and students during this process. This reduced the time span to approximately one month. Secondly, while GitHub is a dynamic environment where projects change over time, we could only work on a static snapshot (and thus, somehow outdated version) of it (captured by GHTorrent). In particular, many repositories might have been removed or made private in the time that goes from GHTorrent obtaining its data (which is before February 1<sup>st</sup> 2016) and our request to the GitHub API (during Summer of 2017). This resulted in a huge number of repositories (around 3 million) where we obtained an empty JSON file or an error message from the GitHub API.

In step (ii) and (iii), the main aim was to identify those files that actually contain UML models. There were two main challenges. Firstly, browsing through the enormous amount of files stored on GitHub is a challenge itself. Secondly, the file formats in which UML models are stored are diverse, making it difficult to develop a systematic searching approach. For example, UML models are often stored as images to which simple textual searching approach does not work - it required images processing technology. In fact, we split the files into textual-format and image-format, and developed different technologies to treat them separately. Details about the technologies that were used can be found in [137].

Last but not least, since some steps are manually done and therefore are

---

<sup>5</sup><https://developer.github.com/v3/git/trees/#get-a-tree>

expensive, we could not provide updates on the dataset in a frequent and automatic manner. This results in a number of threats to the availability of models in the dataset, such as models become no longer available/accessible and missing mid-flight projects in which UML models were introduced later than the time of analysis.

### 6.3.2 Data Curation

Having the dataset collected, we moved on to more in-depth studies about use of UML models in the context of open source projects. As these studies require a set of projects and models with specific characteristics, the dataset had to be curated. For example, when studying the practices and perception of UML use, we were interested in the projects where we could observe long-term use of UML models and collaboration between contributors [165]. To obtain these projects, we applied some filters on the number of contributors, number of commits and active time-span. With that, we are willing to accept false rejections (e.g. ‘serious’ projects that use UML might be rejected) in favour of no (very low) false positives.

Successful curation can also be achieved by adding extra knowledge to the existing dataset. In particular, we performed a number of classifications: a) Classifying types of UML models was done manually and b) Automatically classifying reverse engineering diagrams and forward design diagrams [166]. The classification results were then added/annotated in the dataset.

### 6.3.3 Sharing the Lindholmen dataset

The availability of the Lindholmen dataset has attracted researchers in the field to use and study the data set. For example, El Ahmar et al. used more than 3500 diagrams from the dataset to study the use of visual variables (such as size, brightness, texture/grain, etc.) in UML models in open source projects [61]. Schulze et al. used 50 sequence diagrams from the Lindholmen dataset for evaluating their automatic layout and label management [64]. Unfortunately, the results of these research have never been integrated/annotated into the Lindholmen data set because of two reasons. Firstly, these investigations have been conducted with small subsets of the dataset, making it hard to generalize the research result to the whole dataset. Secondly, there has been no systematic and convenient way for researchers to integrate their findings to the data set.

## 6.4 Challenges for Big-data Driven Empirical Studies in Software Architecture

In this section, we discuss the challenges (C) for conducting big-data driven empirical studies on software architectures. The discussion reflects our observations on research in the field as well as our experience in building the Lindholmen dataset.

### **C1: Finding a common representation for software architectures.**

Source code is always represented as some type of text-file that conforms to some formal grammar. For example, object-oriented source code consists of

classes, methods and interfaces. Indeed the aim to be ‘compilable’ enforces that the source code conforms to a formal grammar. Notwithstanding the existence of standards for software architecture and UML, there is a very high diversity in the representation of software architecture across different projects. Software architecture documents may be represented in formats as diverse as Word (doc(x)), PDF, HTML, PowerPoint (ppt), among others. The content of software architecture documents is a mix of natural language, images, and sometimes tables and diagrams. Indeed the content is a mix of descriptions of the system architecture, sometimes including design principles, design rationale, and even source code examples. This complicates the definition of a common representation (data-model) of which information to represent for each architecture.

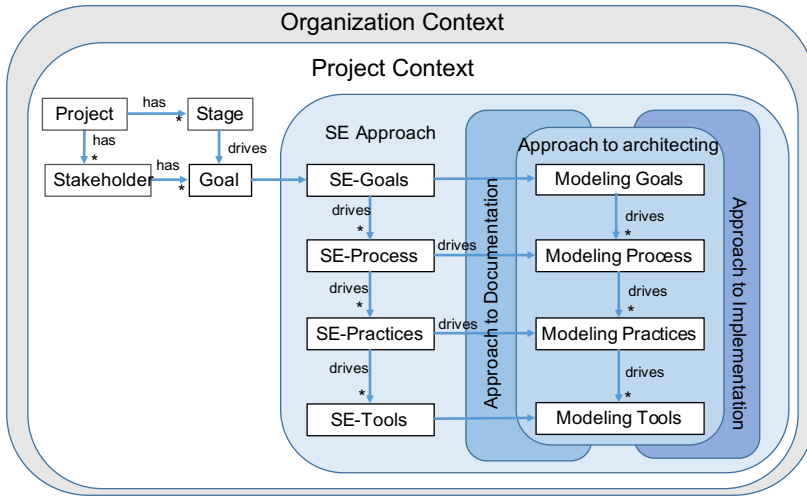
**C2: Capturing relevant context information.** Source code has as main purpose to represent the implementation in a manner that is compilable and executable by a computer. Software architecture on the other hand, serves different purposes to different consumers over time: in early stages of projects, architecture documentation is typically used to create a shared understanding among architects. Later on such documenting happens after (or in concert with) making the implementation, and serves as to align architecture and implementation. Moreover, the documentation serves as a reference for developers to record which parts of the system have been implemented and stabilized. Also, testers of the software draw on information from the software architecture, e.g. to understand quality objectives as well as scoping decisions. In open source repositories, we can observe the production of architecture, but not its use/purpose/aim(s). The way an architecture is used is key to analysing the benefits that can be harvested from it. This includes processes and practices of the project (such as quality assurance, processes for monitoring conformance of implementation, or the way in which architecture is used in producing implementation).

Indeed, the representation, completeness and level of abstraction of the description of an architecture depends on the stage of the project it is used in: at the start of a project, architectures may not be crystallized very much, hence little of the system is represented by an explicit architecture representation. For mature projects, architecture documentation usually focuses on high level views of the system (so as to be able to provide one overview of the system), especially in large software projects. As a consequence, the representation will need to leave out many details. In summary, when we want to understand the role of architecture in a project, we need to consider as well various contextual factors, such as the stage of development, project size and geographical distribution of the development team. Fig 6.1 generalizes the complex nested contexts that influence the goals of architecture and thereby the various processes, practices and tools used (generalized from [84]). This Figure illustrates the empirical finding that there is a hierarchy of contexts that influence how software practices are used. There are organizational and project factors that include the goals of the stakeholders. For example, these may prioritize delivery date over quality of the software. Such priorities in turn



affect the ways in which architecting is done. In particular they will affect the goals of doing architecting and via this also the processes, practices and tools used for architecting. Indeed, for a true understanding of the value of achitecture practices, all these context factors would need to be understood. However, this contextual data is typically not obtainable via 'artefact mining' approaches.

Figure 6.1: Impacts of contexts to software modeling approach



**C3: High effort for crawling big-data.** Empirical research into software architecture requires a non-trivial amount of software architecture (empirical) data in order to draw representative findings and conclusions. However, collecting/building such dataset is challenging for the following two reasons.

Firstly, due to the vast variety in representation and use of software architecture, identification of such SADs is a huge challenge per se. This becomes even more challenging when searching for SADs in big data such as GitHub, SourceForge. For example, when building the Lindholmen dataset, it was impossible to manually scan through the whole GitHub data to look for UML models. We had to apply some heuristic searches and develop automated methods to identify UML models in different file formats, including images. Building up such technology was a challenging and time consuming task in itself [117].

In addition to the unavailability of automatic identification methods, it is worth noting that the limited (human- and machine-) resources could hugely affect the amount and the quality of software architecture data to be collected. In particular, studies that involve the identification of SADs often target a small amount of SADs because of limited human resource within the research team (for identifying, verifying, maintaining the data), thus running the risk of data not being representative. Moreover, to many studies that use the GitHub API (such as [167]), the limitation of a maximum of 5000 request per hour is a technical challenge that limits

the speed and scope of SADs search.

**C4: High effort for curation.** Collecting software architecture artefacts from open source requires a lot of curation. Firstly, public repositories are frequently very ‘noisy’: they do not only contain software development project, but also, e.g., student projects and course material [127]. Secondly, as argued in the previous section, SADs exist in a very wide variety. Studies that aim to employ ‘big data’/machine learning techniques must realize that there are “many different animals in the SAD-zoo” that share very little commonality. One way to understand the zoo of SADs is to enable community/crowd-sourcing curation, e.g., through annotation and classification. We elaborate on the need for curation in the next section. Another recommendation is to set up mechanisms as early as possible to monitor and improve the quality of the dataset. Given that typically large volumes of data are involved, this must be automated as much as possible. This is complicated by the fact that each ‘entry’/datapoint for one software architecture is very rich in many different types of attributes and context factors.

**C5: Collaborating in empirical software architecture research.** Collaboration has become a common practice in doing big-data driven empirical research. This is due to the fact that such type of research often requires a huge amount of efforts to which a single researcher might not be able to cover all parts by his own. For example, in order to build the Lindholmen dataset, collaboration between researchers who are specialised in specific fields was necessary - some researchers were more specialised in mining big-data from GitHub, some others were responsible for developing techniques for detecting UML content in arbitrary files. Prior to forming the working team, it was important to establish the research intent and look up for potential collaborators via researcher’s own network. When analysing data, the researchers needed to communicate with each other on the steps and progress of data analysis as well as the preliminary results. Team effort was also needed in developing a community around the research. This included creating website, communicating with relevant research groups at various conferences/workshops and responding to (extra-feature) requests from the research community. However, the level of (tool-)support for the collaborative empirical research activities was far from sufficient. The authors of the Lindholmen dataset was not aware of any tool that supports team-working for all the above-mentioned activities.

## 6.5 Directions for a Community Infrastructure for Big-data Driven Empirical Research in Software Architecture

Given the challenges for big-data driven empirical research in software architecture, a solution could be to build up a community-based infrastructure that enables researchers to share and reuse software architecture artefacts as well as

to collaborate in their empirical studies. In this section, we discuss the main requirements (**R**) for building such an infrastructure (called “*the infrastructure*” hereafter).

**R1: Be able to host big- & heterogeneous data of SADs.** As mentioned in the previous section, to many empirical studies identifying and collecting software architecture artefacts is a big challenge. Therefore, the first and foremost requirement is that the infrastructure should be built upon and be able to host enormous corpus(es) of software architecture artefacts. Since the representation of architecture artefacts is highly heterogeneous, the hosting solution might need to be flexible enough to integrate and handle both structured and unstructured data (such as design documentations, requirements).

**R2: Share not only data but also software architecture knowledge and analysis.** Having access to a huge corpus of software architecture artefacts is a big advantage, but it is not enough. Often, researchers and practitioners collect data for their studies with a set of criteria in mind. In many cases, the criteria concern not only the data itself, but also additional knowledge about and existing analyses on the data. For example, El Ahmar et al. browsed the Lindholmen data set manually to be able to collect 3500 UML diagrams with different visual variables for their study [61]. In the study, the authors found many UML diagrams where *color variable* was misused and thus, could have had a negative impact on the communication where the diagrams were used as an intermediate. The additional knowledge and analysis (hypotheses, method & results) could potentially serve as an input for other researchers to make a more knowledgeable selection of data or even to make a follow-up study.

**R3: Enable links to contextual data.** Mining studies have given a huge boost to empirical studies on software development, and source code-related studies in particular. Indeed, various important open questions related to software architecture require studying the relation between the architecture and the source code. For example: how does software architecture affect the quality of the source code or how does architecture affect the evolution/maintainability of a system.

**R4: Support evolving artefacts and architectural knowledge.** Software development is an evolving process in which software artefacts, including architecture artefacts, are constantly updated during project’s life time. Accordingly, software architecture and corresponding architectural knowledge are subject to change as the project progresses. In order to facilitate research about evolution of software architecture, the infrastructure should provide means for managing versions of not only software architecture artefacts, but also corresponding architecture knowledge. While there have been numerous solutions for managing versions of software artefacts, collecting, organizing and administrating versions of architectural knowledge remain challenging. This is mainly due to the fact that the process of discovering architecture knowledge is not always fully-automated, making it hard to collect multiple versions in a frequent and systematic manner. Indeed, the infrastructure should help

to mitigate this issue by allowing users to define their scientific workflow and enabling reuse of analysis and computational services.

- R5: Keep annotations separately.** Ongoing research efforts require the enriching of empirical data by annotations. For example, annotations delineating the location of features in source code (such as in [168]), or annotations indicating whether a piece of source code (or architecture) is related to security. Performing annotations invasively by adding annotations into the artefact itself is not scalable. In fact, this will become very messy when multiple types of annotations need to be combined. Indeed it should be a requirement that one artefact from a software project can be annotated by multiple parties. Hence, we should implement exogenous ways of annotating the artefacts found in software projects, i.e., the annotations should exist outside the actual artefact. This triggers the question on how to refer to particular parts of an artefact (source code or document) in order to link an annotation to a fragment of an artefact.
- R6: Crowdsourcing annotation.** Annotating large systems, possibly from multiple perspectives is a colossal task. Possibly in some cases, such annotations can be done automatically, either using external sources or through machine learning algorithms. For those cases where automation is impossible, this task is probably best addressed as a community-effort. For supporting such efforts, the tooling-infrastructure should support some way of crowdsourcing. When opening a system up for annotation and crowdsourcing, one would need to also introduce mechanisms for i) authentication, ii) quality assurance, and iii) traceability of annotations.
- R7: Enable comparison & aggregation of research findings.** Empirical studies in software architecture sometimes contradict in their findings. One of the main reasons is that different studies base their analyses and findings on different samples of the population, thus, observe different phenomena. This infrastructure provides a way for different research to study on the same empirical process and possibly on the same data sources, therefore it should also enable comparison and aggregation between research findings from different studies.
- R8: Encourage discussion and peer-review.** A benefit of joining a community is to get early feedback and support. This benefit should therefore be considered as a core value when building the infrastructure. As sharing interests in studying software architecture, researchers in the community should indeed be able to report their experience and give comments/questions/feedback on other studies. The infrastructure should learn from scholarly social network such as ResearchGate <sup>6</sup> to encourage members to interact and exchange knowledge such as bonus points and badges for reviewers/commenters.
- R9: Promote collaborative research and enrich a collaboration network.** Many initiatives in creating a corpus for software architecture artefacts have become inactive or left outdated. Only 5 out of 12 reported

---

<sup>6</sup><https://www.researchgate.net/>

corpora in the SEMI index [24] are active at the time of writing this chapter - none of them have been updated since 2014. It seems these corpora have not been successful in growing a research community around them. A reason could be that inadequate effort has been spent on promoting the corpora as well as maintaining the research networks around the corpora. This should be taken as a lesson learned when building the infrastructure. For example, the following methods are successfully used in ResearchGate: i) suggest researchers with related studies, analyses and questions/answers; ii) allow researchers to invite other researchers (that might be interested) to review/visit their studies; iii) support announcing research plans and calls for joint efforts (and then team composition).

While the above list of requirements and desiderata is by no means complete, it should be a starting point to capture many of the core characteristics. As the infrastructure aims at facilitating collaboration within a scientific domain, requirements for building a scientific workflow management system are relevant. In particular, together with the above-mentioned nine requirements, we would recommend to consider an addition of nine requirements mentioned by Ludäscher et al. [169] when building the infrastructure (Table 6.1 summarises these requirements).

## 6.6 Overview of CoSARI

In this section, we present our proposal for the novel framework CoSARI (**C**ollaborative **S**oftware **A**rchitecture **R**esearch **I**nfrastructure) and our ongoing efforts to construct it.

### 6.6.1 Overview of the CoSARI Framework

Fig. 6.2 illustrates the architecture of CoSARI, which is composed of three layers: Data Storage layer, Business SaaS layer and Presentation layer.

#### 6.6.1.1 Data Access Layer

This layer provides access to the data sources of the system, which consists of the following:

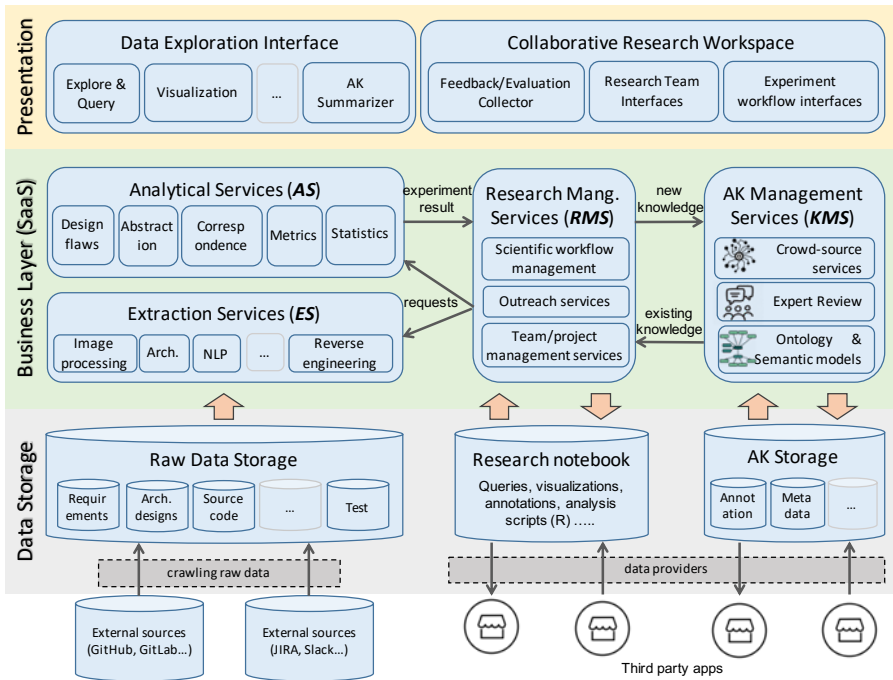
**Raw Data Storage** stores SADs as well as contextual data related to the project that the SAD belongs to (**R1**, **R3**). The contextual data includes amongst others: source code, requirements, testing documents. Data in this storage is crawled from common code sharing systems (such as GitHub, GitLab), issue tracking systems (such as JIRA) and developer communication channels (such as Slack).

**Research Notebook** is the component that stores research profiles of all empirical studies conducted on the system, thus enables sharing of scientific analysis on software architecture (**R2**, **R4**). The research profile might consist of hypotheses, analyses, queries, progress, result, etc. of a specific empirical study. This component also keeps track of the human records of empirical research such as research individuals and teams which are essential for promoting collaborative research network (**R8**, **R9**)

Table 6.1: Nine requirements for building a scientific workflow management (by Ludäscher et al. [169])

<b>Requirement</b>	<b>Content</b>
<i>Seamless access to resources and services</i>	Using web services for remote service execution and remote database access.
<i>Service composition &amp; reuse and workflow design</i>	Web services should be constructed in a way that can be combined to do complex tasks. This is similar to the idea of using using micro-services (in SOA architecture).
<i>Scalability</i>	Should support data-intensive and compute-intensive workflows.
<i>Detached execution</i>	Long running workflows require an execution mode that allows the workflow control engine to run in the background on a remote server, without necessarily staying connected to a user’s client application that has started and is controlling workflow execution.
<i>Reliability and fault-tolerance</i>	To make a workflow more resilient in an inherently unreliable environment, contingency actions must be specifiable, e.g., fail-over strategies with alternate web services.
<i>User-interaction</i>	Allows users to inspect intermediate results and select and re-rank them before feeding them to subsequent steps. Allows user to reconnect to the running instance and make a decision before the paused (sub-)workflow can resume.
<i>“Smart” re-runs</i>	A “smart” rerun would not execute the workflow from scratch, but only those parts that are affected by the parameter change.
<i>“Smart” (semantic) links</i>	Assists users in workflow design and data binding phases by suggesting which actor components might possibly fit together, or by indicating which data sets might be fed to which actors or workflows.
<i>Data provenance</i>	The results of a conventional experiment should be reproducible, computational experiments and runs of scientific workflows should be reproducible and indicate which specific data products and tools have been used to create a derived data product.

Figure 6.2: Architecture for Architecture Research Framework



**Architecture Knowledge Storage (AK Storage)** is dedicated to storing all architecture knowledge generated on top of the raw data. Data in this database can be annotations, meta-data, quantitative or qualitative assessment of the software architecture artefacts. *Research Notebook* and *Architecture Knowledge Storage* are kept separately from the Raw Data Storage for the reasons mentioned in **R5**.

Data from these databases can be provided via a API for third parties to use.

### 6.6.1.2 Business Layer (SaaS)

This layer provides services for the following main tasks: i) extracting and analysing data from software artefacts in order to generate new knowledge on top of the raw data ; ii) managing the process of reviewing and retrieving architecture knowledge, and iii) managing the collaborative scientific-workflow and maintaining research networks. This layer should be built on a Software-as-a-Service (SaaS) architecture as this would allow researchers/practitioners to create and run their own analyses on the system. This is expected to increase the flexibility of CoSARI towards hosting and analysing various types of input data (as mentioned in **R1**). The four main components of this layer are the following:

**Extraction Services (ES)** are responsible for accessing and extracting data from the Raw Data Storage. For example, this could employ services for parsing and extracting various information from source code. This could also

contain numerous natural language processing (NLP), image processing (IM) and data-mining services to extract useful information from the unstructured data. Another example could be services for reverse-engineering the software design from source code. This could serve as a basis for various analyses about conformance of implementation to requirements and original design, etc.

**Analytical Services (AS)** employ methods for analysing data collected from the Extraction Services to provide qualitative or quantitative results for answering research questions about the software architecture. The following is an example of such a service: A service that computes the correspondence between software design and implementation (source code). For this, the AS service needs ES to provide a list of class names from source code and a list of class names from the design documentation of the project. Then, the AS service is responsible for checking the similarity between the names and calculating the correspondence rate (as the naming convention used when designing and coding might be very different, this is a challenging task itself). Other services that might fall in this layer could be calculating design flaws (from source code and architecture documents), analysing the role of software components in design of a system, establishing mappings and traces between software artefacts, building statistical analysis, etc.

**Collaborative Research Management Services (RMS)** provides core services for managing the collaborative work-flow for software architecture research (**R9**). This should look similar to existing scientific work-flow management systems such as Kepler [169] with additional support for team-work and outreach of the research. In particular, using the services, research teams should be able to discuss, modify and experiment with various experimental settings. *RMS* also allows research teams/individuals to search and compare research profile and findings from other empirical studies (**R7**). Outreach services aim at promoting the research within and outside of CoSARI by various activities (depending on the stage of the projects), for example: i) supports creating a project and looking for potential co-researcher in the network; ii) supports announcing research result/event/milestone within and outside of CoSARI; iii) allows other researchers to follow/subscribe specific studies; iv) manages rewarding system to encourage researchers to contribute more to projects; etc.

*RMS* interacts with the *Research Notebook* in order to store and update research profile. Besides, *RMS* delegates management of research findings (and software architecture knowledge) to the Architecture Knowledge Management Services (KMS).

**Architecture Knowledge Management Services (KMS)** are services that manage architecture knowledge within CoSARI. In particular, the *KMS* have two main tasks: i) Collecting and storing new architecture knowledge; and ii) Generating meaningful answers to questions regarding software architecture of a specific system.

Regarding the first task, new knowledge is derived from two main sources, including human-intelligence (e.g., experts and crowd-sourcing assessments) and results from empirical studies. The request to store/update architecture knowledge is directed from *RMS* or the crowd-source services to *KMS* (**R6**). The new knowledge might need to be reviewed before being stored/updated at AK Storage.

Regarding the second task, the *KMS* makes use of an ontology which de-



scribes generic concepts used in software architecture and relationship between the concepts. This allows *KMS* to be able to generate answers to both pre-defined questions and user-articulated questions (**R2**). The answers will then be returned to the *RMS* or to the *Presentation* layer. Feedback from experts and users to the answers can be used to improve the vocabulary of the ontology.

### 6.6.1.3 Presentation Layer

This layer provides interfaces for end-users to explore and start a new empirical research project on CoSARI. The interfaces can be split into the two following categories:

**Data Exploration Interface** which supports users to search, identify and understand the architecture artefacts to be used in their future research. Example applications in this layer include textual summarization or graphical visualization of software architecture, dashboards that show architecture quality, etc.

**Collaborative Research Workspace** which supports users with everything to create, cooperate on a collaborative research project. This also provides application forms for collecting feedback from users and experts. Section 6.6.2 shows the use of applications in this layer with more details.

## 6.6.2 Main Use-cases of CoSARI

In this section, we provide two main use-cases of CoSARI: i) Identifying a set of SADs to study; and ii) Starting and managing an empirical research project on CoSARI.

### **Use-case 1: *Identifying the set SADs to study.***

A researcher wants to search for an architecture dataset in CoSARI for his study. The researcher can either browse the existing (public) datasets and studies hosted by CoSARI or perform a search to filter relevant data. In particular, the researcher can choose to apply different filters on different properties of the data. For example, he can search across all UML diagrams that contain a specific “search-term” and are large-sized (e.g. have more than 25 classes).

Maybe the researcher wants to filter on a property that is not yet provided. In that case, he can implement and run his own extraction services and analytical services to extract the property from the Data Storage. Maybe the researcher wants to further understand a specific dataset or project; he can then consult visualizations or summaries of the architecture data.

After this step, the researcher should have identified a dataset to start with his experimental study.

### **Use-case 2: *Starting and managing an empirical research project.***

After identifying a dataset, the researcher can start an empirical research project by creating a work space for the project. This includes: i) creating research profile and work plan; ii) defining scientific workflow; and iii) inviting fellow researchers/practitioners to join the projects. CoSARI can suggest researchers that might be interested in joining based on matching of their research profiles to the chosen data-set and research topic; ii) *creating the experimental workflow*, which consists of small steps of data extraction & analysis, data visualization and statistical test, etc.

As the project runs, CoSARI supports team-work by allowing team members to cooperate in creating, configuring and executing analytical services. A research team can also use outreach services provided in CoSARI to call for joint efforts as well as to get feedback about the project's approach. Feedback to a research can be given via comments and discussions. The system records updates on the experimental approach and results into the project notebook (profile). As a result, the history of the research approach can be traced.

### 6.6.3 Ongoing and Future Work

In this section, we present our on-going efforts and future work on building CoSARI.

**Creating raw data storage & querying interface.** The Lindholmen dataset of UML designs is the initial dataset to be part of the core database for CoSARI. We are constantly working on curating this dataset, for example by labeling UML diagrams with their type (Class Diagram, Sequence Diagram, etc.) and by identifying reverse-engineered from forward-design diagrams [166]. We provide access to the Lindholmen dataset via a website and a REST API (these are currently under testing and not yet available for use). In particular, the website would allow users to search for UML diagrams with multiple filters at both project level (such as project name, founder, number of commits/stars/issues, etc.) and model level (such as model name, type of UML model, number of elements inside, etc.). Fig. 6.3a shows the advanced search interface of the website when searching for GitHub projects that: i) have project name containing the word "hotel", and ii) have at least 1 and maximum 10 UML diagrams, and iii) have class diagrams that are small/medium sized, i.e. the diagrams contain from 3 to 15 classes. Fig. 6.3b shows the search result which contains 25 projects that match the advanced search.

**Building an ontology of software architecture knowledge.** As mentioned in a previous section, such an ontology is an essential part for the *KMS* of CoSARI. The design of our ontology was originally inspired by the ontology demonstrated by De Graaf et al. [170]. Some modifications were made for the ontology to be able to capture different perspectives of software architecture, i.e., by following a 4+1 architecture view model. Moreover, the ontology is also able to reason about requirements, rationale and implementation regarding software architecture. For example, the rationale behind a *design option* of software architecture is expressed by *arguments*, *constrains* and *assumptions* and results in specific technologies (e.g. libraries, languages, frameworks, etc.). Fig. 6.4 shows a part of the ontology (a full-size version is available online <sup>7</sup>). Tao et al. demonstrated how to use the ontology to compose answers to various questions that are commonly asked by developers [171].

**Future work.** CoSARI is as of now a reference architecture and there are lots of work to be done in order to implement such architecture. For example, at the *Data Storage Layer*, data collected from different external sources are heterogeneous, thus needed to be standardized/hamonized (e.g. popularity metrics are different between GitHub and Bit Bucket). Thus, CoSARI should be equipped with an ETL (Extract-Transform-Load) tool for hamonising the data. At the *Business Layer*, as of now, the *RMS* has not yet been implemented.

<sup>7</sup><http://bit.ly/software-explanation-composer>

A first step toward implementing it would be to evaluate how much can be reuse from existing scientific workflow management systems [162].

As CoSARI aims at supporting the research community of software architecture, getting early feedback from the community when building it is a must. We plan to build a prototype of CoSARI and conduct a user-study to assess its usability (e.g. by using the System Usability Scale SUS [172]) and improve it accordingly.

## 6.7 Summary and Conclusions

In this chapter we explored the topic of doing big-data driven empirical studies in software architecture, and in particular reflected on what an infrastructure would look like that could support a community to do collaborative research on this topic. We pointed out that we can build on earlier works in the area of scientific workflow systems and software knowledge discovery. Also we discussed several existing and ongoing effort in creating large collections of software architecture representations (either models or complete design documents). We discussed some lessons learned as well as remaining challenges and future directions. Building on these experiences, we proposed a reference architecture *CoSARI* that can serve as a starting point for our community towards building a common infrastructure for performing collaborative empirical research in software architecture.

*Lindholmen Database* QUERY UPLOAD ABOUT LOGIN/REGISTER

**Advanced search :**  
Expand the type of result you want and fill the search fields which match your requests.

**- Project**

Project level criteria:

hotel  Project Name  +

1  10  N# of UML Diagrams  + -

**- Diagram**

Diagram level criteria:

Diagram Name  +

Select one type:  Class  Sequence  Other

Class Diagram level criteria:

3  15  N# of Classes  +

**- Documentation**

Documentation level criteria:

Architecture  Specific Term  +

Select click type.  
Comb from time.

(a) Advanced Search user interface

*Lindholmen Database* QUERY UPLOAD ABOUT LOGIN/REGISTER

Show charts  Sort by:  Increasing **Number of results : 25**

**#1 Tyger-Waterfront-Hotel**

UML models: 3 Overall quality:  More info

Founder: AlexErasmus Project type:

N# of xml representation: 1 N# of commits : 92

N# of docs:

Class diagram:  Sequence diagram:  Other diagram:

User tags:

XML:  UML:  IMG:

**#2 Hotelio**

UML models: 1 Overall quality:  More info

Founder: Ans1353 Project type:

N# of xml representation: 1 N# of commits : 15

N# of docs:

Class diagram:  Sequence diagram:  Other diagram:

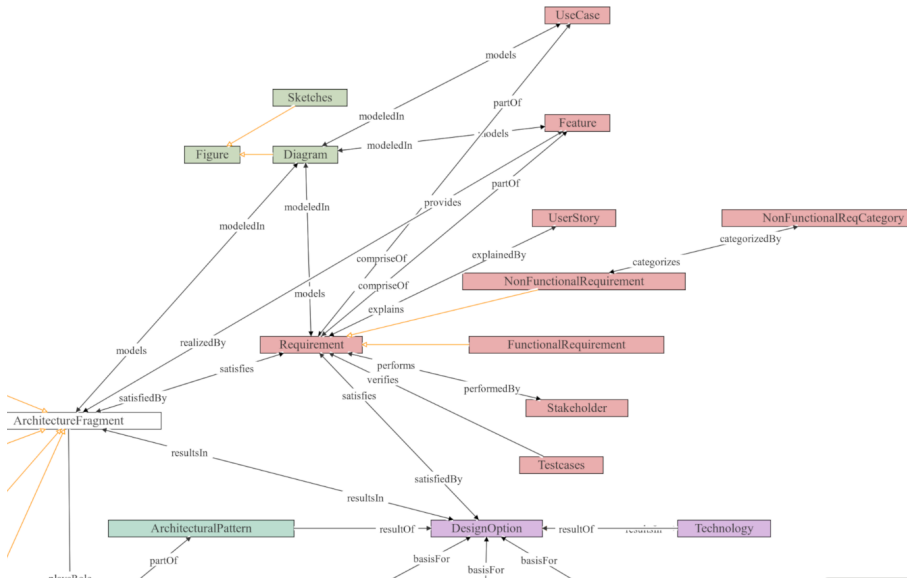
User tags:

XML:  UML:  IMG:

(b) Search Result UI - "card" view

Figure 6.3: GUI for sharing Lindholmen DB

Figure 6.4: A part of the architecture knowledge ontology





# Chapter 7

## Paper F

**Does UML Modeling Associate with Lower Defect Prone-ness?: A Preliminary Empirical Investigation**

A. Raghuraman, T. Ho-Quang, A. Serebrenik, M.R.V. Chaudron,  
B. Vasilescu

*In Proceeding of the 16th International Conference on Mining Software Repositories (MSR 2019), Montréal, Canada, May 26 - May 27, 2019 .*





## Abstract

The benefits of modeling the design to improve the quality and maintainability of software systems have long been advocated and recognized. Yet, the empirical evidence on this remains scarce. In this paper, we fill this gap by reporting on an empirical study of the relationship between UML modeling and software defect proneness in a large sample of open-source GitHub projects. Using statistical modeling, and controlling for confounding variables, we show that projects containing traces of UML models in their repositories experience, on average, a statistically minorly different number of software defects (as mined from their issue trackers) than projects without traces of UML models.

## 7.1 Introduction

Software design is widely accepted as a fundamental step to developing high-quality software [173].

By making designs developers go through a process of reflection, including discussing trade-offs and alternatives, which should result in more thoughtful designs and more maintainable systems [174]. The communication benefits to explicit software design are also well understood: architectural decisions that developers make become well-documented, reducing information loss and potential misinterpretation during system implementation, and facilitating communication among team members and the onboarding of new developers [174]. Both commercial [174] and open-source software developers [165] alike recognize these potential benefits.

Among modeling languages, the Unified Modeling Language (UML) is often viewed as de-facto standard for describing the design of software system using diagrams [165]. In practice, UML is often used in a loose/informal manner (not adhering strictly to the standard [13]). Also UML is used selectively, focusing on important, critical or novel parts.

Still, despite many expected benefits of UML modeling on software development outcomes, the empirical evidence on the matter is scarce. Notable exceptions include a study by Arisholm *et al.* [175], showing through two controlled experiments involving students that, for complex tasks and after a learning curve, the availability of UML models may increase the functional correctness and the design quality of subsequent code changes. There is also work by Fernández-Sáez *et al.* [84] that suggests an overall positive outlook of practitioners towards UML modeling in software maintenance. Finally, we note an empirical study by Nugroho and Chaudron [174] of an industrial Java system, showing that classes for which UML-modeled classes, on average, have a lower defect density than those that were not modeled.

In this paper we study the intuitive and widely held belief that *the use of UML modeling, on average, should correlate with higher software quality*. To this end, we statistically analyse empirical data obtained from 143 open-source GITHUB projects. Many hypotheses about the benefits of UML models on specific software maintenance outcomes have been proposed [5]. However, more generally, one can expect that the mere practice of UML modeling as part of software development indicates a high team- and process maturity and deliberateness that, in turn, should lead to higher-quality code.

In search of evidence [176] to substantiate this belief, we start from a publicly available data set of open-source software projects on GITHUB that use UML models [164], and: 1) assemble a control group of GITHUB projects not known to use UML models; 2) mine data from the GITHUB issue trackers of both sets of projects (using and not using UML models), estimating their defect rates (“bug” issue reports) as a proxy for software quality; and 3) use multivariate statistical modeling to estimate the impact of having UML models on defect proneness, while controlling for confounding factors. Our results reveal a small statistically significant effect of using UML models on defect proneness, *i.e., projects with UML models tend to have fewer defects*.

Table 7.1: GITHUB slugs for the 50 UML projects in our data set

---

abb-iss/SrcML.NET	kite-sdk/kite
aegif/NemakiWare	LibrePCB/LibrePCB
asciidocfx/AsciidocFX	longkerdandy/mithqtt
boost-experimental/di	lvggiano/owner
christophd/citrus	lycis/QtDropbox
claeis/ili2db	mbeddr/mbeddr.core
cliqz-oss/keyvi	MvvmFx/MvvmFx
collate/collate	MyRobotLab/myrobotlab
Comcast/cats	Particular/docs.particular.net
cpvrlab/ImagePlay	pipelka/roboTV
crowdcode-de/KissMDA	plt-tud/r43ples
dandelion/dandelion-datatables	Protocoder/Protocoder
djeedjay/DebugViewPP	rbei-etas/busmaster
droidstealth/droid-stealth	robotology/codyco-modules
eProsima/Fast-RTPS	SINTEF-9012/ThingML
Freeyourgadget/Gadgetbridge	smartdevicelink/sdl_core
GeertBellekens/Enterprise-Architect-Toolpack	SpineEventEngine/core-java
GluuFederation/oxAuth	SpoonLabs/astor
godc/godc	telefonicaid/fiware-cygnus
HPI-Information-Systems/Metanome	timolson/cointrader
imixs/imixs-workflow	UESTC-ACM/CDOJ
infinidb/infinidb	uwescience/myria
IQSS/dataverse	vicrucann/dura-europos-insitu
kamilfb/mqtt-spy	xamarin/monodroid-samples
kermitt2/grobid	xen2/SharpLang

---

## 7.2 Methodology

We designed a quasi-experiment to compare the defect proneness between two groups of open-source GITHUB projects: a *treatment* group of projects using UML models, part of a public data set [164]; and a *control* group of projects sampled randomly using GHTORRENT [45]. We describe our data collection and analysis process next.

### 7.2.1 Data

As part of a previous study [?], Robles *et al.* [164] released a data set of 4,650 non-trivial GITHUB projects,<sup>1</sup> defined as having at least six months of activity between their first and most recent commits and at least two contributors, that use UML models, as identified by a manually-augmented automated repository mining technique. As our operationalization of defect proneness involves mining the projects' GITHUB issue trackers (details below), we only include in our *treatment* (UML) group those projects that had at least 30 issues on GITHUB as of March 2018; we determined the threshold empirically after manual exploration of the data, to filter out largely inactive projects. In

<sup>1</sup>Available online at <http://oss.models-db.com>

Table 7.2: Breakdown of our data set by language

	Java	C#	C++
UML projects	31	6	13
Control group projects	63	10	20

addition, we identified using Google’s `langdetect` library<sup>2</sup> those projects not using English as their natural language, as our operationalization of defect proneness makes expects issue discussions in English. We further filtered out projects that are not primarily written in either C++, C#, or Java (as labeled by GITHUB), the languages traditionally associated with UML. Next, we filtered our projects with fewer than 10 stars on GITHUB and projects in which the gap between the first commit and the first GITHUB issue is more than a year, in an effort to further exclude student homework assignments and largely trivial or inactive projects [127]. Finally, we excluded projects started before 2009 – shortly after GITHUB itself started – such that all remaining projects could have plausibly used the GITHUB issue tracker from the beginning. After all these filters, the treatment group (Table 7.1) consists of 50 UML projects.

Note the relatively small size of the treatment group after applying the different activity-based filters when compared to the size of the original data set by Robles *et al.* [164], and especially when compared to the size of GITHUB. Still, to our knowledge, this is the largest data set on which our research question has ever been studied empirically.

To assemble a *control* group of projects not known to use UML models, we randomly sampled, using the March 2018 version of GHTORRENT [45], projects that satisfied the same criteria (see above), for a total of 93 non-UML projects after filtering; the version we queried is newer than the one used by Robles *et al.* [164], hence we did not consider projects that did not already exist in the older version. Moreover, we further ensured that the randomly selected control projects were not already present in the treatment group.

Table 7.2 presents a breakdown of our two groups of projects by programming language.

## 7.2.2 Operationalization

*Dependent variable.* As a measure of a project’s defect-proneness and a proxy for its software quality, we estimate the *number of bug-related issues* reported in its GITHUB issue tracker. To identify *bug-related* issues, as opposed to feature requests, tasks, or other types of issues commonly found in open-source issue trackers [177], we developed a *Naive Bayes* classifier [178]; a similar approach was considered by Zhou *et al.* [179]. Our classifier takes as input the title and the body of an issue, and produces one of two labels, *bug* or *not bug*.<sup>3</sup>

To this end, we started with one author coding randomly sampled issues as *bug* or *not bug*, until labelling 100 of each; unclear cases were discussed between two authors, and subsequently resolved. After manual data labelling,

<sup>2</sup><https://pypi.org/project/langdetect/>

<sup>3</sup>Classifier code and data analysis script available online at <https://github.com/adi1234567890/UML-defect-proneness>

we created 20 random splits of our labelled data containing two equally sized train and test sets, trained the classifier on the train set, and computed the accuracy on the test set; each split preserved the balanced nature of the data, *i.e.*, the train and test sets contained 50 *bug* and 50 *not bug* issues each. The average accuracy of our classifier over the 20 random splits is 89%.

Ultimately, we chose the best performing of the 20 classifier instances and ran it on the unlabelled data, *i.e.*, all the issue reports of all the projects in our UML and control groups. Overall, we labelled 29,983 issues as *bug* and 48,579 issues as *non bug* across the 143 (50 + 93) projects in our data.

*Independent variables.* Our main predictor variable is a dummy *has UML* that distinguishes between the treatment and control groups. In addition, we cloned all the GITHUB repositories locally and computed several variables for co-variates and confounding factors: *project age*, *i.e.*, the number of days between the first and the most recent commit; *primary programming language*, as reported by GITHUB; *number of contributors*, *i.e.*, distinct commit authors in the project's history, after resolving aliases using a script published<sup>4</sup> by Vasilescu *et al.* [180]; *number of commits*, as a proxy for project complexity; *number of stars*, as a proxy for popularity and size of user base; and *programming language*, coded relative to Java as baseline. Moreover, we used Munaiah *et al.*'s [181] *reaper*<sup>5</sup> to compute: *has CI*, a dummy indicating whether or not the project uses continuous integration; *has license*, defined analogously; *test suite ratio*, the fraction of test lines of code to source lines of code; and *code comment ratio*, the fraction of comment lines of code to all lines of code. These variables control for projects' level of maturity of their practices.

### 7.2.3 Analysis

To test our hypothesis, we build a multiple linear regression model, a robust technology which enables us to estimate the effects of having UML models on defect proneness *while holding the other independent variables fixed*. We diagnose the model for multicollinearity, checking that the variance inflation factor (VIF) remains below 3 [182]; no variables violated the threshold. We further check if modeling assumptions hold, and observe no significant deviation from a normal distribution in QQ-plots and randomly distributed residuals across the range. The model fits the data well, explaining 58% of the variance (Adj. R<sup>2</sup>). In addition to the regression coefficients, we also report the amount of variance explained by each term (the *Sum. sq.* column in Table 7.3), as obtained from an ANOVA analysis; the relative fraction of the total variance explained by the model that can be attributed to a particular variable can be considered as a measure of its effect size.

## 7.3 Results and Discussion

Interpreting the regression summary in Table 7.3, we observe that among the control variables, only the *number of commits*, the *number of stars*, and the *has license* dummy have statistically significant effects at 0.05 level or below. All

<sup>4</sup>[https://github.com/bvasiles/ght\\_unmasking\\_aliases](https://github.com/bvasiles/ght_unmasking_aliases)

<sup>5</sup><https://github.com/RepoReapers/reaper>

Table 7.3: Linear regression model summary

	Response: log(num_bug_issues)	
	Coeffs (Errors)	Sum. sq.
(Intercept)	0.40 (0.63)	
log(age + 1)	0.05 (0.09)	19.11***
log(num_commits)	0.53 (0.06)***	58.35***
log(num_contributors)	0.10 (0.09)	9.03***
log(stars + 1)	0.18 (0.05)***	6.62***
test_suite_ratio	-0.24 (0.47)	0.02
comment_ratio	0.26 (0.54)	0.01
has_CI	-0.17 (0.13)	1.38
has_license	-0.61 (0.28)*	2.73*
languageC#	-0.19 (0.20)	2.60
languageC++	-0.33 (0.15)*	
has_UML	-0.30 (0.14)*	2.24*
R <sup>2</sup>	0.61	
Adj. R <sup>2</sup>	0.58	
Num. obs.	143	

\*\*\* $p < 0.001$ , \*\* $p < 0.01$ , \* $p < 0.05$

three behave as expected: larger projects and projects with larger communities (of users and, hence, potential issue reporters) tend to have more bugs reported, other variables held fixed; projects that declare a license tend to have fewer bugs reported.

Regarding programming languages, we note that C# projects are statistically indistinguishable from Java projects, while C++ projects tend to have fewer bugs reported than Java projects, other variables held fixed.

Finally, we note a small (approximately 2% of the variance explained by the model) but statistically significant effect of UML models: other variables held fixed, projects with UML models are expected to have about 35% ( $\exp(-0.3)$ ; note the log-transformed response) fewer bugs reported than projects without UML models.

## 7.4 Threats to validity

We note several threats to validity [47].

*Construct validity.* We measured defect-proneness by computing the number of bug-related issues in the issue tracker. However, issues and bugs may not map one-one [183], *e.g.*, several issues could pertain to the same bug, or one issue may encompass several bugs. While imperfect, this operationalization is common in the literature, *e.g.*, [184]. We also note that projects may use issue trackers outside of GITHUB, which we did not track, potentially biasing our defect estimates. We tried to alleviate this threat by only considering projects which used the GITHUB issue tracker substantially, thus arguably reducing the risk that they also use external issue trackers. Another construct validity threat stems from not accounting for different types of UML modeling, *e.g.*, sequence vs class diagrams, and lumping the different UML modeling techniques into one group. We leave analysing this distinction for future work.

*Internal validity.* Given the multiple linear regression technology we used, with controls for known confounding factors, our results should be relatively

robust. Still, we note a threat to internal validity from using a Naive Bayes classifier to label issues. In particular, the algorithm works on the Naive Bayes assumption that given the target label, each of the covariates is independent [185], which may not always hold. Finally we also note that we did not run the original UML mining technique on the control group projects to further confirm absence of UML models in their repositories; since the original UML data set was created by comprehensively mining every project from GHTORRENT, we assume that the risk of mislabeling non-UML control group projects is low. However, it must also be noted that there is the possibility of projects having information about design/architecture in *.pdf*, *.ppt* and other such files which therefore could've been wrongly classified as projects not using UML modeling by [164].

*External validity.* We note, again, the relatively small size of our data set, which can largely be explained by the small number of open source projects that meet all selection criteria. Moreover, our sample is not representative (by construction) of open-source or GITHUB as a whole. It is unclear without ample future work and replications, beyond the scope of this paper, whether our results can generalize. Another threat to the external validity comes from the dataset used to train the Naive Bayes classifier. Mature projects are known to write thorough issue reports while less mature projects tend to completely ignore or very sparingly make use of the issue tracker [186]. As a result, the classifier is inherently rigged towards learning features from more mature projects.

## 7.5 Conclusions

Prior work in this area focused on understanding the impact that UML design had on software projects in a qualitative manner. Through this paper, we try to provide a quantitative analysis of the way in which UML modeling of design relates to the defect proneness of the projects. Our work shows that after controlling for confounding factors, projects that use UML experience, on average, a statistically minorly different number of software defects

**Future Work.** One of the immediate next steps for this work, as indicated by the threats to validity section, is to distinguish between the type of UML modeling in our treatment group projects. In particular, making the distinction between sequence and class diagrams and studying their corresponding correlation with defect proneness, we believe, will further reduce the dearth of research being done in this area. Another future work involves making a distinction between forward designed and reverse engineered projects [187]. Work done by Fernández-Sáez *et al.* [188] shows a positive outlook of practitioners towards the use of forward design but empirical research regarding the correlation between forward/reverse design and defect proneness, once again, is scarce. A third future work involves reproducing the investigation in a new sample of projects but performing a more precise verification of certain variables. The project members may perhaps be also inquired to check whether UML models were used or not and the way in which they were being used. Lastly, we believe that UML modeling directly influences structural aspect of the software architecture/design to some extent. For instance, we would expect the use of UML structure-diagrams to have an influence on metrics such as

cohesion, coupling, etc. of the software projects. Our current work opens the door to studying these questions in the near future.



# Chapter 8

## Paper G

**Using Machine Learning for Automated Classification of Class Responsibility Stereotypes in Software Design**

**T. Ho-Quang, A. Nurwidyantoro, M.R.V. Chaudron**

*Under submission.*

This paper is an extended version of the paper "Automated Classification of Class Role-Stereotypes via Machine Learning", accepted at EASE2019.



## Abstract

Role stereotypes indicate generic roles that classes play in the design of software systems. Knowledge about the role-stereotypes can help in various tasks in software development and maintenance, such as program understanding, program summarization and quality assurance. This paper presents an automated machine learning-based approach for classifying the role-stereotype of classes in Java projects. We analyse the performance of this approach against a manually labelled ground truth for three open source projects which contain altogether 1,500+ Java classes. The contributions of this paper include: i) a machine learning (ML) approach to address the problem of automatically inferring role-stereotypes of classes (for Object Oriented Programming Languages), ii) the manually labelled ground truth, iii) an evaluation of the performance of the classifier, iv) an evaluation of the generalizability of the approach, and v) an illustration of 3 new uses of role-stereotypes. The evaluation shows that the Random Forest algorithm yields the best classification performance. We find however, that the performance of the ML-classifier varies a lot for different role-stereotypes. In particular its performance degrades for rare role-types. Among the 23 features that we study, features that relate to collaboration-characteristics of classes and complexity of classes stand out as the best discriminants of role-stereotypes.

## 8.1 Introduction

The concept of "role stereotype" was introduced by Wirfs-Brock as a concept to denote ideal types of well-scoped responsibilities of classes [189]. Such role stereotypes indicate generic responsibilities that classes play in the design of a system, such as controller, information holder, or interfacier. Knowledge about the role-stereotypes can help in various tasks in software development and maintenance, such as program understanding, program summarization and quality assurance.

Dragan et al. have proposed methods for automatically inferring the role-stereotype of classes in C++ [190]. Moreno et al. [191] migrated this approach to Java. Both approaches are based on a collection of expert-designed decision rules that are applied to the syntactical characteristics of the source code of classes. This inference of role-stereotypes can be seen as an enrichment of reverse engineering, in particular in the area of uncovering design: Role-stereotypes indicate generic types of responsibility which characterize both the type of functionality that a class should perform as well as the type of interactions a class can have with other classes. Hence role-stereotypes are important clues for the design-intention of classes.

Several studies [55–59] have demonstrated the benefits of using stereotypes in various software development and maintenance activities. These benefits include: program design, program comprehension, quality assurance, and program summarization. We mention the following as concrete examples of the usefulness of role-stereotypes as demonstrated in earlier work: Using role-stereotypes in creating lay-outs of UML class diagrams improves the comprehensibility of the diagrams [57–59]. Alhindawi et al. [192] show that enhancing the source code with stereotype information helps improve feature location in source code.

This paper makes the following contributions:

- [a] We present a machine learning (ML) approach to address the problem of automatically inferring the stereotype of Java classes. Dragan [190] and Moreno [191] observe that the robustness of their rule-based approach leaves room for improvement: the rules of their approach are not 'complete': they do not classify a fairly large portion of classes of a system. In this paper we present an ML-based classifier which classifies all classes and thus is very robust.
- [b] We publish the first sizeable validated dataset of 1,547 Java classes and their role-stereotypes. This dataset can serve as (groundtruth) resource for other researchers.
- [c] We evaluate the performance of our approach. From this, we infer which features are most important for classifying stereotypes, and which machine learning algorithm works best.
- [d] We evaluate the generalizability of our approach.
- [e] We illustrate three new uses of role-stereotypes: i) Use of stereotype specific rules for quality assurance; ii) Discovery of stereotype uses for profiling software system's design style and intention; iii) Discovery of patterns of collaboration between role-stereotypes: We assert that this knowledge contributes fundamental insights into the anatomy of software design.

This paper has the following structure: We first explain the key concept of role stereotype (Section 8.2) and discuss related work (Section 8.3). Next, we explain our research methodology in Section 8.4. As part of this we explain the taxonomy of role-stereotypes that we use in our study and how this relates to other taxonomies. Then, we analyze the performance of various machine learning algorithms for this classification task (Section 8.5), and identify the most influential features (Section 8.6). Next, we investigate the generalizability of the classifier using combination of projects in our dataset in Section 8.7. Then we illustrate two new uses of role-stereotypes in Section 8.8. We end it with discussion, conclusions and future work.

## 8.2 Class Role Stereotypes

Wirfs-Brock [193] proposed an object-oriented design-approach based on the notion that each software object should have a well-defined responsibility in order to play one of a few generic roles in a system's design. Wirfs-Brock classified the roles of software objects into six stereotypes:

- (CT) Controller: objects designed to make decisions and control complex tasks,
- (CO) Coordinator: objects that do not make many decisions, but in a rote or mechanical way, delegate work to other objects
- (IH) Information holder: object designed to know certain information and provide that information to others.
- (IT) Interfacer: objects that transform information and requests between distinct parts of a system. It can be a user interfacer object that interacts with users. An interfacer can communicate with external systems or between internal subsystems.
- (SP) Service provider: objects that perform work and offer services to others on demand.
- (ST) Structurer: objects that maintain relationships between objects and information about those relationships. Structurers might pool, collect, and maintain groups of objects.

This taxonomy aims for orthogonal non-overlapping categories. However, there may be situations where a class can play different roles towards different collaborators.

For our study it is important to realize that Wirfs-Brock suggests to use role stereotypes while *designing* a system. In our study, we aim to establish the role-stereotypes based on the implementation of a system. In general, the classes that end up in an implementation are not ideal. In fact they may mix two (or more) responsibilities.

## 8.3 Related Work

Dragan et al. [194] proposed an automated tool to detect stereotypes of *methods* in the C++ programming language. They define a taxonomy of methods such

as structural (accessor and mutator), collaborational, and creational. They propose several rules to determine method stereotypes based on the type of the method, the return type, and the way in which the method modifies the state of the class. On top of their method for classifying method stereotypes, Dragan et al. create rules to determine class stereotypes [190]. The proposed class stereotypes are: entity, minimal entity, data provider, commander, boundary, factory, controller, pure controller, large class, lazy class, degenerate, data class, and small class. While several of these seem close to Wirf-Brocks's, the Dragan classification is presented as being derived empirically from studying 21 open source projects.

The rules in Dragan's approach consist of a collection of conditions on the quantities of *method stereotypes*; e.g.  $\#mutators > 2 * \#accessors$ . These conditions include thresholds that are based on a mix of theoretical arguments and statistical techniques (based on [195]). In [190], Dragan states that "The rules for stereotype identification are subjective and thresholds might vary depending on differences in subject's interpretations.". Moreover, these rules leave a large number of classes in software systems unclassified. Apparently, these rules do not cover the spectrum of possible combinations of method-stereotypes that occurs in practice.

Based on the work of Dragan, Moreno and Marcus propose a method for the identification of class stereotypes in the Java programming language [191]. For this, they adapted the procedures from [190] and provided additional method- and class stereotypes. In the classifier approach by Dragan and Moreno's rules are not disjoint: different rules each may assign different stereotypes to a single class. Moreno's subsequent research has mostly used this classifier for automatically generating summaries of classes in natural language [196], and seems not to have continued in improving its performance.

Budi et al. [197] built an automated tool to detect design flaws based on design stereotypes. They use a taxonomy of 4 role-stereotypes (different from the ones by Wirf Brock): boundary, control, regular entity and data manager. For these design stereotypes, there exist rules that describe how classes of these stereotypes should be allocated to typical layers of 3-tiered software architectures and how they are supposed to collaborate. The authors used SVM to automatically labels classes into categories. Then, they used rules about the relationship between the stereotypes to detects potential design flaws in the system.

The Gang-of-Four (GoF) design patterns [198] also represent idealized patterns of software design. However, there are important differences between the GoF-patterns and the design stereotypes that we consider: Firstly, in general only a small portion of the classes of a system are part of a GoF-design patterns. Whereas in the design philosophy of Wirfs-Brock, each and every individual class in a system should play at least one of her proposed stereotype-roles. Secondly, GoF-patterns are defined as the specific ways in which individual types of classes collaborate. In contrast, WB-stereotypes are the property of individual classes. For completeness, we mention one approach by Fontana et.al. [199] that uses machine learning to identify design patterns in source code.

## 8.4 Methodology

Figure 8.1 shows an overview of our research methodology. First, we select three case studies and collect their source code (Step 1). Second, we define and establish a ground truth (Step 2 & 3). Then, we extract features from the source code that are to be used by the machine learning algorithms (Step 4). After that, we experiment with various machine learning algorithms (Step 5). Finally, we evaluate the performance of the machine learning algorithms and the discuss three new uses of the role-stereotypes (Step 6). Next, we elaborate these steps in more detail.

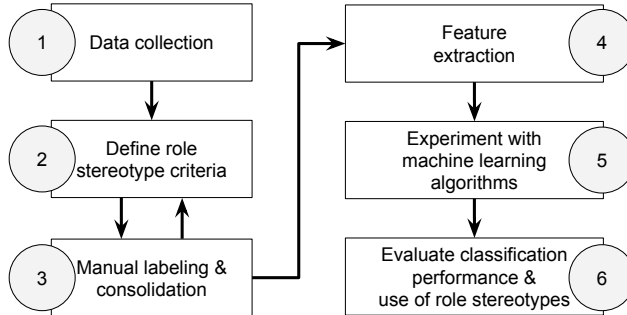


Figure 8.1: Proposed methodology

### 8.4.1 Data Collection

In this research, we use three open source software systems as our case studies: K-9 Mail <sup>1</sup>, SweetHome3D <sup>2</sup> and BitcoinWallet <sup>3</sup>. Table 8.1 shows descriptive information about the projects. K-9 Mail and BitcoinWallet are Android applications and are hosted in GitHub, while SweetHome3D is a pure-Java application and is hosted in SourceForge.

Table 8.1: Description of OSS projects used in this study  
 #Class is calculated at the studied version. #Release, #Contributor and #Star are retrieved at the time of writing this paper

#	OSS project	Studied Version	Released Date	#Class	#Release	#Contributors	#Stars
1	K-9 Mail	v5.304	Nov. 22, 2017	779	367	202	4276
2	SweetHome3D	v5.6	Oct. 25, 2017	546	46	n.a	4.7/5.0
3	BitcoinWallet	v6.31	Oct. 1, 2018	222	274	26	1705

These projects are chosen for this study because of the following reasons:

- they are active open source projects,

<sup>1</sup><https://github.com/k9mail/k-9/tree/1a12b18f0c4a452b74941340179735f0383bd1fb>

<sup>2</sup><https://sourceforge.net/projects/sweethome3d/files/SweetHome3D-source/SweetHome3D-5.6-src>

<sup>3</sup><https://github.com/bitcoin-wallet/bitcoin-wallet/releases/tag/v6.31>

- they use Java as the main programming language,
- the projects vary in size (#Class), domain and technology (Android & pure-Java)
- they are from domains that can be understood by non-experts

We downloaded the source code of these projects from the corresponding GitHub and SourceForge links. We found a small number of “nested classes” that might interfere with the feature-extraction of their outer classes. Therefore, the next step was to extract these nested classes into independent classes. The extraction process was performed as follows. Firstly, source code was parsed using srcML [200]. For a given source code, srcML creates a list of classes, including nested classes, and their details in a standardized XML representation. Then, we used XPath query to generate all classes from the saved XML file. As a result, every nested class was extracted into a separate Java file. We used *checkstyle*<sup>4</sup> to remove unused import statements in every class to obtain the actual number of import statements used and the number of lines in the class. The scripts for automating this extraction process are included in the replication package of this paper [201]. After these steps, we obtained in total 1,547 Java classes over three cases.

## 8.4.2 Ground Truth step 1: Criteria for Role Stereotypes

In order to produce a ground truth for machine learning, we first establish criteria to be used by human experts for manually classifying classes into role stereotypes. The initial criteria were obtained by the authors by studying the descriptions by Wirfs-Brock [193]. Then the criteria were refined and calibrated in follow up meetings where the authors had assessed additional sets of classes (details in section 8.4.3). The criteria can be divided into 3 categories: i) Criteria regarding characteristics of class; ii) Criteria regarding relationship between role stereotypes; and iii) Other criteria. We discuss each of these next.

### 8.4.2.1 Criteria regarding characteristics of classes

These criteria focus on intrinsic (static) properties of classes. We take the *Structurer* stereotype as first example: Table 8.2 shows the criteria used to characterize Structurer classes. In this particular case we look into data types of attributes, library use and content of methods inside a class in order to get an impression whether the class is capable of organizing/manipulating a collection of objects. As a second example, the *Information Holder* may include persistence mechanisms (files or databases). Other class properties such as class name, getter/setter methods, etc. are used for other role stereotypes. A complete list of the criteria for all stereotypes can be found in the replication package of this study [201].

Some of the criteria are rather similar to the rules developed by Dragan and Moreno. For example, we both consider the presence of getter and setter methods as an indication for the Information Holder-stereotype (in our classification) and Entity/Data Provider-type (in Dragan’s classification). However, different

<sup>4</sup><https://github.com/checkstyle/checkstyle>



Table 8.2: Properties of a Structurer class

<p><b><i>What makes a class a Structurer?</i></b></p> <ul style="list-style-type: none"> <li>- May contain “user defined object” type as attributes</li> <li>- May extend Java’s Collection framework or equivalent</li> <li>- Has method(s) to maintain relationships between objects <ul style="list-style-type: none"> <li>+ methods that manipulate the collection such as sort(), compare(), validate(), remove(), updates(), add(), etc.</li> <li>+ methods that give access to a collection of objects such as get(index), next(), hasNext(), etc.</li> </ul> </li> </ul>
--

from Dragan and Moreno, our criteria also consider the presence of other features, such as persistence functionality.

#### 8.4.2.2 Criteria regarding relationship between roles

In [193] Wirfs-Brock mentions that “the roles an object plays imply certain kinds of collaborations”. For this, we form a set of criteria that look at the collaborations stereotypes have with other classes. From Wirfs-Brock’s theory of role stereotypes and collaborations, we come up with the graph in Figure 8.2. This demonstrates common relationship between different role stereotypes. This graph, for example, shows that Information Holders are used by Controllers, Service Providers or Structurers, but not commonly by Coordinators or Interfacers. Hence, the presence or absence of relations can be used as a criterion in the decision-making about the possible roles of a class.

The following are additional examples of criteria on the relationships between roles: Structurer may link to several Information Holders. A Service Provider can store information by collaborating with Information Holder and Structurer classes. An Interfacers class, as an intermediate between different layers of a system, might collaborate with Coordinators and Service Providers in each layer to conduct a cross-layer task. An Interfacers class is often controlled by Controller classes.

#### 8.4.2.3 Other Considerations

We defined additional criteria aimed at essential difference in behaviour between different role stereotypes. For example, both Service Provider and Controller class may include some control-flow logic. The design intention of these classes suggest that decisions made by a Controller should affect a broader control flow of the system, while decisions made by a Service Provider should mostly have effect on the flow within the class itself.

Secondly, we draw a special attention to Android case projects. In particular, Android applications are built upon Android frameworks which encapsulate low-level functionalities of the Android OS. Thus, a number of Controller, Structurer and Interfacers classes at UI and activity management level and collaboration between them might be hidden away. Roles/responsibilities of those classes that extend/implement these base functionalities might possibly be overlooked. The experts pay extra attention by reviewing roles and collaborators of its

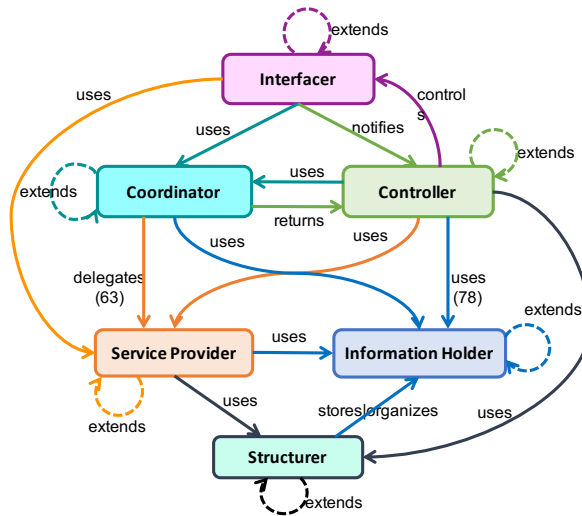


Figure 8.2: Relationship between role stereotypes

ancestor (Android) classes (mainly from Android’s API reference <sup>5</sup>).

Lastly, sometimes a class may carry more than one role. This possibility of multiple roles is also discussed by both Wirfs-Brock [193] (p.4) and Dragan [190]. In this case, experts discuss and choose one most prominent role for this class, and if any secondary role is identified, then this is also recorded.

### 8.4.3 Ground Truth step 2: Manual Labeling and Consolidation

We used an iterative process to establish agreement on criteria and how to apply them. Initially, we randomly chose 20 classes for each project. Two of the authors and one additional Ph.D. student manually labelled these classes. They discussed any differences in classification, and refined the criteria based on this discussion. These steps were repeated two more times on K-9 Mail (as this is the biggest case) until the criteria seemed sufficient/saturated. Next, two of these persons labeled all the remaining classes. A final discussion round took place between the two graders in order to resolve disagreement and hard-to-stereotype cases. The whole manual labeling process took about 100 hours in total (approx. 45 hours per each main grader and 10 hours by the extra grader) spread out over 3 months. Ultimately, we established a ground truth of 1,547 labeled classes, which is all of the three cases [201]. Table 8.3 summarizes the distribution of all classes in the ground truth by projects and by role-prototype.

### 8.4.4 Feature Extraction

Our classification is based on static analysis of the Java source code of a system. We extract the features using the srcML tool [200]. For a given source code, srcML creates a list of classes and their details in a standardized XML

<sup>5</sup><https://developer.android.com/reference/>

Table 8.3: The distribution of role-stereotypes in each project under study

Project	CO	CT	IH	IT	SP	ST	Total
K-9 Mail	79	20	231	77	323	49	779
SweetHome3D	21	38	227	63	159	38	546
BitcoinWallet	2	5	83	62	57	13	222
<b>Total</b>	102	63	541	202	539	100	1547

representation. The values of features are calculated by XPath queries. We chose to use 23 source code features because they correspond to the criteria used in the manual classification (see Section 8.4.2). In the following, we list the features (**F**) together with a short subjective opinion on their relevance to classifying different role-stereotypes. The features are categorized into five categories (**C**), i.e. *Accessibility*, *Complexity*, *Functionality*, *Naming Convention*, and *Collaboration*.

**C1: Accessibility features** represent how accessible a class and its content is.

*F1: classPublicity*: the access modifier of the class. We assume that Service Provider and Information Holder classes might offer public access so that other classes can collaborate with them.

*F2: numPublicMethods*: the number of public methods inside the class. We assume that an Information Holder, an Interfacer, a Service Provider or an Interfacer class might have many public methods.

*F3: numPrivateMethods*: the number of private methods inside a class. We assume that a Controller, Coordinator and Service Provider might distribute the job on separate methods inside their class.

*F4: numProtectedMethods*: the number of protected methods inside a class. We assume that a Controller, Coordinator and Service Provider might distribute the job to separate methods inside their class. These methods might still be used or overridden by its sub-classes.

**C2: Complexity features** represent the complexity of a class.

*F5: loc*: the number of lines in the class' source code. We assume that the Controller and Service Provider stereotype will have more lines of code than the other.

*F6: numIfs*: the number of conditional statements in the class body, i.e., if...else...switch statement. Controller classes might use lots of conditional statements in order to make decisions and to control work flows.

*F7: numParameters*: the total number of parameters in all methods in the class. We assume that methods in a Service Provider or a Coordinator class might have many parameters.

*F8: numAttr*: the number of attributes declared in the class. We assume that an Information Holder class might have many attributes.

*F9: numMethod*: the number of methods declared in the class (constructors are excluded). We assume that Service Provider and Coordinator classes have many methods.

*F10: setters*: the number of methods started with 'set' phrase. We assume that this method is a setter method, i.e., the method that modifies variable in Information Holder.

*F11: getters*: the number of methods started with 'get' phrase. We assume that this method is a getter method, i.e., the method that accesses variable values in the Information Holder class.

**C3: *Functionality features*** aim at detecting specific functions that a class may have.

*F12: isPersist*: a boolean value that indicates whether a class has persistence features, i.e., implements a Serializable interface or importing database connectivity libraries. We assume that Information Holder classes are more likely to employ persistence features.

*F13: isCollection*: a boolean value that indicates whether a class is a subclass of Java's collection library. We assume that a Structurer might possibly need it to maintain relations between objects.

*F14: isClass*: a boolean value that indicates whether the source code file is a class. We assume that a class can represent all of the role stereotypes.

*F15: isEnum*: a boolean value that indicates whether the source code is a Java's enum. We assume that the enum type can represent an Information Holder.

*F16: isAbstractClass*: a boolean value that indicates whether a class is an abstract class.

*F17: isStaticClass*: a boolean value that indicates whether a class is a static class. We assume that a static class can represent a Service Provider or an Information Holder.

*F18: isInterface*: a boolean value that indicates whether the source code file is a Java's interface. We assume that an Interface can provide methods that must be implemented by a Service Provider or a Structurer.

**C4: *Naming Convention features*** detect specific naming conventions in the class name.

*F19: numWordName*: the number of words in the class name. We assume that Information Holder and Structurer role stereotypes have a simple short name, while the others might have a longer name.

*F20: isOrEr*: a boolean value that indicates whether the class name is ended with 'or' or 'er'. We believe a Controller or a Service Provider class is more likely to have a name that ends with that 'or' or 'er'.

*F21: isController*: a boolean value that indicates whether the class name ends with 'controller'. We believe that those classes with this characteristic are more likely Controller classes.

**C5: Collaboration features** indicate level of collaboration of a class with other classes.

*F22: numImports*: the number of import statements in the class. Classes carrying the roles like Controller, Coordinator, Interfacier and Service Provider might need to collaborate with many other classes. Thus, we assume that they might have many import statements.

*F23: numOutboundInv*: the number of invocation to outside of the class methods. We assume that the Coordinator, Controller and Interfacier have many invocations outside of its class.

In comparison with the approach proposed by Dragan et al. [190] and Budi et al. [197]: some of their features are similar to our features, i.e. numMethod, setters, getters, and numOutboundInv. However, they only considered smaller number of features and the majority of our features were not considered in their work.

### 8.4.5 Machine Learning Classification Experiments

We experiment with 3 machine learning algorithms, i.e. Random Forest (RF), Multinomial Naive Bayes (MNB) and Support Vector Machine (SVM). These algorithms are widely used in machine learning research and provide good performance on various applications. We use stratified 10-fold cross-validation to evaluate the performance of each algorithms. The classification performance is measured using Recall, Precision, F1 Score and Matthews Correlation Coefficient (MCC).

We perform two experiments for machine learning algorithms. In the first experiment, we analyse which algorithm provides the best performance in classifying all role-stereotypes. For this we use each role-stereotype as a separate classification category. Hence this constitutes a multi-class classification. We explore the use of the SMOTE [202] resampling technique to handle the imbalanced distribution of role-stereotypes. We compare the performance between using the regular and the SMOTE resampling technique.

In the second experiment, we analyse which features are important for classifying each individual role-stereotype. For this we use only one machine learning algorithm: the one that came out best in the first experiment. In this second experiment, we perform binary classifications for each stereotype; i.e. we use two categories: i) that specific stereotype, and ii) all other stereotypes together. We then evaluate the importance of the features in this classification. For this, we use the Scikit-toolkit for machine learning and its built-in method to compute feature-importance based on Gini scores [203]. We can get this score by calculating the importance of the node for each feature split divided by the importance of all nodes in the tree, then normalize it by the sum of all feature importance values.

### 8.4.6 Generalizability of the Trained ML Classifier

We study the generalizability of the trained machine learning classifier by evaluating the use of different software projects as training data to classify the others remaining software projects. We use the best performed machine

learning algorithm in the previous experiment and use different combination of software projects as the training data and measure their performance.

## 8.5 Experiment Results

In this section, we present experiment results on our classifier Class Role Identifier (CRI).

### 8.5.1 Multi-role Classification of all Stereotypes

In the first experiment, we run the evaluation on the dataset of aggregated data (of 1,547 classes) from our three cases, namely K-9 Mail, SweetHome3D, and BitcoinWallet. In particular, we calculate Precision, Recall, F1 Score and MCC using a 10-fold cross validation. Then we compare this result with the performance reported in our previous work that used only K-9 Mail dataset [204]. We use 1000 trees in the Random Forest classifier to handle the large number of features that we used. Table 8.4 demonstrates the performance result in these two datasets.

Table 8.4: Performance comparison of the additional dataset

Dataset	Classifier Method	Precision	Recall	F1 Score	MCC
All	<b>RF</b>	$0.74 \pm 0.04$	$0.73 \pm 0.03$	$0.71 \pm 0.04$	$0.63 \pm 0.05$
	<b>MNB</b>	$0.58 \pm 0.05$	$0.59 \pm 0.04$	$0.58 \pm 0.04$	$0.43 \pm 0.05$
	<b>SVML</b>	$0.62 \pm 0.04$	$0.66 \pm 0.03$	$0.62 \pm 0.03$	$0.51 \pm 0.05$
	<b>RF (SMOTE)</b>	$0.88 \pm 0.02$	$0.88 \pm 0.01$	$0.88 \pm 0.01$	$0.86 \pm 0.02$
	<b>MNB (SMOTE)</b>	$0.49 \pm 0.03$	$0.46 \pm 0.03$	$0.45 \pm 0.03$	$0.36 \pm 0.03$
	<b>SVML (SMOTE)</b>	$0.43 \pm 0.18$	$0.40 \pm 0.17$	$0.40 \pm 0.17$	$0.32 \pm 0.13$
K-9	<b>RF</b>	$0.69 \pm 0.06$	$0.70 \pm 0.04$	$0.67 \pm 0.04$	$0.57 \pm 0.07$
	<b>MNB</b>	$0.52 \pm 0.11$	$0.50 \pm 0.09$	$0.49 \pm 0.11$	$0.33 \pm 0.13$
	<b>SVML</b>	$0.54 \pm 0.05$	$0.59 \pm 0.06$	$0.55 \pm 0.05$	$0.40 \pm 0.08$
	<b>RF (SMOTE)</b>	$0.89 \pm 0.06$	$0.89 \pm 0.06$	$0.89 \pm 0.06$	$0.87 \pm 0.07$
	<b>MNB (SMOTE)</b>	$0.51 \pm 0.06$	$0.49 \pm 0.05$	$0.48 \pm 0.06$	$0.39 \pm 0.06$
	<b>SVML (SMOTE)</b>	$0.69 \pm 0.03$	$0.69 \pm 0.03$	$0.68 \pm 0.03$	$0.63 \pm 0.03$

From Table 8.4, we observed that Random Forest outperforms MNB and SVML and stays as the best classification algorithm. There is an increase of the performance of the Random Forest using all three projects compared to the performance using only K-9 Mail. This is possibly due to the addition of training data for less frequently role-stereotypes, such as Controller and Structurer. On the other hand, we also observed that there are no significant differences in applying SMOTE resampling technique in both cases. We argue that, by using the SMOTE resampling technique, the number of dataset of each role in both cases will still be equal. More over, the increase of the size of the dataset from using only K-9 Mail to using all the three projects may not large enough to increase the performance of the classifier.

### 8.5.2 Single Role (Binary) Classification

In the second experiment, we run a binary classification on each role-stereotype. From our original dataset we create six new datasets (one for each role-stereotype)

that use exactly two-labels: one label for the role-stereotype at hand, and one label 'Other' that represents the combination of all the remaining role-stereotypes. As for the machine learning algorithm, we use Random Forest which gave the best performance in the multi-role classification experiment (Section 8.5.1). Table 8.5 shows the results for 10-fold cross-validated evaluation for each role-stereotype classification using the imbalanced K-9 Mail dataset from our previous work [204] and both imbalanced and SMOTE-resampled of our three cases extended dataset.

Table 8.5: Single role (binary) classification result

	<b>Precision</b>	<b>Recall</b>	<b>F1-Score</b>	<b>MCC</b>
<b>CO</b>	$0.98 \pm 0.08$	$0.29 \pm 0.12$	$0.43 \pm 0.14$	$0.50 \pm 0.10$
<b>CT</b>	$0.0 \pm 0.0$	$0.0 \pm 0.0$	$0.0 \pm 0.0$	$0.0 \pm 0.0$
<b>IH</b>	$0.89 \pm 0.07$	$0.75 \pm 0.10$	$0.81 \pm 0.08$	$0.75 \pm 0.11$
<b>IT</b>	$0.50 \pm 0.43$	$0.14 \pm 0.13$	$0.21 \pm 0.17$	$0.23 \pm 0.20$
<b>SP</b>	$0.72 \pm 0.08$	$0.65 \pm 0.11$	$0.68 \pm 0.08$	$0.47 \pm 0.12$
<b>ST</b>	$0.55 \pm 0.44$	$0.19 \pm 0.15$	$0.27 \pm 0.21$	$0.30 \pm 0.25$

(a) Imbalanced K-9 Mail dataset [204]

	<b>Precision</b>	<b>Recall</b>	<b>F1-Score</b>	<b>MCC</b>
<b>CO</b>	$0.80 \pm 0.42$	$0.20 \pm 0.17$	$0.30 \pm 0.23$	$0.37 \pm 0.24$
<b>CT</b>	$0.91 \pm 0.17$	$0.43 \pm 0.16$	$0.56 \pm 0.14$	$0.60 \pm 0.12$
<b>IH</b>	$0.91 \pm 0.17$	$0.43 \pm 0.16$	$0.56 \pm 0.14$	$0.60 \pm 0.12$
<b>IT</b>	$0.77 \pm 0.14$	$0.43 \pm 0.12$	$0.54 \pm 0.10$	$0.52 \pm 0.10$
<b>SP</b>	$0.77 \pm 0.07$	$0.70 \pm 0.06$	$0.73 \pm 0.03$	$0.60 \pm 0.04$
<b>ST</b>	$0.60 \pm 0.44$	$0.15 \pm 0.13$	$0.23 \pm 0.19$	$0.27 \pm 0.21$

(b) Imbalanced three cases dataset

	<b>Precision</b>	<b>Recall</b>	<b>F1-Score</b>	<b>MCC</b>
<b>CO</b>	$0.99 \pm 0.00$	$0.96 \pm 0.02$	$0.98 \pm 0.01$	$0.96 \pm 0.02$
<b>CT</b>	$0.99 \pm 0.01$	$0.99 \pm 0.01$	$0.99 \pm 0.00$	$0.98 \pm 0.01$
<b>IH</b>	$0.94 \pm 0.04$	$0.9 \pm 0.04$	$0.92 \pm 0.02$	$0.84 \pm 0.05$
<b>IT</b>	$0.96 \pm 0.01$	$0.96 \pm 0.01$	$0.96 \pm 0.01$	$0.92 \pm 0.03$
<b>SP</b>	$0.87 \pm 0.04$	$0.88 \pm 0.02$	$0.87 \pm 0.03$	$0.74 \pm 0.05$
<b>ST</b>	$0.98 \pm 0.02$	$0.97 \pm 0.02$	$0.97 \pm 0.01$	$0.94 \pm 0.02$

(c) SMOTE-resampled three cases dataset

Comparing imbalanced K-9 Mail dataset (Table 8.5a) and imbalanced three cases dataset (Table 8.5b), we can see the increase of performance in the less frequent role-stereotypes, such as Interfacier and Structurer. The addition of two other projects in the dataset increased the number of those less frequent role-stereotypes that led to the performance increase. Meanwhile, we can see that the use of SMOTE resampling technique (Table 8.5c) significantly increase the classification performance compared to the classification performance of the imbalanced dataset (Table 8.5b), especially in rare role-stereotype binary classifiers.

Table 8.5c demonstrates the excellent performance of SMOTE-resampled dataset for binary classification in all role-stereotypes. The MCC score [205], which is known for measuring the performance of imbalanced binary clas-

sification [206], shows a high score in almost all role-stereotypes. We think this happens because SMOTE resampling technique increases the number of rare role-stereotypes (i.e., Coordinator and Controller) which leads to a good performance. However, more frequent role-stereotypes classifiers, such as Information Holder and Service Provider, have a slightly lower MCC score compared to the less frequent role-stereotypes. We think this happens because, in more frequent role-stereotypes, the SMOTE technique generate less “synthetic” role-stereotypes such that it gave a little impact on the classification result.

**Random Forest classifier gives the best performance in classifying role-stereotypes. The addition of two more cases in the dataset increases the performance of the less frequent role-stereotype binary classifiers. The use of the SMOTE resampling technique increases the performance of the binary classifier for rare role-stereotypes but has very little impact on more frequent role-stereotypes.**

## 8.6 Classification Feature Importance

In this section, we discuss the importance of our classification features with regards to their performance in classifying each role-stereotype. Table 8.6 shows the average Gini scores of each feature (represented in a row) on every role-stereotype (represented in columns) obtained from the above-mentioned binary classification experiment. We omit some features having a very low scores (less than 0.001) namely `isStaticClass`, `isClass`, `isAbstract`, and `classPublicity`. In Table 8.6, for each role-stereotype, the top five features (highest score) are marked with “\*”, while the features that we were expected to determine a role-stereotype explained in Section 8.4.4 are highlighted. For each feature, the number of times where it ends up in the Top 5 and the number of times where the it is expected to be determinant are computed and represented in column “#Top.” and “#Exp.”, respectively. The rows are then sorted from highest to lowest #Top. value.

It can be seen from Table 8.6 that most of scores are greater than 0.00 (except the two cells valued 0.000 in column **CT**), meaning that all features in the list have an effect on identifying a role stereotype from others. In terms of prediction power, feature `numImports` (number of import statements) stands out as always being one of the top five most predictive features for every role stereotypes. This implies that the level of collaboration of a class could reveal the role stereotype the class plays in design of a software system. Interestingly, Wirfs-Brock mentioned to the causal relationship between roles and collaboration in her book, that “*the roles an object plays imply certain kinds of collaborations*” (p.159 [193]). With this finding, which is drawn from analysing three realistic software systems, we can confirms the statement in the other direction “*collaborations of a class could characterize its roles*”.

`loc` (number of lines of code), `isOrEr` (class names end with `er` or `or`) and `numParameters` (total number of parameters in method signatures of a class) ranked second to fourth places in the list as being the most predictive features for 4 to 5 out of 6 role stereotypes. This is interesting as we expected neither `loc` nor `isOrEr` to play a major role in identifying Structurer. To our surprise,



Table 8.6: Feature importance for each role-stereotype

Feature	#Top.	#Exp.	CO	CT	IH	IT	SP	ST
numImports	6	4	0.080*	0.073*	0.115*	0.175*	0.071*	0.079*
loc	5	2	0.070*	0.077*	0.068*	0.053	0.093*	0.070*
isOrEr	4	2	0.058	0.166*	0.063	0.092*	0.098*	0.076*
numParameters	4	2	0.088*	0.039	0.106*	0.084*	0.060*	0.045
numAttr	2	1	0.046	0.032	0.116*	0.032	0.140*	0.053
getters	2	1	0.096*	0.024	0.044	0.042	0.042	0.124*
numMethod	1	1	0.059	0.059	0.049	0.044	0.050	0.064*
numWordName	1	2	0.063	0.021	0.037	0.067*	0.048	0.061
isController	1	1	0.004	0.162*	0.002	0.003	0.008	0.006
numIfs	1	1	0.067	0.089*	0.063	0.030	0.058	0.056
numOutboundInvocation	1	3	0.056	0.043	0.054	0.071*	0.048	0.040
isInnerClass	1	2	0.031	0.022	0.063*	0.050	0.024	0.039
numPublicMethods	1	4	0.080*	0.036	0.048	0.043	0.055	0.056
setters	0	1	0.022	0.054	0.022	0.049	0.041	0.020
isPersist	0	1	0.005	0.001	0.004	0.001	0.008	0.001
isCollection	0	1	0.001	0.001	0.001	0.001	0.003	0.047
isInterface	0	2	0.001	0.000	0.021	0.004	0.020	0.012
isEnum	0	1	0.004	0.000	0.034	0.001	0.026	0.002
numPrivateMethods	0	3	0.018	0.040	0.019	0.047	0.021	0.029
numProtectedMethods	0	3	0.034	0.017	0.017	0.033	0.021	0.021

*numParameters* performs well in classifying Interfacers classes. This might be due to the fact that Interfacers play an intermediate role between different layers of the system (e.g. between users and system, between different layers within the system), thus carrying complex methods with notable amount of parameters to harmonize the communication between the layers.

To understand the how values of these features spread over different role-stereotypes, we create boxplots of the values of top 5 predictive features as shown in Fig. 8.3. Feature *isOrEr* is a boolean type, thus is not shown. In each boxplot graph, role-stereotypes are sorted from the lowest to the highest median value of the corresponding feature. It can be seen from Fig. 8.3 that the value of the features ranges differently by role-stereotypes, some are greater than others. For example, Information Holder classes are likely to have smaller *numImports*, *loc* and *numParameters*, while having a generally higher *numAttr* compared to Service Provider, Provider and Structurer classes. This supports our expectation when selecting these features (as mentioned in Section 8.4.4). There is also a case where our expectation goes otherwise: we expected Service Provider classes to have high *numImports*, *loc* and *numParameters*. The graphs, in fact, show that these values of Service Provider classes are generally lower than that of Coordinator, Structurer, Interfacers and Controller. A consistent trend across the boxplots is that Controllers and Interfacers are always placed last in the list as having highest values of *numImports*, *loc*, *numParameters*, *numAttr*.

Moreover, 3 out of 5 most predictive features (i.e. *loc*, *numParameters*, *numAttr*) are Complexity features. This suggests that the complexity of a class might characterize its role-stereotypes. We further discuss the relation between role-stereotypes and system's complexity in Section 8.8.2.

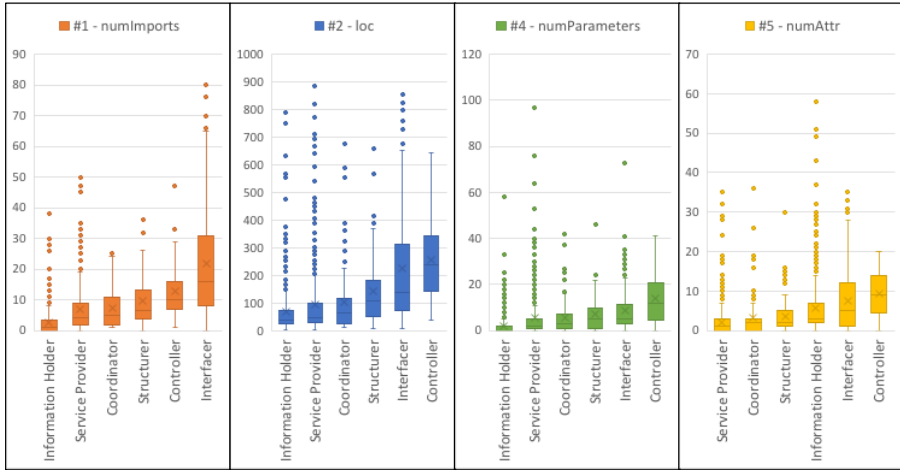


Figure 8.3: Distribution of `loc`, `numImports`, `numParameters`, `numAttr` across role-stereotypes

In each graph: Role-stereotypes are sorted from lowest to highest median value of the corresponding feature

Most of our features contribute to identifying roles. Some features seem to be more predictive than others. Among them, *numImports* and *loc* stand out as the best discriminants. Role stereotypes and collaborations have a bidirectional relationship in which the roles a class plays can be revealed by its collaboration/collaborators and vice versa.

## 8.7 Generalizability of the Classifier

In this section, we study the generalizability of our approach. To this end, we explore different choices of training- and testing-sets. In particular, we study the use of one or two projects for training and then testing the classifier on the remaining project(s).

### 8.7.1 Generalizability Experiment 1: Single Case Training

We start by applying the classifier trained with data from K-9 Mail from our previous work [204] to the other two cases. In this experiment, we use the Random Forest classifier with SMOTE resampling that gave the best performance.

Table 8.7: Performance of the classifier trained on K-9 Mail

Project	Precision	Recall	F1 Score	MCC
BitcoinWallet	0.65	0.52	0.56	0.38
SweetHome3D	0.72	0.73	0.72	0.62

Table 8.7 demonstrates the performance of the classifier trained on K-9 Mail on the other two cases, namely BitcoinWallet and SweetHome3D. From Table 8.7, we can see that the classifier performs average on classifying role-stereotypes of BitcoinWallet and slightly better on SweetHome3D. We investigate this further by the confusion matrix of the classifier for both cases presented in Table 8.8.

Table 8.8: Confusion matrix of the classifier trained on K-9 Mail

		Predicted Label					
		CO	CT	IH	IT	SP	ST
Actual Label	CO	<b>0</b>	0	1	1	0	0
	CT	1	<b>0</b>	0	1	2	1
	IH	5	1	<b>42</b>	7	25	3
	IT	18	0	1	<b>36</b>	7	0
	SP	7	5	4	8	<b>32</b>	1
	ST	0	0	1	3	4	<b>5</b>

(a) BitcoinWallet (222 classes)

		Predicted Label					
		CO	CT	IH	IT	SP	ST
Actual Label	CO	<b>3</b>	4	3	0	10	1
	CT	0	<b>32</b>	0	3	3	0
	IH	1	6	<b>200</b>	4	8	8
	IT	0	2	2	<b>43</b>	13	3
	SP	1	6	15	12	<b>116</b>	9
	ST	1	2	13	7	10	<b>5</b>

(b) SweetHome3D (546 classes)

The confusion matrix for both cases (Table 8.8) shows that the classifier misclassified all Coordinators (2 classes) and Controllers (5 classes) in the BitcoinWallet project (Table 8.8a) but managed to classify some of these two role-stereotypes correctly for SweetHome3D (Table 8.8b). We think the poor classification happens more in BitcoinWallet because the number of Coordinator and Controller in BitcoinWallet (2 CO and 5 CT) is much smaller than their numbers in SweetHome3D (21 CO and 38 CT). We believe that this is the main reason why the performance of the classifier on SweetHome3D is better than the performance on BitcoinWallet. Another concern for the classification of BitcoinWallet was that there was frequent misclassification of Information Holder as Service Provider (25 cases).

## 8.7.2 Generalizability Experiment 2: Double Cases Training

Next, we investigate the generalizability of our approach using the combination of two projects as training data and the remaining case as the testing data. The aim of this experiment is to study the effect of having more training data compared to the previous experiment (i.e., use only K-9 Mail as the training data). Table 8.9 summarizes the performance of the classifier in this experiment.

Table 8.9: Performance of the classification trained on two cases and tested on the remaining case

Training Data	Testing Data	Precision	Recall	F1-Score	MCC
K-9 and SweetHome3D	BitcoinWallet	0.65	0.55	0.58	0.41
K-9 and BitcoinWallet	SweetHome3D	0.68	0.71	0.69	0.59
BitcoinWallet and SweetHome3D	K-9	0.58	0.6	0.57	0.42

When comparing Table 8.9 and Table 8.7, it seems that the performance of the classifier does not differ significantly. The classifier still has the same medium performance in classifying role-stereotypes of BitcoinWallet and SweetHome3D. In other words, the addition of another project to K-9 Mail as the training set did not give significant impact to the classifier. Using two new projects, i.e., BitcoinWallet and SweetHome3D as the training set and tested it on K-9 Mail dataset also gave similar performance.

Table 8.10: Confusion matrix of the classifier trained on two cases and tested on the remaining case

		Predicted Label					
		CO	CT	IH	IT	SP	ST
Actual Label	CO	1	0	1	0	0	0
	CT	1	0	0	0	3	1
	IH	6	0	44	7	24	2
	IT	12	0	1	38	11	0
	SP	8	0	5	7	37	0
	ST	0	1	1	3	6	2

		Predicted Label					
		CO	CT	IH	IT	SP	ST
Actual Label	CO	1	3	6	1	9	1
	CT	1	32	0	2	3	0
	IH	1	8	202	3	9	4
	IT	1	1	5	38	13	5
	SP	1	6	26	7	109	10
	ST	1	2	15	7	9	4

(a) K-9 Mail and SweetHome3D (train) on BitcoinWallet (b) K-9 Mail and BitcoinWallet (train) on SweetHome3D

		Predicted Label					
		CO	CT	IH	IT	SP	ST
Actual Label	CO	1	1	14	16	37	10
	CT	0	4	3	4	8	1
	IH	0	1	170	7	44	9
	IT	0	1	4	40	25	7
	SP	2	4	27	24	242	24
	ST	0	1	7	4	29	8

(c) BitcoinWallet and SweetHome3D (train) on K-9 Mail

We then investigated the confusion matrix of the classification result using two projects training data presented in Table 8.10. In Table 8.10, we can see a lot of misclassification of Coordinators in all three cases. We think this is due to the low number of Coordinators, especially in the classification of K-9 Mail (Table 8.10c) which has the smallest total number of Coordinator from two cases in the training set (23 CO).

In the case of classifying Controller, the classifier failed to classify all Controllers in BitcoinWallet (Table 8.10a), even though the total number of Controllers from the other two cases in the training set was the highest (58 CT). We think this is because the number of Controllers in BitcoinWallet is too small (5 CT) as the result of different coding style in which the developer decided to have only small number of Controllers.

To see how the addition of another project to the training set affects the classification of each role-stereotypes, we also compared the confusion matrix of classifying BitcoinWallet and SweetHome3D using K-9 Mail as training set (Table 8.8a and Table 8.8b) with the confusion matrix of two cases training set (Table 8.10a and Table 8.10b). In classifying BitcoinWallet, the addition of SweetHome3D in the training set increases the correct classification of Coordinator, Information Holder, Interfacer, and Service Provider but reduces the correct classification of Structurer, resulted in a little increase of performance. On the other hand, adding BitcoinWallet in the training set for classifying SweetHome3D reduces the correct classification of all role-stereotypes except Controller resulting in performance decrease of the classifier.

On a different angle, comparing the combination of Android applications (K-9 Mail and BitcoinWallet) and pure Java application (SweetHome3D) as the training and test set led to an interesting finding. The combination of two Android applications in the training set gave better performance in classifying pure Java application (Table 8.10b). Meanwhile, combining an Android application with a pure Java application in the training set to classify the other Android application gave almost equal performance (Table 8.10a and Table 8.10c) but less than the previous combination. However, we cannot make any further conclusion without studying more Android and pure Java application cases.

**The Random Forest classification model trained with data from one or two projects shows a medium performance when classifying the other project(s). Learning from two projects does not lead to a significant increase in classification performance compared to using one.**

## 8.8 New Applications of Role Stereotypes

The automatic classification into role stereotypes opens up new directions for analyzing and understanding software designs. In this section, we describe three such new uses:

- i) the discovery that different role stereotypes have different characteristics in terms of design metrics can be used for novel approaches to checking design quality of software.
- ii) the discovery that role-stereotypes can be used for profiling software system's design style and intention.
- iii) the discovery of patterns in the anatomy of software designs.

Next, we elaborate on these uses.

### 8.8.1 Stereotype-specific Design Metrics

Classes that carry different roles might differ in their design characteristics. More specifically, on theoretical grounds we expect that Information Holders contain little complicated logic, hence generally should have a low WMC (Weighted Method per Class). In contrast Controllers contain complicated

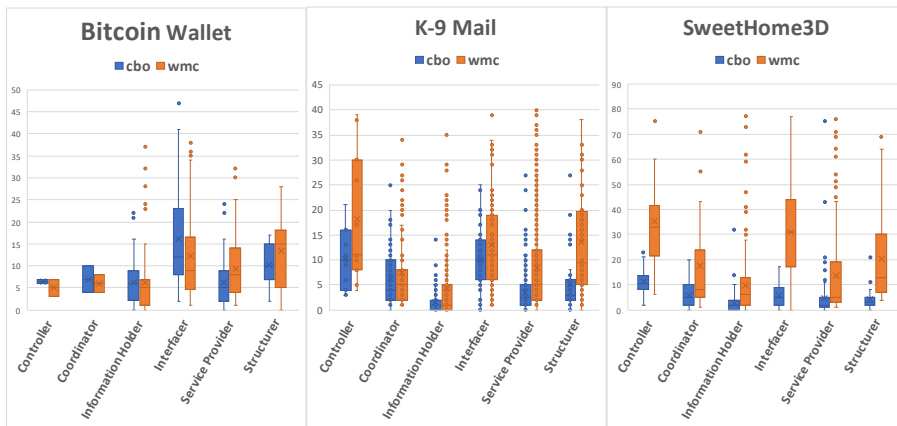


Figure 8.4: Distribution of CBO and WMC across role-sterotypes in three cases

decision logic and hence generally should have higher WMC. We also expect that Controller, Interfacer, and Coordinator to have a higher CBO (Coupling between Object Classes), because these stereotypes communicate with many other classes. We examine these theoretical prediction by checking whether there is any significant difference between WMC and CBO values between classes in different role stereotypes. For that, at first, we used a static-analytical tool named *ck*<sup>6</sup> to calculate WMC and CBO metrics of all classes from our three cases. Then, we display the distribution of the WMC and CBO values on three boxplots corresponding for three cases (Fig. 8.4). Lastly, we ran one-way ANOVA tests with SPSS<sup>7</sup> to check the significance of the difference between the mean values of WMC and CBO across the six role stereotypes. It is observable from Fig. 8.4 that WMC and CBO values range differently between projects. Therefore, a separate test was performed for each project but not on the aggregated data of all projects. Table 8.11 shows results of the test. We found that in K-9 Mail and Sweet Home 3D cases, Information Holder classes have lowest mean value of WMC and CBO, while that values of Controller classes seem to be the highest comparing to other role-sterotypes. In K-9 Mail and Bitcoin-Wallet cases, we also found that Interfacer classes have significantly higher CBO compared to Information Holder, Service Provider and Structurer classes. We could, however, not find any significant difference from Controller and Coordinator to other role-sterotypes in Bitcoin-Wallet case. This is possibly due to the small amount of Controller (5) and Coordinator (2) classes in the system. We discuss about this in the next section (8.8.2). At this point, we could conclude that finding from this analysis is largely aligned with our aforementioned theoretical predictions.

Currently, tools aimed at detecting design smells/anti-patterns typically work by computing design metrics for all classes and then either check these against fixed thresholds or look for outliers (e.g. in [207]). Our analysis shows that design metrics for different role stereotypes have very different ranges: thus

<sup>6</sup><https://github.com/mauricioaniche/ck>

<sup>7</sup><https://www.ibm.com/analytics/spss-statistics-software>

Table 8.11: Results of one-way ANOVA test on the differences in mean values of CBO and WMC between role-stereotypes in 3 cases: K-9 Mail (8.11a&8.11b); Sweet Home 3D (8.11c&8.11d); Bitcoin-Walltet (8.11e&8.11f).

Note: Each cell represents the difference between mean values of CBO or WMC between Source(S) and Target(T) role-stereotypes, i.e. S-T. Cells marked by \* denote statistical significance of more than 95%.

		Source (S)					
		CO	CT	IH	IT	SP	ST
Target (T)	CO	0					
	CT	-3.89*	0				
	IH	4.81*	8.70*	0			
	IT	-3.81*	0.08	-8.62*	0		
	SP	2.75*	6.63*	-2.07*	6.55*	0	
	ST	1.17	5.06*	-3.65*	4.98*	-1.58	0

(a) Differences in average CBO (K-9Mail)

		Source (S)					
		CO	CT	IH	IT	SP	ST
Target (T)	CO	0					
	CT	-10.60*	0				
	IH	3.26*	13.86*	0			
	IT	-5.57*	5.02	-8.84*	0		
	SP	-0.86	9.74*	-4.12*	4.72*	0	
	ST	-6.03*	4.57	-9.30*	-0.46	-5.17*	0

(b) Differences in avg. WMC (K-9Mail)

		Source (S)					
		CO	CT	IH	IT	SP	ST
Target (T)	CO	0					
	CT	-15.58*	0				
	IH	4.13	19.71*	0			
	IT	-4.17	11.41*	-8.30*	0		
	SP	0.60	16.18*	-3.54	4.76	0	
	ST	0.42	16.00*	-3.71	4.59	-0.18	0

(c) Differences in avg. CBO (Home3D)

		Source (S)					
		CO	CT	IH	IT	SP	ST
Target (T)	CO	0					
	CT	-117.19*	0				
	IH	28.41	145.60*	0			
	IT	-24.87	92.32*	-53.28*	0		
	SP	19.96	137.15*	-8.45	44.83*	0	
	ST	-7.22	109.97*	-35.62	17.66	-27.18	0

(d) Differences in average WMC (Home3D)

		Source (S)					
		CO	CT	IH	IT	SP	ST
Target (T)	CO	0					
	CT	-9.00	0				
	IH	0.54	9.54	0			
	IT	-15.88	-6.88	-16.42*	0		
	SP	0.07	9.07	-0.47	15.95*	0	
	ST	-3.92	5.08	-4.46	11.96*	-3.99	0

(e) Differences in average CBO (Bitcoin)

		Source (S)					
		CO	CT	IH	IT	SP	ST
Target (T)	CO	0					
	CT	-33.50	0				
	IH	-1.49	32.01	0			
	IT	-19.12	14.38	-17.63*	0		
	SP	-4.91	28.59	-3.42	14.21*	0	
	ST	-12.50	21.00	-11.01	6.62	-7.59	0

(f) Differences in average WMC (Bitcoin)

some value of a metric can be an outlier for an Information Holder, but would be a normal value for an Interfacer or Controller. This implies that it makes little sense to applying the same threshold on a design metric uniformly for all classes, and that metric thresholds should be tailored to different (stereo)types of classes. This is aligned with the prior work on code metrics that suggests an adaptive approach to classes that have different design characteristics. This is aligned with previous studies that show the distribution of code metrics can vary due to various contextual factors, including design decision [208, 209].

**Classes that carry different roles are likely to differ from each other with respect to their design characteristics. Therefore, design smell detection should be tailored to different role-stereotypes. In particular, design metric thresholds should be tailored to different role-stereotypes.**

## 8.8.2 Using Role Stereotypes for Profiling Software Design Intention/Principles

In this section we explore whether there is any regularity with respect to the occurrence of role-stereotypes across the multiple cases that we consider. Fig. 8.5 contains three diagrams that show the frequency of occurrence of the role stereotypes and the relationships between them for our three cases.

In each diagram, the numbers below the role-stereotype names indicate the absolute number of occurrences and the relative number of occurrence of the role-stereotypes. The labels on the edges indicate the type and frequencies of occurrence of relations between these role stereotypes. Looking at these diagrams side by side, we can see some similarities and differences between the three cases. Next, we will look into these in some more detail.

**The distribution of role stereotypes in systems is imbalanced** In Fig 8.5 we can see that some stereotypes occur often and some are rare. Yet numbers suggest that there may be regularities across software systems. For example, for all systems, the Information Holder-stereotype covers between 30% - 40% of the classes. As another example, the Controller stereotype seems to be rather rare: it represents only between 2.3% - 7.0% in all systems. This finding is aligned by a finding in the work by Dragan: in [190], by applying StereoClass on 5 open source systems, Dragan also finds that the stereotypes that they consider differ in frequency of occurrence, but the relative frequencies of occurrence are somewhat regular across systems.

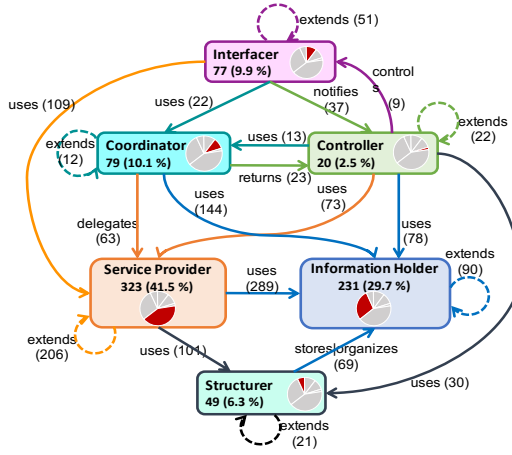
We note that the the categories of stereotypes used by Dragan have similarities to our categories, but also have differences. Both our and their approaches use categories for 'Information Holder' and 'Controller' for which their semantics indeed seem to match. For these categories we find quite similar percentages of occurrence: Information Holder on average 29.6% and Controller on average and 1.9%) comparing to that amounts of Information Holders and Controllers across our three cases (avg. 34.0% and 4.0% respectively). Indeed, The fact that this distribution of occurrence of stereotypes is very unbalanced has a negative effect on the performance of the machine learning algorithms.

### **On the occurrence of stereotypes and the complexity of systems**

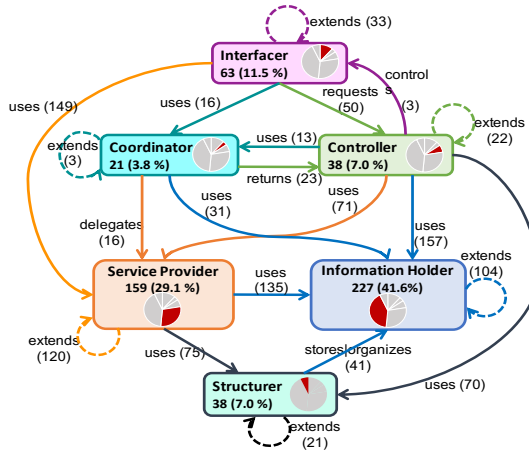
Coordinators are present in the designs of systems when there is a need for coordinating or delegating jobs from one class to another class. This need arises when a class become too big (in size) and complicated. This suggests that Coordinators are seen less often in small systems and more often in large systems. From our 3 cases we observe indeed that the frequency of Coordinators increases with the size of the systems. In particular, in the case of Bitcoin-Wallet, there are only 2 Coordinator and 5 Controller classes - together less than 4 percent of all classes. On a closer look, these classes contain little logic for the controlling and coordinating of workflows from Interfacers- to Service Provider- and Information Holder classes. In fact, in Bitcoin Wallet, most user requests are distributed directly to Information Holders and Service Providers by Interfacers. This results in a high frequency of collaboration between these role stereotypes. This is different from the SweetHome3D and K-9 Mail cases where a large amount of work is coordinated via Coordinators and Controllers.

Indeed, the fairly small amount of Coordinators and Controllers in BitcoinWallet can partly be explained by looking into the design intention of the system. That is, BitcoinWallet has a focus on providing wrapper functions to the *bitcoinj*-library for Android devices. It relies on *bitcoinj* in order to maintain a wallet and send/receive transactions in Bitcoin protocols. As a consequence, the logics and workflows defined in BitcoinWallet are mostly used for monitoring the process and adapting it for Android users to use. Given

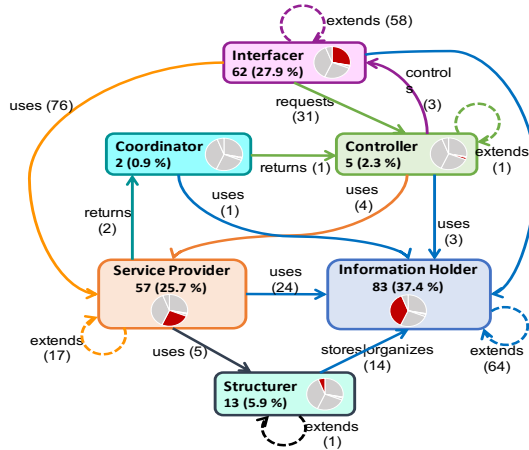




(a) K-9 Mail



(b) SweetHome3D



(c) BitcoinWallet

Figure 8.5: Occurrences of role stereotypes in three cases:  
 (a) K-9 Mail, (b) SweetHome3D; (c) BitcoinWallet

this characteristic of the design, the direct contact from Interfacers (where user requests are made) to execution units (Service Providers) and data storage (Information Holder) can be considered as efficient. Compared to BitcoinWallet, K-9 Mail and SweetHome3D can be considered as being more complex in the sense that the main logic/workflows are defined within the system itself and whereas BitcoinWallet can build on a lot of logic that is contained in an external library. In particular, while K-9 Mail handles the mailing process from the level of mail protocols to end-user management level (such as multiple account management, scheduling services), Sweet Home 3D provides services for designing home plans which might include creating/joining walls, arches, insertion of windows, doors etc. The complexity of the systems/services requires more fine-grained coordination mechanisms, i.e. via employing Coordinators and Controllers in between Interfacers and Service Provider/Information Holders.

**The presence and distribution of role stereotypes in a system reflects its architectural characteristics** Our analysis of three different systems from the perspective of role-stereotypes has led us to understand that such analysis can uncover that the architectural design of a system follows some architectural design or design principles. In this section, we illustrate some of the insights that can be obtained from such analyses. For understanding these analyses, recall that BitcoinWallet and K-9 Mail are built on the Android framework, while SweetHome3D is a pure-Java desktop application.

In Android apps there is smaller amount of Controllers (2.5% & 2.3%) compared to the pure-Java app (7%). This can partly be explained by the nature of Android applications: they are built upon Android frameworks which encapsulate low-level functionalities of the Android OS. Thus, a number of Controller, Structurer and Interfacer classes at UI and activity management level and collaboration between them might be hidden away.

The Android framework offers encapsulated basic functions such as for example persistence, activity life-cycle management. As a result, fewer control logic needs to be created as part of an overall application. In SweetHome3D, persistence tasks are implemented via basic java.io functions and the user interfaces are mostly implemented by using and customizing Swing components. Moreover, being a pure Java app, extra control-logic is needed for handling portability across different execution environments (such as different operating systems (Windows and Macintosh)). Implementing these could result in extra control- and data-units. For this we can expect a higher occurrence of Controller and Information Holder classes).

For the two Android applications, we observe that the portion of Interfacers in BitcoinWallet is much higher than the portion of Interfacer in K-9 Mail. On the other hand, K-9 Mail contains more Service Providers (41.5%) compared to BitcoinWallet (25.7%). This suggests that BitcoinWallet is required to handle a greater amount of user-interaction and a smaller amount of actual transactional services, whereas K-9 Mail has a greater focus on building business mailing services. This is aligned with a finding from the paper [210] by Bagheri et al. The authors, via studying a set of 200 Android applications in Google Play store, found out that Android applications in different domains have different architecture characteristics regarding type and number of components. In particular, *finance apps*, such as BitcoinWallet and other banking or payment

systems, provide richer user-interface compared to the other kinds of apps. App for which *communication* is the key feature, such as K-9 Mail, largely depend on listening, receiving and handling system events. Such events in turn are typically handled by a Service Provider-type of class.

The role a class plays within a software system reflects design intention. Design intention is, in its turn, affected by architecture style, choices of technology/library use and domain-specific requirements. Therefore, it is possible to use role-stereotypes of as a tool for profiling/capturing software systems' design style and intention. This also enables a possibility to compare designs of different software systems via their role-stereotypes.

### 8.8.3 Collaboration Pattern between Stereotypes

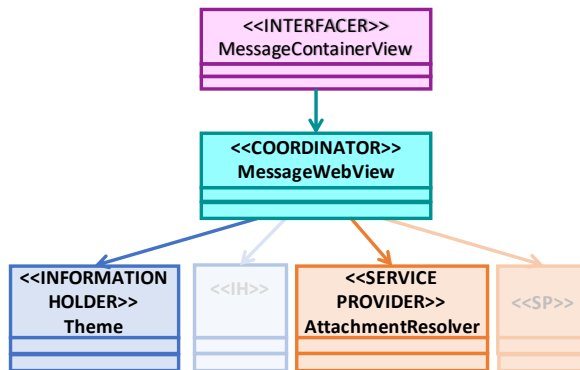


Figure 8.6: Typical Fragment of Collaboration between Stereotypes (Example from K-9 Mail)

To find patterns, we firstly created reverse-engineering class diagrams of the three cases using the parsed srcML files and plantuml<sup>8</sup>. Then, we visualize the role of every class by coloring based on its role-stereotype. The visualisation and its source (in *plantuml* format) can be found in the replication package of this study [201]. The following findings are based on the researchers' visual assessment of patterns in a visualisation architecture models of the three cases.

Fig 8.6 shows a typical pattern of collaboration between role stereotypes in K-9 Mail case: on the top, there is an Interfacier which receives UI-events and reacts to the events by sending messages to an associated Coordinator. This Coordinator contains logic to break down the task and to pass requests to one or more associated Service Providers and one or more associated Information Holders. The Coordinator is also responsible for gathering the results from the Service Provider(s) and Information Holder(s) and finally returning the results to the Interfacier. Via our visualisation, we have found multiple occurrences of this pattern in K-9 Mail and Sweet Home 3D. This pattern empirically confirms the architectural control style for user events described by Wirfs-Brock (p.

<sup>8</sup><http://plantuml.com/>

207 [193]). Moreover, the repeated occurrence of this pattern of stereotypes implies that there is a regularity in the design of software systems that can be made visible by looking through the 'lens' of role-stereotypes.

## 8.9 Threats to Validity

*Threats to Internal Validity.* An OOP class may be responsible for multiple roles and responsibilities [193]. In this study, we however have chosen to build a machine learner that captures only one role for each class. This choice might cause an incomplete view of roles and responsibilities of a single class. However, given that only a low number of classes carry multiple roles (68 out of 1547 classes,  $\approx 4.4\%$ ), we consider this threat is acceptable and is a trade-off to be made to keep our classification model rather simple.

*Threats to External Validity.* In this study, our machine-learning classification model was trained and evaluated on two Android projects and one pure Java project. There might be threats to the generalization of the classification model to other projects. In the future, we plan to extend the ground truth and possibly retrain the classification model proposed in this study with more projects, e.g. more pure Java projects or projects in other languages than Java.

We believe the methods used in this study can be generalized to other OOP systems in various programming languages because of the following reasons: i) the notion of class role-stereotype applies to OOP in general, regardless implementation programming languages; ii) scripts [201] used in this study can be used to extract source code features from different languages than Java. The XPath queries that was used to extract features from parsed srcML files can be adapted to other languages such as C#, C/C++ by following srcML language and grammar rules<sup>9</sup>.

We however would not generalize the result of this study to programming languages and mechanisms other than OOP.

## 8.10 Conclusion and Future Work

In this paper we presented a machine learning based approach for the automatic classification of role-stereotypes of classes of Java software. We find that the Random Forest algorithm enhanced by SMOTE resampling to address imbalanced data yields the best performance for the multi-class classification with an F1-Score = 0.88. For the binary classification, we experienced challenges with an imbalanced dataset, i.e. some role-stereotypes are rare compared to others. On the imbalanced data, the classifier performs good (MCC score = 0.60) at detecting some role stereotypes (Controller, Information Holder and Service Provider, medium good (MCC score = 0.52) at detecting others (Interfacer) and poor (MCC score  $\leq 0.37$ ) at detecting a third category (Coordinator and Structurer). Partially this can be explained by the low frequency of occurrence of these roles in the design (offering few training examples). The SMOTE resampling technique increases the number of rare role-stereotypes in the dataset, thus resulting in a more balanced training data. Ultimately, we could

---

<sup>9</sup>srcML grammar rules: <https://www.srcml.org/documentation.html>

achieve better performance for classification of all role-stereotypes with MCC Score between 0.74 and 0.98.

At this point of the research, we identified some other directions that could potentially help to improve the performance of the classifier: i) using a probabilistic iterative approach, i.e. roles with high-precision can swing the classification of other classes based on the relationship between these classes, ii) combining machine learning with the rule-based approach (in an 'ensemble method', thereby exploiting that rules are not sensitive to small numbers of training data), iii) combining individual binary classifiers, and iv) using deep learning methods (although this probably requires a much larger training set).

The fact that the classifier work in an automated way enables the rapid labelling of role-stereotypes for large collections of classes, and in practice for entire source code of systems. This, in turn, enables novel analysis that sheds light on the anatomy of software designs, such as for example analyses of the frequently of particular collaboration patterns between different stereotypes. We believe there are yet undiscovered regularities in the anatomy of software designs that can be uncovered through further studying these role-stereotypes for larger sets of projects. Moreover, now that we have a robust automatic classifier for role-stereotypes, we can study advanced questions such as: How do the distributions of role-stereotypes differ across different types of architectures or business domains? Can role-stereotypes help us to better understand evolution of software over time? Can role-stereotypes be used for tailoring test-generation strategies?



# Chapter 9

## Paper H

**Interactive Role Stereotype-Based Visualization To Comprehend Software Architecture**

**T. Ho-Quang, A. Bergel, A. Nurwidyantoro, M.R.V. Chaudron**

*Under submission.*





## Abstract

*Motivation:* Software visualization can be helpful in comprehending the architecture of large software systems. Traditionally, software visualisation focuses on representing the structural perspectives of systems. In this paper we enrich this perspective by adding the notion of *role-stereotype*. This role-stereotype carries information about the type of functionality that a class has in the system as well as the types of collaborations with other classes that it typically has.

*Objective:* We propose an interactive visualization called *RoleViz*, that visualizes system architectures in which architectural elements are annotated with their role-stereotypes.

*Method:* We conducted a user-study in which developers use RoleViz and Softagram (a commercial tool for software architecture comprehension) to solve two separate comprehension tasks on a large open source system. We compared RoleViz against Softagram in terms of participant's: (i) perceived cognitive load, (ii) perceived usability, and (iii) understanding of the system.

*Result:* In total, 16 developers participated in our study. Six of the participants explicitly indicated that visualizing roles helped them complete the assigned tasks. Our observations indicate significant differences in terms of participant's perceived usability and understanding scores.

*Conclusion:* The participants achieved better scores on completing software understanding tasks with RoleViz without any cognitive-load penalty.

## 9.1 Introduction

Software architecture visualization is a tool that can be used to understand complex software system. It can help developers maintain and further develop the system. In particular, it can be utilized to improve the search, navigation, and exploration of software architecture design [211] [212].

In UML, stereotypes are a way to add complementary semantic information to the elements of a software design. Using such stereotypes in visualisation has been demonstrated to aid in the comprehension of software architectures. For instance, Genero et al. use object interaction stereotypes to improve the comprehension of UML sequence diagram [56]. Another example, Ricca et al. propose the use of web-specific notations to make UML applicable to model web application [55]. Beside those, a number of work focus on investigating the usefulness of class stereotype [213] to better understand UML class diagram [57] [58] [59] [214].

The well-known class stereotypes, namely boundary, control, and entity, were introduced by Jacobson et al. as an extension to UML [213]. However, their definition of stereotypes is quite simple. Alternatively, Wirfs-Brock proposes role-stereotypes as the responsibilities that a class can have in an object-oriented system [189]. Some of both stereotypes are similar (e.g. *entity* and *information holder*), but Wirfs-Brock provides additional stereotypes beyond the class stereotypes. For example, a *service provider* is a class that performs work and offers services to others, which is not fit in any class stereotypes definition. To the best of our knowledge, no visualization tool has utilized role-stereotypes to help understand software architecture.

In this paper, we present RoleViz, a role-stereotypes-based visualization tool, and evaluate its usefulness to understand the architecture of an object-oriented system. We use the role-stereotypes [189] of a manually labeled ground-truth provided in [204]. Our study shows the effectiveness of RoleViz to help developers in realistic software comprehension tasks.

**Contributions.** This paper makes the following contributions:

- We present RoleViz, an innovative visualization tool that overlay roles on top of a software architecture.
- We conduct a user study to investigate how RoleViz can help developers in real comprehension task, e.g. bug fixing.
- We compare the effectiveness of RoleViz against Softagram as our baseline. Softagram is a well-known software architecture visualization tool commonly used by software developers and architects.

**Outline.** The paper is structured as follows: 9.2 provides background of Wirfs-Brock's role stereotypes; 9.3 describes the RoleViz visualization; 9.4 presents the research questions that leads our evaluation; 9.5 presents the user-study we conducted in order to answer the research questions; 9.6 describes sources of data we collected and the methods for analysing the data; 9.7 presents the result of our analysis; 9.8 discusses a number of aspects of our work (including threats to validity); 9.9 briefly presents the related work; 9.10 concludes and outlines our future work.

## 9.2 Role Stereotype

Our visualization is centered around the notion of *role* of an object-oriented class. Wirfs-Brock [189] identified six stereotypical role types that a class can play:

- (CT) *Controller* makes decisions and control complex tasks;
- (CO) *Coordinator* does not make many decisions, but in a rote or mechanical way, delegates work to other classes;
- (IH) *Information holder* holds certain information and provides that information to others;
- (IT) *Interfacer* transforms information and requests between distinct parts of a system. It can be a user interfacier class that interacts with users. An interfacier can communicate with external systems or between internal subsystems;
- (SP) *Service provider* performs specific work and offers services to others on demand;
- (ST) *Structurer* maintains relationships between classes and information about those relationships. Structurers might pool, collect, and maintain groups of classes.

It is noted that each class should play at least one role. There is a possibility where a class may carry more than one role. In this study we decided to only consider the primary responsibilities of the class as documented in the replication package of [204] where the authors attempted to classify role-stereotypes of a class automatically.

## 9.3 RoleViz

We will use the K-9 Mail application as the running example to illustrate RoleViz. K-9 Mail is an open source alternative mail application in Android. K-9 Mail is composed of 779 classes distributed in 52 different packages. K-9 Mail totals over 97 kLOC. Note that although K-9 Mail is written in Java, RoleViz is not tied to the Java programming language or Android platform.

### 9.3.1 RoleViz in a Nutshell

Figure 9.1 shows the use of RoleViz on K-9 Mail. RoleViz locates K-9 Mail's 52 packages in a circular fashion. Each package contains abstract class, class, enum, and interface. Each structural unit is colored according to the role it has.

Dependencies between two packages are represented with a *bimetric line* (number of dependencies are mapped to the size of the extremities, as described below). The package `k9` has classes heavily used in the system (indicated with tall inner colored boxes, marked with **A**), while `activities` has classes with outgoing dependencies variables (indicated with wide inner colored boxes,

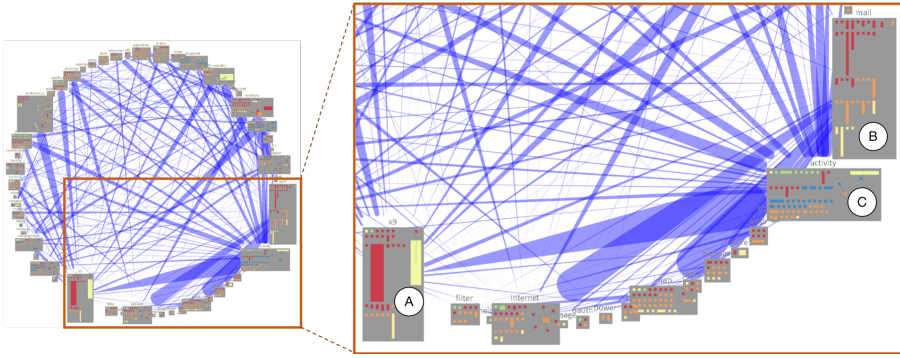


Figure 9.1: Example of RoleViz

marked as c). Although, the application does not exhibit an architecture with crystal clear modularity boundaries, some tendencies may be visually inferred: for example, many packages depend on the package `k9`, while `k9` has relatively few external dependencies. Similarly, many packages depend on the package `mail`.

RoleViz is a polymetric view [215] in which software metrics are applied to visual dimensions, including height, width, and colors, as described below.

### 9.3.2 Compilation Unit

The source code in Java is organized as *compilation unit*, which is a technical jargon in Java to designate a definition contained in a `.java` file. We will, therefore, use this term along this paper to refer to a class, an enum, an abstract class, or an interface. Each unit is represented as a colored box, contained in a package.

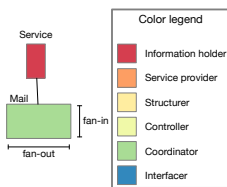


Figure 9.2: Compilation unit detail

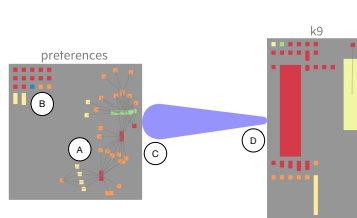


Figure 9.3: Package detail

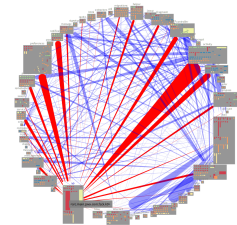


Figure 9.4: Highlighting a package

9.2 details the visual representation of a compilation unit. The visual representation of a unit  $U$  uses two metrics:

- [a] the height of a unit represents the fan-in, *i.e.*, number of units that depends on  $U$ ;
- [b] the width of a unit represents the fan-out, *i.e.*, number of units that  $U$  depends on.

The shape of the box is, therefore, an indicator for visually spotting *exceptional entities* [216]. For example, in the K-9 Mail example (9.1), one can recognize classes with a high fan-in value (marked as **A** and **B** in the figure) and a high fan-out (**C**). The visual shape is not meant to give an accurate value of the associated metrics, but instead, to give an idea of where significant visual differences lay in the visualization. As indicated below, in Section 9.3.4, the visualization offers a number of interactions to obtain details about exact numerical values and offer numerous options to drill-down complementary information. A unit color indicates its role.

Edges between units indicates dependencies between these units. To not overload the visualization, edges are presented as bidirectional (*i.e.*, one cannot distinguish a caller from a callee). Hovering the mouse above a unit highlight callers and callees, as described below, as described in in Section 9.3.4.

### 9.3.3 Package

9.3 details the representation of a package. A package is represented as a labeled gray box. The label, located above the gray box, is extracted from the name of the represented Java package.

The gray box contains inner colored boxes, representing the compilation units contained in the package. Units having dependencies between them are located on the right hand-side using a force-based layout (*i.e.*, units are assimilated as repulsing magnets and edges as springs, **A** in 9.3). Note that edges between units are scoped to the package, *i.e.*, only dependencies between units that belong to the same package are represented. Units not connected with other units within the same package are simply located as a grid and sorted by their role (**B**).

Dependencies between packages are deduced from the dependencies between units. Inter-package dependencies are represented using a *bimetric line*, in which the number of dependencies from the package `preferences` to `k9` is represented by the extremity size on the package `preferences` (**C**). Similarly, dependencies initiated in `k9` toward `preferences` are represented in the extremity size close to `k9` (**D**).

Such a bimetric line is adequate in presence of multiple birectional connections. 9.3 clearly indicates that `preferences` heavily depends on `k9`, while `k9` depends little on `preferences`.

### 9.3.4 Interaction

RoleViz offers a number interactions to ease the exploration of the software under analysis.

**Mouse hovering.** Hovering the mouse cursor above a package highlight in red dependencies between dependent and depending packages. 9.4 illustrates the overall K-9 Mail application with `k9` highlighted. In addition, a popup appears to give the full package name of it. The figure shows that `k9` has little dependencies toward other packages however many are depending on `k9`.

When hovering the mouse cursor above a compilation unit, lines between the pointed unit toward all dependent other units appear (not shown in the figure). Lines are also colored according to the role of the dependent class.

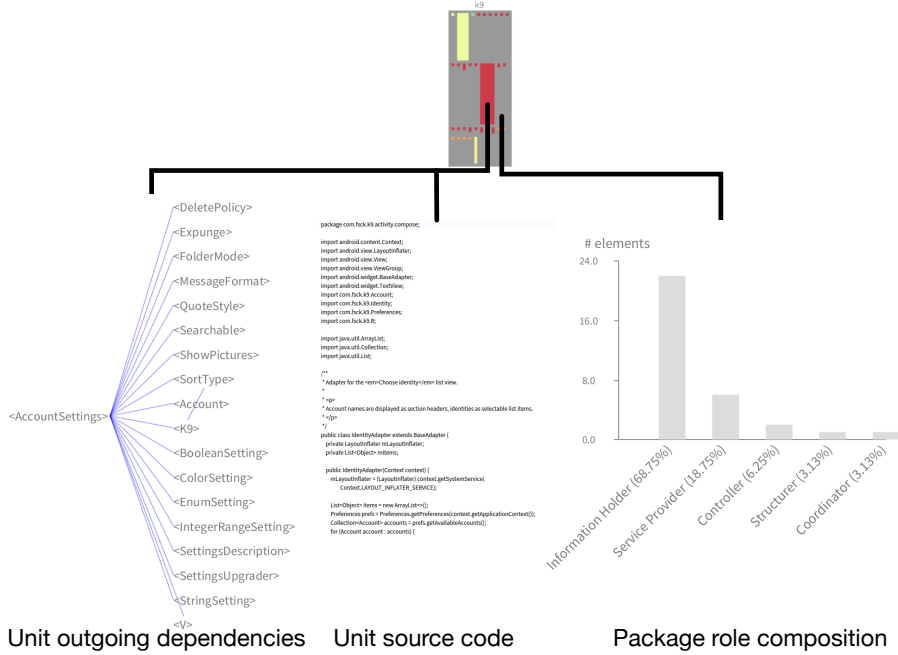


Figure 9.5: Drill down

**Drill down.** In a graphical environment, drill-down is an action to obtain detailed data about a particular visual element. Clicking on a package augment the main visualization with the *package role composition* histogram, indicating the proportion of different roles. In 9.5, the histogram indicates that 68.75% of the compilation units contained in the `k9` package have the *Information Holder* role.

Clicking on unit shows two views. *Unit outgoing dependencies* is a visualization that indicates the outgoing dependencies of the selected unit. *Unit source code* gives the source code, in which one can search using regular expressions. The view obtained when drilling down are displayed next to the main RoleViz visualization. For example, the source code may be shown all the time while using RoleViz.

**Visualization Alteration.** RoleViz offers five actions to alter the visualization. (i) First, packages and classes matching a provided a regular expression may be highlighted using a stark color. Such a feature is useful to highlight a particular cross cutting concern. The highlight remains until the user decides to explicitly remove it. (ii) Second, selected elements may be kept while all the others are removed. (iii) Third, selected elements may be removed. (iv) Fourth, the visualization can be reset to its original state, thus removing all the alterations. (v) Fifth, the visualization may be spawned into a new window, thus leading to a second instance of the visualization. This interaction allows for parallel unrelated system explorations. This can also be used in combination with the other alteration actions to produce a new visualization with smaller number of elements, e.g. the ones that match the search terms, thus allows users to focus

on a specific parts of the system.

These interactions alter the visualization. As a consequence, they are likely to be triggered after a shallow exploration using mouse hovering and drill down.

## 9.4 Research Questions

The research objective of our study is to determine whether RoleViz helps in enhancing the understandability of software architecture. We form two research questions to guide our study:

**RQ1:** *How does RoleViz compare to Softagram?* In particular, we compare the two visualisation tools in terms of:

- participant’s perceived cognitive load,
- participant’s perceived usability,
- participant’s understanding of the software system regarding the tasks.

By “understanding”, we refer to the participant’s ability to: a) locate components/entities of the system relevant to the tasks, b) describe the responsibility of the located components/entities and relationship between them, and c) formulate a plan to solve the tasks.

**RQ2:** *What are the perceptions of the participants on the current features of RoleViz?*

Determining whether RoleViz meets the expectation of the participants is crucial to identify where exactly RoleViz falls short of feature. In addition, this research questions helps formulating the future direction of RoleViz.

## 9.5 User Study

To answer the research questions stated above, we designed and conducted a user study. The design of the user study involves the following five components.

### 9.5.1 Baseline

The performance of RoleViz has to be compared against a baseline visualization. Softagram, which is a commercial tool to visualize software system<sup>1</sup>, was chosen to be the baseline tool for two main reasons.

Firstly, Softagram has been defined to address concrete problems of visualizing software architecture and it has been developed under a strong industrial influence. The visualization metaphor is UML-inspired: a software entity (*e.g.* a file, a package or a class) is represented as a node with attributes and links to other nodes. The associations between software entities are used to show various types of relationship (between the entities), such as inheritance, library usage, method-calls, etc. 9.6 shows K-9 Mail with Softagram. At the center we see

---

<sup>1</sup><https://softagram.com>

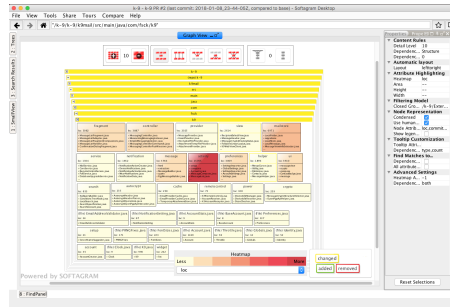


Figure 9.6: Softagram main GUI - Structural View

different packages, to which the red fading indicates a metric, number of lines of code in this example. Different layouts are accessible from the control panel located on the top of the window. On the right-hand side different properties to adjust the visualization are available.

Secondly, Softagram allows for a software exploration in an interactive fashion. In particular, users can drill down/up to navigate among levels of data ranging from the top package (up) to variables of a class (down). Mouse scroll can be used to zoom in/out at specific parts of the visualization canvas, thus allowing users to read details when the diagram is too large. Similarly to RoleViz, associations of an entity are highlighted when clicking on the entity. Moreover, Softagram also provides two search options which allow users to search globally in all entities of the studying system or locally within the entities showed on the main canvas.

Softagram can also be used to highlight architectural changes (such as new dependencies) introduced by the pull request author. Softagram does not offer source code view within the application but can direct users to the Github page of the source code file (via a web browser and the Internet).

## 9.5.2 Comprehension Tasks

We need to define two comprehension tasks. We started by defining a number of criteria (C) as the following:

- C1:** *Realistic.* The comprehension tasks should be derived from realistic software development or maintenance issues/tasks.
- C2:** *Simple.* The tasks should be simple enough so that participants can complete them within the limited time of the study.
- C3:** *Independent.* The two tasks should not depend on each other and should not be semantically close. As we use a within-subject method, this criterion aims to mitigate the learning effects from solving one to another task.
- C4:** *Comparable.* The two tasks should be comparable in terms of complexity. With this criterion, we expect the differences between the tasks do not create any additional cognitive load or lead to any major changes in the performance of participants.



**C5:** *Verifiable.* We assess participant’s understanding based on their solutions to the tasks. The assessment method should be built on top of a verified solution to the tasks. Therefore, it is important to find the tasks but also the solutions that are confirmed to solve the tasks.

Then, we looked into the issue tracking system of K-9 Mail to find realistic issues (**C1**). The issues were labeled by senior contributors of the project. We relied on these labels in order to filter relevant tasks for the study. In particular, we filtered those issues that were labeled as **good first issue** (for simplicity and comparability - **C2** & **C4**) and were solved/closed at the time of searching (so a working solution exists - **C5**). We found two issues namely “*Export/Import Settings*” (#2969) and “*Attachment Size Format*” (#3343) that satisfied the criteria.

Then, two comprehension tasks were built on the basis of the identified issues. Task “*Export/Import Settings*” (EXPORT/IMPORT) concerns with the problem that email settings (including preferences, contacts, etc.) do not display in the same order when being exported and imported from an old to a new Android device. Task “*Attachment Size Format*” (ATTACHMENT SIZE) aims at changing the size format of downloaded attachments from long number of bytes to a more human-readable form (e.g. in KB, MB, GB). The two tasks are independent and are not semantically close (**C3**).

It is noted that the main aim of the comprehension tasks is to locate and build up understanding around the part(s) of the software system that is(are) relevant to solving the given issues, not to implement or evaluate specific code changes. Details of the tasks can be found in the replication package of this study [217].

For each selected task, the following information was collected:

- The *description of the issue* is collected directly from the issue tracking system. The instructors do not modify or add any text to the description.
- The *discussion about the issue* includes messages regarding the issue and relevant/similar issues. Instructors of the user study focus on building up knowledge on the following two aspects when browsing the discussion: i) context/clarification of the issue; ii) solution(s) to resolve the task: This is the part of discussion where solutions are discussed.
- The *implementation of the solutions* is assessed against the code approved by the K-9 Mail community.

During the user study, participants are given the description of the issue only. The instructor of the user-study have access to all the information and used it to: i) build up understanding about the issue and the context where it arises. With this, the instructor is expected to be able to answer participant’s questions regarding the task during the user study; ii) create a grading schema for assessing participant’s understanding. We elaborate on the grading schema in Section 9.6.4.

### 9.5.3 Participants

The user study targets participants who have some kinds of experience with software development in Java programming language. The participants do not

need to have prior understanding on the role-stereotypes or have any experience of using software comprehension tools.

We sent a call for voluntary participation to the user study via personal networks of the authors. In the call, the following information was clearly mentioned: i) a short description about the study; ii) requirements to the participants; and iii) expected time and duration of the study as well as expected amount of work from the participants. After two weeks, we received numerous responses and could finalize a list of 16 participants for the study. 2 weeks prior to the working session of the study (see Section 9.5.5), an email with training materials (see Section 9.5.4) and an URL to the online background form was sent to the participants. The instructor of the study also communicated with the participants in order to schedule time and location for the working session.

### 9.5.4 Training Period

The aim of the training period is to equip the participants with essential information to work effectively during the working session. By essential information, we refer the following pieces of knowledge: i) use of visualisation tools, i.e. functions and interaction mechanism of the visualisation tools; ii) role-stereotypes, i.e. what is the responsibility of each role.

We provided the participants with several training materials, including presentation slides about role-stereotypes and two self-designed tutorial videos on RoleViz <sup>2</sup> and Softagram <sup>3</sup>. These training materials were sent to the participants two weeks prior to the working session. The materials are included in the replication package of this study [217]. During this training period, the instructor was open to any questions regarding both of the tools and role-stereotypes.

### 9.5.5 Work Session

After the training period, we assume all participants have proper knowledge to start working on the comprehension tasks. The working sessions were designed to be 80 minutes long and were conducted on a desktop computer provided by the instructor in a scheduled time and room. All participants used the same screens and input devices. The activity (**A**) of a participant was structured as follow:

**A1:** *Introduction (5 mins):* The instructor gave a brief introduction about the purpose and procedure of the session.

**A2:** *Warm-up (15 mins):* During this time, the participant was allowed to actually use the visualisation tools. The main aim was for the participant to be more familiar with the control and interaction mechanism of the tools. The participant was also allowed to adjust the settings of the desktop computer and input/output devices (such as keyboard, mouse, screen) to fit his/her preferences.

---

<sup>2</sup><https://youtu.be/HqCUA1ai4qw>

<sup>3</sup><https://youtu.be/YXizTrJ5j7I>

**A3:** *Comprehension sessions (50 mins):* Each participant performed two comprehension tasks (ATTACHMENT SIZE and EXPORT/IMPORT), each with help of a visualisation tool (RoleViz or Softagram). Each comprehension session was scheduled in 25 minutes with the following activities:

**A3.1:** Giving task description (3 min);

**A3.2:** Comprehending with a visualisation tool (15 mins);

**A3.3:** Answering post-task questionnaire (7 mins).

**A4:** *Post-study Questionnaire (10 mins):* Participants were asked to answer open questions regarding their perceived benefit of using RoleViz and desired improvements of the tool.

## 9.6 Data Collection & Analysis

We collected the following data (i) background information, (ii) NASA Task Load Index (TLX) Questionnaire, (iii) System Usability Scale (SUS) Questionnaire, (iv) Understanding Questionnaire, (v) Video Recording, (vi) Post-study Questionnaire. Next, we discuss how the data is collected and analyzed.

### 9.6.1 Background Questionnaire

Prior to the working session, participants were asked to fill in a background questionnaire. The questionnaire contains 10 questions regarding participant's experiences with Java programming language, Android and K-9 Mail system. If a participant answers that he knows/has experience with K-9 Mail, 2 extra questions are asked for clarification about this.

### 9.6.2 TLX Questionnaire

**Measurement.** The NASA-TLX is a widely used technique for measuring subjective mental workload [218]. It relies on a multidimensional construct to derive an overall workload score based on six workload sources: mental demand, physical demand, temporal demand, performance, effort, and frustration level. There are two ways to compute the total workload score. One way, called *Weighted TLX*, involves a two-step process where participants first give rating for the six workload sources, then make a series of 15 pairwise comparisons between each pair of the sources as a basis for calculating weight of each source. The second way, called *Raw TLX*, is a light-weight approach in which the total mental workload score is simply calculated as the average of the 'raw ratings' of the six workload sources [219]. In this study, we chose to follow this light-weight approach to collect TLX data and calculate the total TLX score.

**Data collection.** After finishing a comprehension task (**A3.2**), participants were directly given a TLX rating sheet in paper form and a pen to mark on it. In total, each participant gave two rating sheets after the two comprehension sessions. We collected the sheets and transferred the result into a csv file for computational purpose. The instructor only gave explanation or clarification

regarding the TLX scale based on NASA's TLX manual [220]. The instructor did not interfere or influence participant's ratings in any mean.

**Data analysis.** We compare the mean values of TLX scores between the two tasks in order to see the workload. Since our study is within-subject, we will use Wilcoxon signed-rank test to measure the differences.

### 9.6.3 SUS Questionnaire

**Measurement.** The System Usability Scale is an easy, standard way of evaluating the usability of a system [172]. It is a form containing ten statements, and users provide their feedback on a 5-point scale (1 is "strongly disagree" and 5 is "strongly agree"). It effectively differentiates between usable and unusable systems by giving a measure of the perceived usability of a system. It can be used on small sample sizes and be fairly confident of getting a good usability assessment [221].

**Data collection.** The ten SUS questions were integrated into the post-task questionnaire (**A3.3**). The participants were given the questionnaire after finishing with a comprehension task and the corresponding TLX ratings paper.

**Data analysis.** We follow the formula proposed by Brooke [172] to calculate the total SUS scores reported by the 16 participants. After that, we calculated the average of the usability values of all participants split by visualization tool to obtain the overall usability score of RoleViz and Softagram. We compare these values in order to examine the difference in usability of the two tools. In order to obtain a more detailed view of the difference (if any), we compare mean values of ratings to each of the 10 SUS questions between the two tools. We test the significance of the differences by using a Wilcoxon signed-rank test which is non-parametric and is often used in situations in which there are two sets of scores derived from same participants [222].

### 9.6.4 Understanding Questionnaire

**Measurement.** In order to measure participant's understanding of K-9 Mail system regarding to the tasks, firstly, we ask the participants to answers the following three questions (**Q**).

- Q1. Can you name 5 elements (packages/classes/methods) that are the most relevant/important to the task?
- Q2. What are the responsibilities of the elements chosen for the question above in performing the functionality related to the task?
- Q3. Which changes of the elements chosen for question above are needed to complete the task? (Describe your plan/solution)

These three understanding questions aim to assess the three aspects of "understanding" (as defined in Section 9.4).

Next, we build and use a 11-point scale grading schema, *i.e.*, with the lowest score being 0 and the highest score being 10 points, to evaluate participant's answers. For each comprehension task, a grading schema is created by (same) one author of this paper based on the three sources of information regarding the task, including *description of the task*, *discussion about the task* and

*approved implementation of solutions to the task* (as described in Section 9.5.2). The grading schema consists of answers to the three understanding questions and criteria to judge the level of participant’s understanding toward each questions. It is noted that different questions are given different maximum points based on our subjective judgment on their importance to forming participant’s “understanding”. In particular, answers to Q1, Q2 and Q3 could get maximum 5 points, 2 points and 3 points, respectively. More details about the grading schema can be found in the replication package of this paper [217].

**Data collection.** The 3 understanding questions are placed in the post-task questionnaire together with the 10 SUS questions (**A3.3**). During the comprehension time, participants were encouraged to take note about the relevant elements of the system to the tasks, thus they could quickly transfer their notes to the answer form. Their answers were then graded by two authors of this paper using the above-mentioned grading schema. Total understanding score was calculated as a sum of the three component scores.

**Data analysis.** Similar to TLX and SUS score, we compute the total understanding scores of all participants and compare the mean values of understanding scores between the two visualization tools and the two tasks. In order to gain an insight about which aspect(s) of “understanding” contribute to the difference (if any), we also compare the scores between the understanding questions by visualization tools and tasks. We test the significance of the differences by using a Wilcoxon signed-rank method.

### 9.6.5 Post-study Questionnaire

**Data collection.** All participants were given a post-study questionnaire (**A4**) after they have finished with two comprehension sessions. The post-study questionnaire contains 7 open questions aiming at collecting participant’s perceived benefit of using RoleViz in program comprehension and desired improvements of the tool.

**Data analysis.** We used the Grounded Theory research method [223] to analyze answers of the post-study questionnaire. We first identified concepts and key phrases are identified and moved into subcategories, and then grouped into categories.

## 9.7 Result

In this section, we first present demographics of the participants of this study. Then, we explore the comparability of the two comprehension tasks used in the study. Lastly, we answer the two research questions of the study.

### 9.7.1 Demographics of Participants

In total, 16 people, with ages ranging from 23 to 36, participated in this study. These include 7 Master students, 4 Ph.D. candidates, 1 post-doc researcher and 4 software development engineers from 3 software companies. All of the participants have some experiences with the Java programming language, ranging from less than 1 year (2 participants) to more than 8 years (1 participant).

The majority of the participants (10 out of 16) have 3-8 years experience with Java.

10 out of 16 participants reported to be familiar with the Android development framework. Among them, 5 participants have less than 1 year of experience, 2 participants have 1-2 years experience and 3 participants have 3-5 years of experience with the Android framework.

Only 5 out of 16 participants answered to know the K-9 Mail application and/or the K-9 Mail development project. 3 of them have been using the K-9 Mail application in a daily basis for managing emails on their Android devices. None of the participants reported to have comprehended the K-9 Mail system prior to the study.

14 participants watched the introduction videos of RoleViz and Softagram prior to the work session. For the two participants who did not watch the introduction videos, the instructor spent extra time (15 - 20 minutes) at the starting of the work session to guide them through important parts of the videos.

## 9.7.2 Are the Comprehension Tasks Comparable?

The two comprehension tasks used in our study were carefully selected (as described in Section 9.5.2) with the expectation that the tasks are comparable. In this section, we examine this comparability in terms of participant's TLX, SUS and Understanding Questionnaire scores.

Figure 9.7 shows the mean values of participant's perceived TLX, SUS and Understanding scores for the two comprehension tasks. A Wilcoxon signed-rank test confirmed that the small differences are statistically insignificant, with *p-values* being *0.80*, *0.85* and *0.17* (all are well-above 0.05) for TLX, SUS and Understanding scores, respectively. We therefore conclude:

- The two comprehension tasks require similar cognitive load to solve (TLX score).
- Solving different tasks does not result in different perceived usability score (SUS score).
- Participants achieved comparable understanding scores after solving the two tasks (Understanding score).

The meaning of this result is two-fold. Firstly, it confirms that our task selection method is effective. Secondly, it suggests that we can ignore the factor of “task-difference” when analysing the difference between visualisation tools.

**The two comprehension tasks are comparable in terms of complexity, required cognitive-load, usability score and understanding score. With this, we can eliminate the “task-difference” factor when analysing the difference between visualisation tools.**

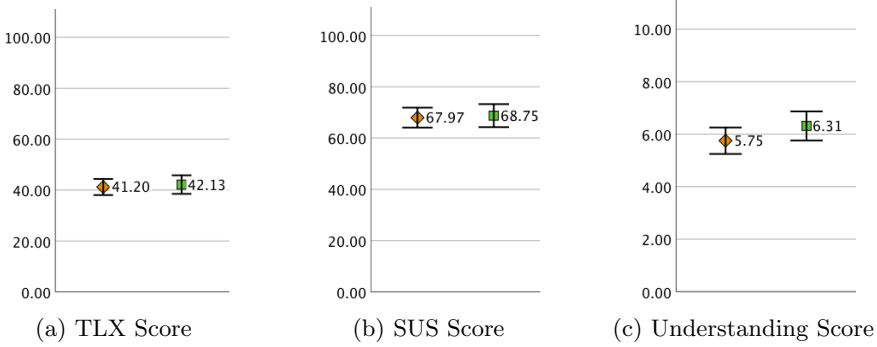


Figure 9.7: Differences in mean values of (a) TLX, (b) SUS and (c) Understanding Scores ( $\pm 1$  SD) between two comprehension tasks: ATTACHMENT SIZE ( $\blacklozenge$ ) and EXPORT/IMPORT ( $\blacksquare$ )

## 9.7.3 RQ1: Comparison between RoleViz and Softagram

### 9.7.3.1 TLX Task Load Score

Table 9.1 shows mean values of the overall- and component TLX scores across the two visualisation tools. The average task load index associated using RoleViz and Softagram in the comprehension tasks are  $39.43 \pm 13.24$  and  $43.91 \pm 13.68$ , respectively. These scores indicate a low to moderate effort according [224].

Table 9.1 shows that the mean values of overall- and component task load associated with using RoleViz are always smaller, with the differences ranging from 0.63 to 12.19, compared to that of Softagram. The Wilcoxon signed rank test, however, shows that none of the differences are statistically significant (all p-values are above 0.05).

Table 9.1: Comparison of average TLX scores by the two visualisation tools (N=16)

		RoleViz		Softagram		Wilcoxon S.R.	
		Mean (M1)	SD	Mean (M2)	SD	M1-M2	p-value
<b>Overall TLX</b>		39.43	13.24	43.91	13.68	-4.48	0.155
<b>Task Load Sources</b>	<i>Mental</i>	52.19	21.68	56.88	21.36	-4.69	0.347
	<i>Physical</i>	24.69	18.02	25.31	18.66	-0.63	0.857
	<i>Temporal</i>	50.63	28.63	54.69	26.86	-4.06	0.262
	<i>Performance</i>	33.75	15.33	36.25	24.87	-2.50	0.975
	<i>Effort</i>	42.19	17.89	54.38	20.65	-12.19	0.088
	<i>Frustration</i>	33.13	24.07	35.94	24.71	-2.81	0.371

The average task load associated with using RoleViz and Softagram for the comprehension tasks is comparable.

### 9.7.3.2 Usability Score

Table 9.2 shows the mean values of total SUS scores and component SUS scores associated with the two visualisation tools. RoleViz achieves an average of  $72.43 \pm 14.93$  in overall SUS score. This is significantly higher compared to that value of Softagram, which is  $64.32 \pm 17.62$  ( $p$ -value = 0.035). According [225], RoleViz is graded “C+” which indicates a *good* usability score, while Softagram is graded “C” which is considered as a *moderate* usability score.

In order to get an idea on which aspects of usability constitute the difference, we take a deeper look at the ten component SUS scores. We find that RoleViz tends to achieve higher (mean values of) rating to questions regarding the positive aspects of usability (i.e. Q1, Q3, Q5, Q7 and Q9). Meanwhile, Softagram seems to score “higher” for questions regarding the negative aspects of usability (i.e. Q2, Q4, Q6, Q8 and Q10). This plain comparison (of mean values) suggests that RoleViz achieved a “better” usability score for most of all component usability aspects (except for the required learning-effort where the mean values was equal).

A Wilcoxon signed rank test confirms that this difference is statistically significant. In the post-study questionnaire, one participant reported a comment that may explain this difference: “source code is not easily accessible using Softagram”, whereas in RoleViz it is easy to navigate between design and source code perspectives.

Table 9.2: Comparison of average SUS scores between RoleViz and Softagram (N=16)

		RoleViz		Softagram		Wilcoxon S.R.	
		Mean (M1)	SD	Mean (M2)	SD	M1-M2	p-value
<b>Total SUS Score</b>		72.34	14.93	64.38	17.62	7.97	0.035*
<b>Usability Measurement</b>	Q1: <i>Willing to use the system</i>	3.50	1.10	3.25	1.13	0.25	0.210
	Q2: <i>Complexity of the system</i>	2.00	1.03	2.56	0.63	-0.56	0.090
	Q3: <i>Ease of use</i>	3.75	0.93	3.38	1.15	0.38	0.110
	Q4: <i>Need of support to use</i>	1.88	0.81	2.25	1.13	-0.38	0.190
	Q5: <i>Integrity of functions</i>	3.69	0.87	3.13	0.96	0.56	0.020*
	Q6: <i>Inconsistency</i>	1.56	0.89	1.81	0.98	-0.25	0.250
	Q7: <i>Intuitiveness</i>	3.88	0.96	3.88	0.96	0.00	1.000
	Q8: <i>Cumbersomeness to use</i>	1.94	0.85	2.31	1.14	-0.38	0.080
	Q9: <i>Feeling confident to use</i>	3.44	0.81	3.00	1.21	0.44	0.080
	Q10: <i>Required learning-effort</i>	1.94	0.93	1.94	1.12	0.00	0.940

**RoleViz is reported to have a significantly higher usability score compared to Softagram. Participants also valued the high level of integrity of available functions of RoleViz over Softagram.**

### 9.7.3.3 Understanding Score

Table 9.3 shows the mean values of total Understanding scores and component Understanding scores associated with the two visualisation tools. Participants



scored on average  $6.56 \pm 1.82$  points with RoleViz. This is significantly higher (by 10%) compared to an average of  $5.50 \pm 2.28$  points when using Softagram (p-value = 0.025).

To gain an insight into the participant's performance on different aspects of understanding, we calculate and analyse the component understanding scores. Table 9.3 shows that participants achieved higher scores for all three understanding questions. In particular, we observe a difference of 0.31, 0.25 and 0.50 points for the questions regarding *Identification* (of relevant components), *Responsibility* (between the identified components) and *Solution* formation, respectively.

The Wilcoxon signed rank test confirms that participants could indeed produce a better solution to the comprehension tasks with RoleViz compared to Softagram (p-value = 0.033).

Table 9.3: Comparison of average Understanding scores between RoleViz and Softagram (N=16)

		RoleViz		Softagram		Wilcoxon S.R.	
		Mean (M1)	SD	Mean (M2)	SD	M1-M2	p-value
<b>Understanding Score</b>		6.56	1.82	5.50	2.28	1.06	0.025*
<b>Comp.</b>	<i>Identification</i>	2.69	1.20	2.38	1.26	0.31	0.353
	<i>Responsibility</i>	1.44	0.51	1.19	0.54	0.25	0.102
	<i>Solution</i>	2.44	0.63	1.94	1.00	0.50	0.033*

**Participants achieved significantly higher understanding scores (by 10%) and produced better solutions when using RoleViz (for comprehension tasks) compared to using Softagram.**

### 9.7.4 RQ2: Participant's perception on the features of RoleViz

The post-study questionnaire focused on collecting the perceptions and suggestions for improvement of RoleViz. The questions are open and need to be answered in plain English. We applied the Grounded Theory [223] to process the post-experiment feedback (9.6.5).<sup>4</sup> The analysis identified 10 general themes. Below, each general theme is annotated with the number of times it appears in the transcripts and the number of participants who explicitly expressed it. The general themes that we consider as positive are:

- *Usability and Efficiency* (50 occurrences / 14 participants): This theme covers the positive aspects of RoleViz regarding the efficiency (*e.g.*, accuracy of the provided information, helpful, searching, no need to read class names, support comprehension, identifying starting point) and usability (*e.g.*, clarity of the visualizations, narrowing down).

<sup>4</sup>The analysis can be found in the replication package of this paper

- *Role* (12 / 6): Overall, participants have positively perceived the way roles are presented by RoleViz (*e.g.*, “coordinator helpful to identify starting point”, “used only the roles to complete the tasks”).
- *Relevant view* (9 / 8): Participants have explicitly indicated that the main RoleViz visualization is helpful to complete the tasks. The possibility to have the source code always present is also reported as important.
- *Visual aspect* (5 / 5): Few participants have indicated some positive aspects of the visual cue. In particular, it was reported that the circle layout gives a good overview of the system. The highlighting and coloring are perceived as useful.

The general themes that we consider as negative toward RoleViz are:

- *Possibility for improvement* (34 / 15): We asked the participants to answer the question “Do you have any suggestions for improvements in RoleViz?”. All participants but one made suggestions about various aspects of RoleViz. In particular, being able to navigate within the source code by only scrolling and textual searching is a limitation. One participant reported that the use of color is problematic (red usually refers to a problem and we use it to represent the information holder role, cf 9.2).
- *Missing information* (7 / 7): Participants criticized missing information about methods and variable accesses. Currently, methods are listed within the source code, obtained by clicking on a compilation unit. Participants found this not convenient.
- *Issue when showing source code* (7 / 6): Source code is poorly supported by RoleViz.
- *Bugs* (5 / 5): A few bugs were reported, in particular that the legend is not always visible.
- *Issue with the experiment* (1 / 1): One participant found that not all the information provided by the visualization are necessary to solve the tasks.
- *Visual element not useful* (3 / 2): Two participants reported that the information about roles and dependencies between elements are not useful.

**Overall, participants appreciate the usability and efficiency of RoleViz to complete the tasks. In total, 6 participants reported the importance of annotating the software architecture with roles information. Participants missed information about methods, in particular the need to inspect a method call graph was reported by 7 participants. Also, source code should be better supported with syntax highlighting and searching.**

## 9.8 Discussions

We first examine whether participant’s background is correlated with their SUS/TLX/Understanding scores. Then, we discuss possible threats to validity

of our study.

### 9.8.1 Does participant’s experiences correlate with their perceived SUS, TLX and Understanding scores?

In order to answer this question, we examine the correlation between participant’s number of years of experiences with Java and Android (called “Java Exp.” and “Android Exp.”) and their SUS, TLX and Understanding scores. This information was collected via the background questionnaire. As Java Exp. and Android Exp. are ordinal data, we use Spearman’s rank correlation coefficient (a.k.a. *Spearman’s rho*), which is a non-parametric statistical method, for measuring the correlation.

Table 9.4 shows the result of the Spearman’s rho test. Each cell of the table shows Spearman’s correlation coefficient between a pair of the following five variables: Java Exp., Android Exp., Understanding, SUS Score and TLX Score. Cells marked with ‘\*’ or ‘\*\*’ indicate a confidence interval of at least 95% (p-value  $\leq 0.05$ ). There is a significant moderate positive relationship between participant’s understanding score and their experiences with Java language. That is, participants with more years of experiences with Java are likely to achieve a better understanding score.

Table 9.4 also reveals an interesting negative relationship between participant’s TLX score and both their understanding and SUS scores. **It is more likely that participants who report a high task load also give a low usability score and achieve a low understanding score.** We do not observe any significant correlation between participant’s experiences with Android to their SUS/TLX/Understanding scores.

Table 9.4: Correlation between participant’s SUS/TLX/Understanding scores and their experiences.

	Java Exp.	Android Exp.	Understanding	SUS Score	TLX Score
Java Exp.	1				
Android Exp.	0.520**	1			
Understanding	0.453**	0.045	1		
SUS Score	0.032	-0.235	0.12	1	
TLX Score	-0.15	0.092	-0.448*	-0.552**	1

## 9.8.2 Threats to Validity

### 9.8.2.1 Threats to Construct Validity

Participants, who are in the network of the authors of this paper, might be biased towards the visualisation that the authors created. We mitigate this issue by not revealing the authorization of the two visualisation tools until the end of the study.

### 9.8.2.2 Threats to Internal Validity

All of the participants are not familiar with both Softagram and RoleViz prior to the study. The unfamiliarity might hinder participant's effective use of the tools for comprehension tasks, thus results in a low SUS/Understanding score and a high task load index. The training period and the warm-up sessions are involved as part of the study to mitigate this threat. In fact, participant's answers to the question 4 of the SUS form (Table 9.1) indicates a small need of assistance when using the two tools.

### 9.8.2.3 Threats to Conclusion Validity

Our answers to RQ1 base mostly on statistical tests on a small sample size. Therefore, there is a threat that the conclusion might not be representative of our analysis. A mitigation strategy could be to involve more people to the next round of tool evaluation.

## 9.9 Related Work

Several studies investigated the effect of using stereotypes on software comprehension tasks. Staron et al. [214] conducted a set of controlled experiments both in academia and in the industry to evaluate the effect of role-stereotypes on UML models comprehension. They found that stereotypes play a significant role in the comprehension of models. In particular, the participants who used stereotyped models scored more correct answers in tests checking the level of understanding. Moreover, these participants required less time to answer comprehension questions and identify the correct answers.

Genero et al. [56] conducted a controlled experiment to investigate the impact of using stereotypes on UML sequence diagrams comprehension. They analyzed the use of sequence diagrams with and without stereotypes. They found that there is a slight tendency in favor of the use of stereotypes in facilitating the comprehension of UML sequence diagrams.

Ricca et al. [55] run a series of experiments to test whether the use of the stereotyped UML diagrams supports the comprehension and maintenance activities of web applications with significant benefits. They compared the performances of subjects in comprehension tasks where they have the source code complemented either by standard UML diagrams or by stereotyped diagrams. They suggested that organizations can achieve a significant performance improvement by letting their less experienced developers (i.e., juniors) adopt stereotyped UML diagrams for comprehension tasks.

Sharif and Maletic [59] studied the effect of two different stereotyped layouts on the comprehension of UML class diagrams: *orthogonal* and *clustered*. The orthogonal layout minimizes edge crossing and bends and does not use information about the class stereotype in layout positioning. The clustered layout uses information about the class stereotype to position classes into multiple clusters in the diagram. They found that the use of stereotyped-clustered layouts demonstrates a significant improvement in subject accuracy and efficiency in solving problems in comprehension tasks.

In the same direction, Andriyevska et al. [57] designed and conducted a user study to evaluate the effect of using a stereotyped UML class diagram layout on diagram comprehension. Andriyevska and her colleagues suggested that stereotyped UML class diagrams support software comprehension tasks by letting developers build better mental models, hence gain more information about the considered software system.

Yusuf et al. [58] conducted a study to assess the effect of using layout, color, and stereotypes on UML class diagram comprehension. As a mean to achieve their goal, the authors used eye-tracking equipment to collect data on subjects' eye gaze which are then used to analyze the cognitive process involved in the visual data processing. They suggested that the use of class stereotypes plays a substantial role in the comprehension of UML class diagrams. Moreover, they suggested that the use of layouts with additional semantic information about the design is the most effective for diagrams comprehension.

Blouin et al. [212] proposed an interactive visualisation tool for comprehending large meta-models. Their tool, *Explen*, used a model slicing technique to allow users focus on subset of model elements of interest. A comparative evaluation of *Explen* with *EcoreTools*<sup>5</sup> showed that *Explen* outperforms in improving large meta-models understanding.

Unlike the previous work which focused on adding role as an extension of static UML models, we built an interactive polymetric view visualisation without referring to UML models. We also utilise the 6 role-stereotypes invented by Wirfs-Brock [189], which are at a different level of abstraction compared to the class-stereotypes used by other previous work [57] [58] [59] [214]. Our visualisation tool also proven to be scalable, i.e. can visualise 700+ classes, compared to the previous work which focuses on assisting the comprehension task of a subset of the model of interest.

## 9.10 Conclusion and Future Work

In this paper we studied the visualisation of large software systems in order to aid comprehension of the design of the system. In particular, we contribute the use of role-stereotypes and a visualisation-tool called RoleViz. We compare our tool to an industrial tool Softagram via a user study with 16 people.

Results from the study indicate that RoleViz achieves a higher score on usability than Softagram. In particular, users like the integration features that enable exploring a software system at both the design and the source code levels of abstraction. According to Sauro's benchmark data, Roleviz achieves 'good' usability, and Softagram achieves 'ok' usability.

Performing comprehension tasks using RoleViz or Softagram is experienced as comparable with respect to the required cognitive effort. According to the benchmark data, the cognitive load of both tool is rated as 'low to moderate' effort.

Participants achieved about 10% better score on comprehension tasks when using RoleViz compared to using Softagram. Specifically, when using RoleViz, participants perform better in the ability to propose a solution to bug-fixing. The factor that may explain this is: RoleViz has as scoping mechanism that

---

<sup>5</sup><https://www.eclipse.org/ecoretools>

allows users to effectively focus on certain parts of the system that are relevant for the task at hand.

We found that some participants use classes with particular stereo-types as starting-points for particular understanding tasks: for example, for tasks that deal with 'user interface' issues, participants start their exploration of the system by looking at classes labelled as 'interface'-type.

For the future directions of the research, the participants in our study prominently pointed out at the need for also visualizing behavioral information (*e.g.*, call-graphs) at a level between source code and architecture level of abstraction.

# Bibliography

- [1] O. Palagia, *Greek sculpture: function, materials, and techniques in the archaic and classical periods*. Cambridge University Press Cambridge, UK, 2006.
- [2] “Definition of software modeling by omg,” <https://www.omg.org/UML/what-is-uml.htm>, accessed: 2019-08-20.
- [3] B. Anda, K. Hansen, I. Gullesten, and H. K. Thorsen, “Experiences from Introducing UML-based Development in a Large Safety-critical Project,” *Empirical Softw. Engg.*, vol. 11, no. 4, pp. 555–581, Dec. 2006.
- [4] M. Grossman, J. E. Aronson, and R. V. McCarthy, “Does UML make the grade? Insights from the software development community,” *Information and Software Technology*, vol. 47, no. 6, pp. 383–397, 2005.
- [5] A. M. Fernández-Sáez, M. Genero, and M. R. V. Chaudron, “Empirical studies concerning the maintenance of UML diagrams and their use in the maintenance of code: A systematic mapping study,” *Information and Software Technology*, vol. 55, no. 7, pp. 1119–1142, 2013.
- [6] B. Dohing and J. Parsons, “How UML is Used,” *Commun. ACM*, vol. 49, no. 5, pp. 109–113, may 2006.
- [7] P. Baker, S. Loh, and F. Weil, “Model-Driven Engineering in a large industrial context—Motorola case study,” *Model Driven Engineering Languages and Systems*, pp. 476–491, 2005.
- [8] C. F. J. Lange, M. R. V. Chaudron, and J. Muskens, “In practice: UML software architecture and design description,” *IEEE Software*, vol. 23, no. 2, pp. 40–46, 2006.
- [9] W. J. Dzidek, E. Arisholm, and L. C. Briand, “A Realistic Empirical Evaluation of the Costs and Benefits of UML in Software Maintenance,” *IEEE Trans. Softw. Eng.*, vol. 34, no. 3, pp. 407–432, may 2008.
- [10] A. Nugroho and M. R. V. Chaudron, “A Survey into the Rigor of UML Use and Its Perceived Impact on Quality and Productivity,” in *Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, 2008, pp. 90–99.

- [11] G. Scanniello, C. Gravino, and G. Tortora, “Investigating the Role of UML in the Software Modeling and Maintenance-A Preliminary Industrial Survey.” in *ICEIS (3)*, 2010, pp. 141–148.
- [12] D. Budgen, A. J. Burn, O. P. Brereton, B. A. Kitchenham, and R. Pretorius, “Empirical evidence about the UML: A systematic literature review,” pp. 363–392, 2011.
- [13] M. Petre, “UML in practice,” in *Proceedings - International Conference on Software Engineering*, 2013, pp. 722–731.
- [14] J. Hutchinson, J. Whittle, M. Rouncefield, and S. Kristoffersen, “Empirical assessment of mde in industry,” in *Proceedings of the 33rd international conference on software engineering*. ACM, 2011, pp. 471–480.
- [15] M. Galster and D. Weyns, “Empirical Research in Software Architecture: How Far have We Come?” in *2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA)*. IEEE, 2016, pp. 11–20.
- [16] J. Lung, J. Aranda, S. M. Easterbrook, and G. V. Wilson, “On the Difficulty of Replicating Human Subjects Studies in Software Engineering,” in *Proceedings of the 30th International Conference on Software Engineering*, ser. ICSE '08. New York, NY, USA: ACM, 2008, pp. 191–200.
- [17] P. Lord, A. Macdonald, L. Lyon, and D. Giarretta, “From data deluge to data curation,” in *Proceedings of the UK e-science All Hands meeting*. Citeseer, 2004, pp. 371–375.
- [18] S. McConnell, *Code complete*. Pearson Education, 2004.
- [19] M. R. V. Chaudron, W. Heijstek, and A. Nugroho, “How effective is UML modeling?” *Software & Systems Modeling*, vol. 11, no. 4, pp. 571–580, oct 2012.
- [20] C. Kobryn, “UML 2001: A Standardization Odyssey,” *Commun. ACM*, vol. 42, no. 10, pp. 29–37, oct 1999.
- [21] G. Booch, *Object Oriented Design with Applications*. Redwood City, CA, USA: Benjamin-Cummings Publishing Co., Inc., 1991.
- [22] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen, *Object-oriented Modeling and Design*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1991.
- [23] I. Jacobson, *Object-oriented software engineering: a use case driven approach*. Pearson Education India, 1993.
- [24] H. Störrle, R. Hebig, and A. Knapp, “An Index for Software Engineering Models,” in *International Conference on Model Driven Engineering Languages and Systems (MoDELS) 2014*, 2014, pp. 36–40.
- [25] R. France, J. Bieman, and B. H. C. Cheng, “Repository for model driven development (ReMoDD),” in *International Conference on Model Driven Engineering Languages and Systems*. Springer, 2006, pp. 311–317.



- [26] B. Karasneh and M. R. V. Chaudron, "Online Img2UML Repository: An Online Repository for UML Models." in *EESSMOD@ MoDELS*, 2013, pp. 61–66.
- [27] J. Noten, J. G. M. Mengerink, and A. Serebrenik, "A data set of OCL expressions on GitHub," in *Proceedings of the 14th International Conference on Mining Software Repositories*. IEEE Press, 2017, pp. 531–534.
- [28] F. Bascianidi, J. Di Rocco, D. Di Ruscio, A. Di Salle, L. Iovino, and A. Pierantonio, "MDEForge: an Extensible Web-Based Modeling Platform." 2014.
- [29] M. Dirix, A. Muller, and V. Aranega, "Genmymodel: an online UML case tool," in *ECOOP*, 2013.
- [30] M. Kunze, P. Berger, and M. Weske, "BPM Academic Initiative-Fostering Empirical Research." 2012.
- [31] M. Weske, *Business Process Management: Concepts, Languages, Architectures*, 2nd ed. Springer-Verlag Berlin Heidelberg, 2012.
- [32] M. Torchiano, F. Tomassetti, F. Ricca, A. Tiso, and G. Reggio, "Relevance, benefits, and problems of software modelling and model driven techniques - A survey in the Italian industry," *Journal of Systems and Software*, vol. 86, no. 8, pp. 2110–2126, 2013.
- [33] A. Forward, O. Badreddin, and T. C. Lethbridge, "Perceptions of software modeling: a survey of software practitioners," in *5th workshop from code centric to model centric: evaluating the effectiveness of MDD (C2M: EEMDD)*, 2010.
- [34] T. Gorschek, E. Tempero, and L. Angelis, "On the use of software design models in software development practice: An empirical investigation," *Journal of Systems and Software*, vol. 95, pp. 176–193, 2014.
- [35] A. Nugroho and M. R. V. Chaudron, "Evaluating the Impact of UML Modeling on Software Quality : An Industrial Case Study," *Springer-Verlag*, pp. 181–195, 2009.
- [36] A. Kuhn, G. C. Murphy, and C. A. Thompson, *An Exploratory Study of Forces and Frictions Affecting Large-Scale Model-Driven Development*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 352–367.
- [37] O. Badreddin, T. C. Lethbridge, and M. Elassar, "Modeling Practices in Open Source Software," *Open Source Software: Quality Verification - 9th IFIP WG 2.13 International Conference, OSS 2013*, pp. 127–139, 2013.
- [38] W. Ding, P. Liang, A. Tang, H. Van Vliet, and M. Shahin, "How do open source communities document software architecture: An exploratory survey," in *Proceedings of the IEEE International Conference on Engineering of Complex Computer Systems, ICECCS*, 2014, pp. 136–145.

- [39] K. Yatani, E. Chung, C. Jensen, and K. N. Truong, "Understanding how and why open source contributors use diagrams in the development of Ubuntu," *Proceedings of the 27th international conference on Human factors in computing systems - CHI 09*, p. 995, 2009.
- [40] H. Osman and M. R. V. Chaudron, "UML Usage in Open Source Software Development : A Field Study," in *Proceedings of the 3rd International Workshop on Experiences and Empirical Studies in Software Modeling co-located with 16th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2013)*, 2013, pp. 23–32.
- [41] R. Kazman, D. Goldenson, I. Monarch, W. Nichols, and G. Valetto, "Evaluating the Effects of Architectural Documentation: A Case Study of a Large Scale Open Source Project," *IEEE Transactions on Software Engineering*, vol. 42, no. 3, pp. 220–260, mar 2016.
- [42] E. Chung, C. Jensen, K. Yatani, V. Kuechler, and K. N. Truong, "Sketching and Drawing in the Design of Open Source Software," in *Proc. VL/HCC*, 2010, pp. 195–202.
- [43] P. Langer, T. Mayerhofer, M. Wimmer, and G. Kappel, "On the usage of UML: Initial results of analyzing open UML models," *Modellierung 2014*, vol. P225, pp. 289–304, 2014.
- [44] G. D. Crnkovic, *Constructive Research and Info-computational Knowledge Generation*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 359–380.
- [45] G. Gousios and D. Spinellis, "GHTorrent: Github's data from a firehose," in *Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on*. IEEE, 2012, pp. 12–21.
- [46] G. Robles, S. Koch, J. M. González-Barahona, and J. Carlos, "Remote analysis and measurement of libre software systems by means of the CVSAnalY tool," in *Proceedings of the 2nd ICSE Workshop on Remote Analysis and Measurement of Software Systems (RAMSS)*. IET, 2004, pp. 51–56.
- [47] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering*. Springer Science & Business Media, 2012.
- [48] S. Easterbrook, J. Singer, M.-A. Storey, and D. Damian, *Selecting Empirical Methods for Software Engineering Research*. London: Springer London, 2008, pp. 285–311.
- [49] R. K. Yin, *Case study research and applications: Design and methods*, 5th ed. Sage publications, 2013.
- [50] B. Flyvbjerg, "Five misunderstandings about case-study research," *Qualitative inquiry*, vol. 12, no. 2, pp. 219–245, 2006.
- [51] M. Kuniavsky, *Observing the user experience: a practitioner's guide to user research*. Elsevier, 2003.

- [52] J. Han, J. Pei, and M. Kamber, *Data mining: concepts and techniques (3rd edition)*. Elsevier, 2011.
- [53] E. J. Chikofsky and J. H. Cross, “Reverse engineering and design recovery: A taxonomy,” *IEEE software*, vol. 7, no. 1, 1990.
- [54] B. Karasneh and M. R. V. Chaudron, “Extracting UML models from images,” in *2013 5th International Conference on Computer Science and Information Technology*, mar 2013, pp. 169–178.
- [55] F. Ricca, M. Di Penta, M. Torchiano, P. Tonella, and M. Ceccato, “How Developers’ Experience and Ability Influence Web Application Comprehension Tasks Supported by UML Stereotypes: A Series of Four Experiments,” *IEEE Trans. Softw. Eng.*, vol. 36, no. 1, pp. 96–118, jan 2010.
- [56] M. Genero, J. A. Cruz-Lemus, D. Caivano, S. Abrahão, E. Insfran, and J. A. Carsi, “Does the Use of Stereotypes Improve the Comprehension of UML Sequence Diagrams?” in *2nd Symposium on Empirical Software Engineering and Measurement*, ser. ESEM ’08, 2008, pp. 300–302.
- [57] O. Andriyevska, N. Dragan, B. Simoes, and J. I. Maletic, “Evaluating UML Class Diagram Layout based on Architectural Importance,” in *3rd IEEE International Workshop on Visualizing Software for Understanding and Analysis*, 2005, pp. 1–6.
- [58] S. Yusuf, H. Kagdi, and J. I. Maletic, “Assessing the comprehension of UML class diagrams via eye tracking,” in *15th Int. Conf. on Program Comprehension. ICPC’07*. IEEE, 2007, pp. 113–122.
- [59] B. Sharif and J. I. Maletic, “The effect of layout on the comprehension of UML class diagrams: A controlled experiment,” in *5th IEEE Int. WS. on Visualizing Software for Understanding and Analysis (VISSOFT 2009)*. IEEE, 2009, pp. 11–18.
- [60] V. Arora, R. Bhatia, and M. Singh, “Synthesizing test scenarios in UML activity diagram using a bio-inspired approach,” *Computer Languages, Systems & Structures*, vol. 50, pp. 1–19, 2017.
- [61] Y. El Ahmar, X. Le Pallec, S. Gérard, and T. Ho-Quang, “Visual Variables in UML: a First Empirical Assessment,” in *Human Factors in Modeling*, 2017.
- [62] Y. E. Ahmar, X. L. Pallec, and S. Gérard, “The visual variables in UML: how are they used by women?” in *Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings*. ACM, 2018, p. 17.
- [63] R. Kretschmer, D. E. Khelladi, A. Demuth, R. E. Lopez-Herrejon, and A. Egyed, “From abstract to concrete repairs of model inconsistencies: An automated approach,” in *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 2017, pp. 456–465.

- [64] C. D. Schulze, G. Hoops, and R. von Hanxleden, “Automatic Layout and Label Management for Compact UML Sequence Diagrams,” in *2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 2018, pp. 187–191.
- [65] J. Ott, A. Atchison, and E. J. Linstead, “Exploring the applicability of low-shot learning in mining software repositories,” *Journal of Big Data*, vol. 6, no. 1, p. 35, 2019.
- [66] A. de la Vega, P. Sánchez, and D. Kolovos, “Pinset: A DSL for Extracting Datasets from Models for Data Mining-Based Quality Analysis,” in *2018 11th International Conference on the Quality of Information and Communications Technology (QUATIC)*. IEEE, 2018, pp. 83–91.
- [67] Ö. Babur, L. Cleophas, and M. van den Brand, “Towards Distributed Model Analytics with Apache Spark.” in *MODELSWARD*, 2018, pp. 767–772.
- [68] H. Agt-Rickauer, R.-D. Kutsche, and H. Sack, “Automated recommendation of related model elements for domain models,” in *International Conference on Model-Driven Engineering and Software Development*. Springer, 2018, pp. 134–158.
- [69] D. C. Torre, “Definition and validation of consistency rules between uml diagrams,” Ph.D. dissertation, Carleton University, 2018.
- [70] O. Baddreddin and K. Rahad, “The impact of design and uml modeling on codebase quality and sustainability,” in *Proceedings of the 28th Annual International Conference on Computer Science and Software Engineering*. IBM Corp., 2018, pp. 236–244.
- [71] M. Petre and A. van der Hoek, *Software Design Decoded: 66 Ways Experts Think*. The MIT Press, 2016.
- [72] M. Brambilla, J. Cabot, and M. Wimmer, “Model-driven software engineering in practice,” *Synthesis Lectures on Software Engineering*, vol. 3, no. 1, pp. 1–207, 2017.
- [73] H. Ossher, R. Bellamy, I. Simmonds, D. Amid, A. Anaby-Tavor, M. Callery, M. Desmond, J. de Vries, A. Fisher, and S. Krasikov, “Flexible modeling tools for pre-requirements analysis: conceptual architecture and research challenges,” in *ACM Sigplan Notices*, vol. 45, no. 10. ACM, 2010, pp. 848–864.
- [74] H. Störrle, “On the impact of layout quality to understanding UML diagrams,” in *2011 IEEE Symposium on Visual Languages and Human-Centric Computing*, 2011, pp. 135–142.
- [75] W. Fu, T. Menzies, and X. Shen, “Tuning for software analytics: Is it really necessary?” *Information and Software Technology*, vol. 76, pp. 135–146, 2016.

- [76] H. Zhang, S. K. Moon, and T. H. Ngo, “Hybrid Machine Learning Method to Determine the Optimal Operating Process Window in Aerosol Jet 3D Printing,” *ACS Applied Materials & Interfaces*, vol. 11, no. 19, pp. 17994–18003, 2019.
- [77] R. Raina, Y. Shen, A. McCallum, and A. Y. Ng, “Classification with hybrid generative/discriminative models,” in *Advances in neural information processing systems*, 2004, pp. 545–552.
- [78] C. F. J. Lange and M. R. V. Chaudron, “Managing model quality in UML-based software development,” in *Proceedings - 13th IEEE International Workshop on Software Technology and Engineering Practice, STEP 2005*, ser. STEP '05, vol. 2005. Washington, DC, USA: IEEE Computer Society, 2005, pp. 7–16.
- [79] ISO 25010:2011, “Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models,” International Organization for Standardization, Geneva, CH, Standard ISO 25010:2011, mar 2011.
- [80] J. Bansiya and C. G. Davis, “A hierarchical model for object-oriented design quality assessment,” *IEEE Transactions on software engineering*, vol. 28, no. 1, pp. 4–17, 2002.
- [81] J. A. McCall, P. K. Richards, and G. F. Walters, “Concepts and definitions of software quality,” *Factors in Software Quality, NTIS*, vol. 1, 1977.
- [82] B. W. Boehm, J. R. Brown, and M. Lipow, “Quantitative evaluation of software quality,” in *Proceedings of the 2nd international conference on Software engineering*. IEEE Computer Society Press, 1976, pp. 592–605.
- [83] B. Hussein, “Automated quality-assessment for UML models in open source projects,” Master’s thesis, University of Gothenburg, 2019.
- [84] A. M. Fernández-Sáez, M. R. V. Chaudron, and M. Genero, “An industrial case study on the use of UML in software maintenance and its perceived benefits and hurdles,” *Empirical Software Engineering*, pp. 1–65, 2018.
- [85] P. Pietsch, D. Reuling, U. Kelter, J. Folmer, and B. Vogel-Heuser, “Experiences on the Quality and Availability of Test Models for Model Differencing Tools,” in *FMI 2014-Free Models Initiative Workshop Proceedings*, 2014, p. 11.
- [86] J. Kramer, “Is abstraction the key to computing?” *Communications of the ACM*, vol. 50, no. 4, pp. 36–42, 2007.
- [87] D. R. Stikkolorum, C. Stevenson, and M. R. V. Chaudron, “Assessing Software Design Skills and their Relation with Reasoning Skills.” in *EduSymp@ MoDELS*, 2013, pp. 1–8.
- [88] F. Leung and N. Bolloju, “Analyzing the quality of domain models developed by novice systems analysts,” in *System Sciences, 2005. HICSS'05. Proceedings of the 38th Annual Hawaii International Conference on*. IEEE, 2005, pp. 188b—188b.

- [89] B. Karasneh, R. Jolak, and M. R. V. Chaudron, "Using Examples for Teaching Software Design: An Experiment Using a Repository of UML Class Diagrams," in *Software Engineering Conference (APSEC), 2015 Asia-Pacific*. IEEE, 2015, pp. 261–268.
- [90] K. C. Thramboulidis, "Using UML in control and automation: a model driven approach," in *Industrial Informatics, 2004. INDIN '04. 2004 2nd IEEE International Conference on*, jun 2004, pp. 587–593.
- [91] C. Secchi, C. Fantuzzi, and M. Bonfe, "On the Use of UML for Modeling Physical Systems," in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, apr 2005, pp. 3990–3995.
- [92] R. P. L. Buse and T. Zimmermann, "Information Needs for Software Development Analytics," in *Proceedings of the 34th International Conference on Software Engineering*, ser. ICSE '12. Piscataway, NJ, USA: IEEE Press, 2012, pp. 987–996.
- [93] "Enterprise Architect," <http://www.sparxsystems.com/>.
- [94] "Visual Paradigm," <http://www.visual-paradigm.com/>.
- [95] B. Karasneh and M. R. V. Chaudron, "Img2UML: A System for Extracting UML Models from Images," in *2013 39th Euromicro Conference on Software Engineering and Advanced Applications*, sep 2013, pp. 134–137.
- [96] E. P. Costa, A. C. Lorena, A. Carvalho, and A. A. Freitas, "A review of performance evaluation measures for hierarchical classifiers." in *Evaluation Methods for Machine Learning II: papers from the AAAI-2007 Workshop, AAAI Technical Report WS-07-05*, C. Drummond, W. Elazmeh, N. Japkowicz, and S. A. Macskassy, Eds. AAAI Press, jul 2007, pp. 182–196.
- [97] D. Blostein, E. Lank, and R. Zanibbi, *Treatment of Diagrams in Document Image Analysis*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 330–344.
- [98] D. Lu and Q. Weng, "A Survey of Image Classification Methods and Techniques for Improving Classification Performance," *Int. J. Remote Sens.*, vol. 28, no. 5, pp. 823–870, jan 2007.
- [99] J. A. Shine and D. B. Carr, "A comparison of classification methods for large imagery data sets," *JSM*, pp. 3205–3207, 2002.
- [100] A. Mishchenko and N. Vassilieva, "Model-based chart image classification," in *International Symposium on Visual Computing*. Springer, 2011, pp. 476–485.
- [101] B. T. Messmer and H. Bunke, "Automatic learning and recognition of graphical symbols in engineering drawings," in *International Workshop on Graphics Recognition*. Springer, 1995, pp. 123–134.

- [102] E. Lank, J. S. Thorley, and S. J.-S. Chen, “An interactive system for recognizing hand drawn UML diagrams,” in *Proceedings of the 2000 conference of the Centre for Advanced Studies on Collaborative research*. IBM Press, 2000, p. 7.
- [103] T. Hammond and R. Davis, “Tahuti: A geometrical sketch recognition system for uml class diagrams,” in *ACM SIGGRAPH 2006 Courses*. ACM, 2006, p. 25.
- [104] E. Lank, J. Thorley, S. Chen, and D. Blostein, “On-line recognition of UML diagrams,” in *Document Analysis and Recognition, 2001. Proceedings. Sixth International Conference on*. IEEE, 2001, pp. 356–360.
- [105] L. Fu and L. B. Kara, “From engineering diagrams to engineering models: Visual recognition and applications,” *Computer-Aided Design*, vol. 43, no. 3, pp. 278–292, 2011.
- [106] R. O. Duda and P. E. Hart, “Use of the Hough transformation to detect lines and curves in pictures,” *Communications of the ACM*, vol. 15, no. 1, pp. 11–15, 1972.
- [107] S. Suzuki and Others, “Topological structural analysis of digitized binary images by border following,” *Computer vision, graphics, and image processing*, vol. 30, no. 1, pp. 32–46, 1985.
- [108] J. C. Russ, *The Image Processing Handbook (3rd Ed.)*. Boca Raton, FL, USA: CRC Press, Inc., 1999.
- [109] M. A. Hall, “Correlation-based feature selection for machine learning,” Hamilton, Tech. Rep., 1999.
- [110] “Waikato Environment for Knowledge Analysis (WEKA),” <http://www.cs.waikato.ac.nz/ml/weka/>.
- [111] I. H. Witten, E. Frank, and M. A. Hall, *Data Mining: Practical Machine Learning Tools and Techniques*, 3rd ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011.
- [112] E. H. Adelson, C. H. Anderson, J. R. Bergen, P. J. Burt, and J. M. Ogden, “Pyramid methods in image processing,” *RCA engineer*, vol. 29, no. 6, pp. 33–41, 1984.
- [113] A. Herout, M. Dubská, and J. Havel, “Review of hough transform for line detection,” in *Real-Time Detection of Lines and Grids*. Springer, 2013, pp. 3–16.
- [114] D. Lagunovsky and S. Ablameyko, “Fast line and rectangle detection by clustering and grouping,” in *International Conference on Computer Analysis of Images and Patterns*. Springer, 1997, pp. 503–510.
- [115] K. Murakami and T. Naruse, “High speed line detection by Hough transform in local area,” in *Pattern Recognition, 2000. Proceedings. 15th International Conference on*, vol. 3. IEEE, 2000, pp. 467–470.

- [116] G. Booch, J. Rumbaugh, and I. Jacobson, *Unified Modeling Language User Guide, The (2Nd Edition) (Addison-Wesley Object Technology Series)*. Addison-Wesley Professional, 2005.
- [117] T. Ho-Quang, M. R. V. Chaudron, I. Samúelsson, J. Hjaltason, B. Karasneh, and H. Osman, “Automatic classification of uml class diagrams from images,” in *2014 21st Asia-Pacific Software Engineering Conference*, vol. 1. IEEE, 2014, pp. 399–406.
- [118] G. Reggio, M. Leotta, and F. Ricca, “Who Knows/Uses What of the UML: A Personal Opinion Survey,” in *Model-Driven Engineering Languages and Systems*. Springer, 2014, pp. 149–165.
- [119] G. Robles, J. M. Gonzalez-Barahona, and J. J. Merelo, “Beyond source code: the importance of other artifacts in software development (a case study),” *Journal of Systems and Software*, vol. 79, no. 9, pp. 1233–1248, 2006.
- [120] B. Vasilescu, A. Serebrenik, M. Goeminne, and T. Mens, “On the variation and specialisation of workload - a case study of the Gnome ecosystem community,” *Empirical Software Engineering*, vol. 19, no. 4, pp. 955–1008, 2014.
- [121] S. McIntosh, B. Adams, and A. E. Hassan, “The evolution of ant build systems,” in *Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on*. IEEE, 2010, pp. 42–51.
- [122] S. McIntosh, B. Adams, T. H. D. Nguyen, Y. Kamei, and A. E. Hassan, “An empirical study of build maintenance effort,” in *Proceedings of the 33rd international conference on software engineering*. ACM, 2011, pp. 141–150.
- [123] Y. Jiang and B. Adams, “Co-evolution of Infrastructure and Source Code - An Empirical Study,” in *12th {IEEE/ACM} Working Conference on Mining Software Repositories, {MSR} 2015, Florence, Italy, May 16-17, 2015*, 2015, pp. 45–55.
- [124] G. Robles, J. M. González-Barahona, D. Izquierdo-Cortazar, and I. Her-raiz, “Tools for the study of the usual data sources found in libre software projects,” *International Journal of Open Source Software and Processes*, vol. 1, no. 1, pp. 24–45, 2009.
- [125] R. Hebig, T. Ho~Quang, G. Robles, and M. R. V. Chaudron, “List of identified projects with UML and replication package,” [\url{http://oss.models-db.com}](http://oss.models-db.com).
- [126] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wess-lén, *Experimentation in Software Engineering: An Introduction*. Norwell, MA, USA: Kluwer Academic Publishers, 2000.
- [127] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian, “The Promises and Perils of Mining GitHub,” in *Proceedings of the 11th Working Conference on Mining Software Repositories*, ser. MSR 2014. New York, NY, USA: ACM, 2014, pp. 92–101.



- [128] C. Gacek and B. Arief, “The many meanings of open source,” *IEEE software*, vol. 21, no. 1, pp. 34–40, 2004.
- [129] B. Fitzgerald, “The transformation of open source software,” *Mis Quarterly*, pp. 587–598, 2006.
- [130] D. M. German, “The GNOME project: a case study of open source, global software development,” *Software Process: Improvement and Practice*, vol. 8, no. 4, pp. 201–215, 2003.
- [131] D. Riehle, “The economic case for open source foundations,” *Computer*, vol. 43, no. 1, pp. 86–90, 2010.
- [132] Ø. Hauge, C. Ayala, and R. Conradi, “Adoption of open source software in software-intensive organizations—A systematic literature review,” *Information and Software Technology*, vol. 52, no. 11, pp. 1133–1154, 2010.
- [133] K. Crowston, K. Wei, J. Howison, and A. Wiggins, “Free/Libre open-source software development: What we know and what we do not know,” *ACM Computing Surveys (CSUR)*, vol. 44, no. 2, p. 7, 2012.
- [134] K.-J. Stol, M. A. Babar, P. Avgeriou, and B. Fitzgerald, “A comparative study of challenges in integrating Open Source Software and Inner Source Software,” *Information and Software Technology*, vol. 53, no. 12, pp. 1319–1336, 2011.
- [135] D. Spinellis and C. Szyperski, “How is open source affecting software development?” *IEEE Software*, vol. 21, no. 1, p. 28, 2004.
- [136] C. Hauff and G. Gousios, “Matching GitHub developer profiles to job advertisements,” in *Proceedings of the 12th Working Conference on Mining Software Repositories*. IEEE Press, 2015, pp. 362–366.
- [137] R. Hebig, T. H. Quang, M. R. V. Chaudron, G. Robles, and M. A. Fernandez, “The quest for open source projects that use uml: mining github,” in *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*. ACM, 2016, pp. 173–183.
- [138] I. S. Wiese, I. Steinmacher, C. Treude, J. T. D. Silva, and M. Gerosa, “Who is who in the mailing list? Comparing six disambiguation heuristics to identify multiple addresses of a participant,” in *Proceedings of the 32nd International Conference on Software Maintenance and Evolution*, 2016.
- [139] C. B. Seaman, “Qualitative methods in empirical studies of software engineering,” *IEEE Transactions on software engineering*, vol. 25, no. 4, pp. 557–572, 1999.
- [140] G. Marczyk, D. DeMatteo, and D. Festinger, *Essentials of research design and methodology*. John Wiley & Sons Inc, 2005.

- [141] M. H. B. Osman *et al.*, *Interactive scalable condensation of reverse engineered UML class diagrams for software comprehension*. Leiden Institute of Advanced Computer Science (LIACS), Faculty of Science . . . , 2015.
- [142] H. Osman and M. R. V. Chaudron, “An assessment of reverse engineering capabilities of UML CASE tools,” in *2nd Int. Conf. on Software Engineering Application*, 2011, pp. 7–12.
- [143] A. Nugroho, B. Flaton, and M. R. V. Chaudron, “Empirical Analysis of the Relation between Level of Detail in UML Models and Defect Density,” in *Conference, MoDELS 2008, Toulouse, France, 2008. Proceedings*, ser. LNCS, vol. 5301. Springer, 2008, pp. 600–614.
- [144] N. Maneerat and P. Muenchaisri, “Bad-smell prediction from software design model using machine learning techniques,” in *2011 Eighth International Joint Conference on Computer Science and Software Engineering (JCSSE)*, may 2011, pp. 331–336.
- [145] F. B. e Abreu, “The MOOD metrics set,” in *proc. ECOOP*, vol. 95, 1995, p. 267.
- [146] A. Halim, “Predict fault-prone classes using the complexity of UML class diagram,” in *2013 Int. Conf. on Computer, Control, Informatics and Its Applications (IC3INA)*, nov 2013, pp. 289–294.
- [147] E. Bagheri and D. Gasevic, “Assessing the maintainability of software product line feature models using structural metrics,” *Software Quality Journal*, vol. 19, no. 3, pp. 579–612, 2011.
- [148] M. H. Osman, M. R. V. Chaudron, and P. v. d. Putten, “An Analysis of Machine Learning Algorithms for Condensing Reverse Engineered Class Diagrams,” in *2013 IEEE International Conference on Software Maintenance*, sep 2013, pp. 140–149.
- [149] F. Thung, D. Lo, M. H. Osman, and M. R. V. Chaudron, “Condensing class diagrams by analyzing design and network metrics using optimistic classification,” in *Proc. of the 22nd Int. Conf. on Program Comprehension (ICPC)*. ACM, 2014, pp. 110–121.
- [150] X. Yang, D. Lo, X. Xia, and J. Sun, “Condensing class diagrams with minimal manual labeling cost,” in *COMPSAC, 2016, IEEE 40th*, vol. 1. IEEE, 2016, pp. 22–31.
- [151] Replication package of paper D. <http://models.cs.chalmers.se/oss/Downloads/SEAA2018/ReplicationPackage/>.
- [152] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, “The WEKA Data Mining Software: An Update,” *SIGKDD Explor. Newsl.*, vol. 11, no. 1, pp. 10–18, nov 2009.
- [153] R. C. Holte, “Very simple classification rules perform well on most commonly used datasets,” *Machine learning*, vol. 11, no. 1, 1993.

- [154] M. Junker, R. Hoch, and A. Dengel, "On the evaluation of document analysis components by recall, precision, and accuracy," in *Document Analysis and Recognition, 1999. ICDAR'99. Proceedings of the Fifth International Conference on*. IEEE, 1999, pp. 713–716.
- [155] M. Sokolova, N. Japkowicz, and S. Szpakowicz, "Beyond accuracy, F-score and ROC: a family of discriminant measures for performance evaluation," in *Australasian joint conference on artificial intelligence*. Springer, 2006, pp. 1015–1021.
- [156] D. Billsus and M. J. Pazzani, "Learning Collaborative Information Filters." in *Icml*, vol. 98, 1998, pp. 46–54.
- [157] J. A. Hanley and B. J. McNeil, "The meaning and use of the area under a receiver operating characteristic (ROC) curve." *Radiology*, vol. 143, no. 1, pp. 29–36, 1982.
- [158] A. Brown and G. Wilson, "The architecture of open source applications, volume i&ii," *Ebook*, May, 2012.
- [159] M. Soliman, A. R. Salama, M. Galster, O. Zimmermann, and M. Riebisch, "Improving the Search for Architecture Knowledge in Online Developer Communities," in *{IEEE} International Conference on Software Architecture, {ICSA} 2018, Seattle, WA, USA, April 30 - May 4, 2018*, 2018, pp. 186–195.
- [160] J. Musil, F. J. Ekaputra, M. Sabou, T. Ionescu, D. Schall, A. Musil, and S. Biffl, "Continuous Architectural Knowledge Integration: Making Heterogeneous Architectural Knowledge Available in Large-Scale Organizations," in *Software Architecture (ICSA), 2017 IEEE International Conference on*. IEEE, 2017, pp. 189–192.
- [161] Z.-Q. Lin, B. Xie, Y.-Z. Zou, J.-F. Zhao, X.-D. Li, J. Wei, H.-L. Sun, and G. Yin, "Intelligent development environment and software knowledge graph," *Journal of Computer Science and Technology*, vol. 32, no. 2, pp. 242–249, 2017.
- [162] A. Barker and J. Van Hemert, "Scientific workflow: a survey and research directions," in *International Conference on Parallel Processing and Applied Mathematics*. Springer, 2007, pp. 746–753.
- [163] G. H. Travassos, P. S. M. dos Santos, P. G. Mian, A. C. D. Neto, and J. Biolchini, "An environment to support large scale experimentation in software engineering," in *Engineering of Complex Computer Systems, 2008. ICECCS 2008. 13th IEEE International Conference on*. IEEE, 2008, pp. 193–202.
- [164] G. Robles, T. Ho-Quang, R. Hebig, M. R. V. Chaudron, and M. A. Fernandez, "An extensive dataset of UML models in GitHub," in *IEEE International Working Conference on Mining Software Repositories*, 2017, pp. 519–522.

- [165] T. Ho-Quang, R. Hebig, G. Robles, M. R. V. Chaudron, and M. A. Fernandez, “Practices and Perceptions of UML Use in Open Source Projects,” in *International Conference on Software Engineering (ICSE): Software Engineering in Practice Track*. IEEE, 2017, pp. 203–212.
- [166] M. H. Osman, T. Ho-Quang, and M. R. V. Chaudron, “An Automated Approach for Classifying Reverse-engineered and Forward-engineered UML Class Diagrams,” in *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 2018, pp. 396–399.
- [167] D. Rusk and Y. Coady, “Location-based analysis of developers and technologies on github,” in *2014 28th International Conference on Advanced Information Networking and Applications Workshops (WAINA)*. IEEE, 2014, pp. 681–685.
- [168] J. Krüger, M. Mukelabai, W. Gu, H. Shen, R. Hebig, and T. Berger, “Where is my Feature and What is it About? A Case Study on Recovering Feature Facets,” *Journal of Systems and Software*, 2019.
- [169] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao, “Scientific workflow management and the {K}epler system,” *Concurrency and Computation: Practice and Experience*, vol. 18, no. 10, pp. 1039–1065, 2006.
- [170] K. A. De Graaf, A. Tang, P. Liang, and H. Van Vliet, “Ontology-based software architecture documentation,” in *Software Architecture (WICSA) and European Conference on Software Architecture (ECSA), 2012 Joint Working IEEE/IFIP Conference on*. IEEE, 2012, pp. 121–130.
- [171] A. Tao and M. Roodbari, “Towards automatically generating explanations of a software systems,” Master’s thesis, Chalmers University of Technology, Gothenburg, Sweden, 2018.
- [172] J. Brooke and Others, “SUS-A quick and dirty usability scale,” *Usability evaluation in industry*, vol. 189, no. 194, pp. 4–7, 1996.
- [173] C. Larman, *Applying {UML} and patterns: an introduction to object oriented analysis and design and interactive development*. Pearson, 2012.
- [174] A. Nugroho and M. R. V. Chaudron, “The impact of {UML} modeling on defect density and defect resolution time in a proprietary system,” *Empirical Software Engineering*, vol. 19, no. 4, pp. 926–954, 2014.
- [175] 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*, vol. 32, no. 6, pp. 365–381, 2006.
- [176] P. Devanbu, T. Zimmermann, and C. Bird, “Belief & evidence in empirical software engineering,” in *Software Engineering (ICSE), 2016 IEEE/ACM 38th International Conference on*. IEEE, 2016, pp. 108–119.

- [177] K. Herzig, S. Just, and A. Zeller, “It’s not a bug, it’s a feature: how misclassification impacts bug prediction,” in *International Conference on Software Engineering (ICSE)*. IEEE, 2013, pp. 392–401.
- [178] I. Rish, “An empirical study of the naive bayes classifier,” T. J. Watson IBM Research Center, Tech. Rep., 2001.
- [179] Y. Zhou, Y. Tong, R. Gu, and H. Gall, “Combining text mining and data mining for bug report classification,” *Journal of Software: Evolution and Process*, vol. 28, no. 3, pp. 150–176, 2016.
- [180] B. Vasilescu, A. Serebrenik, and V. Filkov, “A data set for social diversity studies of GitHub teams,” in *International Conference on Mining Software Repositories (MSR)*. IEEE, 2015, pp. 514–517.
- [181] N. Munaiah, S. Kroh, C. Cabrey, and M. Nagappan, “Curating GitHub for engineered software projects,” *Empirical Software Engineering*, vol. 22, no. 6, pp. 3219–3253, dec 2017.
- [182] P. D. Allison, *Multiple regression: A primer*. Pine Forge Press, 1999.
- [183] G. Rodríguez-Pérez, A. Zaidman, A. Serebrenik, G. Robles, and J. M. González-Barahona, “What if a Bug Has a Different Origin?: Making Sense of Bugs Without an Explicit Bug Introducing Change,” in *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM ’18. New York, NY, USA: ACM, 2018, pp. 52:1—52:4.
- [184] B. Vasilescu, Y. Yu, H. Wang, P. Devanbu, and V. Filkov, “Quality and productivity outcomes relating to continuous integration in GitHub, year = 2015,” in *Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*. ACM, pp. 805–816.
- [185] H. Zhang, “The optimality of naive Bayes,” *AA*, vol. 1, no. 2, p. 3, 2004.
- [186] J. Cabot, J. L. C. Izquierdo, V. Cosentino, and B. Rolandi, “Exploring the use of labels to categorize issues in open-source software projects,” in *International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2015, pp. 550–554.
- [187] N. Medvidovic, A. Egyed, and D. S. Rosenblum, “Round-Trip Software Engineering Using UML: From Architecture to Design and back,” 1999.
- [188] A. M. Fernández-Sáez, M. Genero, M. R. V. Chaudron, D. Caivano, and I. Ramos, “Are Forward Designed or Reverse-Engineered UML diagrams more helpful for code maintenance?: A family of experiments,” *Information & Software Technology*, vol. 57, pp. 644–663, 2015.
- [189] R. Wirfs-Brock, “Characterizing Classes,” *{IEEE} Software*, vol. 23, no. 2, pp. 9–11, 2006.
- [190] N. Dragan, M. L. Collard, and J. I. Maletic, “Automatic identification of class stereotypes,” in *26th {IEEE} International Conference on Software Maintenance {(ICSM} 2010), September 12-18, 2010, Timisoara, Romania*, 2010, pp. 1–10.

- [191] L. Moreno and A. Marcus, “JStereoCode: automatically identifying method and class stereotypes in Java code,” in *{IEEE/ACM} Int. Conf. on Automated Software Engineering, ASE’12, Essen, Germany, September 3-7, 2012*, 2012, pp. 358–361.
- [192] N. Alhindawi, N. Dragan, M. L. Collard, and J. I. Maletic, “Improving feature location by enhancing source code with stereotypes,” in *29th Int Conf on Software Maintenance (ICSM), 2013*. IEEE, 2013, pp. 300–309.
- [193] R. Wirfs-Brock and A. McKean, *Object design: roles, responsibilities, and collaborations*. Addison-Wesley Professional, 2003.
- [194] N. Dragan, M. L. Collard, and J. I. Maletic, “Reverse Engineering Method Stereotypes,” in *22nd {IEEE} ICSM 2006, 24-27 September 2006, Philadelphia, Pennsylvania, {USA}*, 2006, pp. 24–34.
- [195] M. Lanza and R. Marinescu, *Object-oriented metrics in practice: using software metrics to characterize, evaluate, and improve the design of object-oriented systems*. Springer Science & Business Media, 2007.
- [196] L. Moreno, J. Aponte, G. Sridhara, A. Marcus, L. L. Pollock, and K. Vijay-Shanker, “Automatic generation of natural language summaries for Java classes,” in *{IEEE} 21st ICPC (2013) San Francisco*, 2013, pp. 23–32.
- [197] A. Budi, Lucia, D. Lo, L. Jiang, and S. Wang, “Automated Detection of Likely Design Flaws in N-Tier Architectures,” in *Proceedings of the 23rd International Conference on Software Engineering & Knowledge Engineering (SEKE’2011), USA, July 7-9, 2011*, 2011, pp. 613–618.
- [198] E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.
- [199] M. Zanoni, F. A. Fontana, and F. Stella, “On applying machine learning techniques for design pattern detection,” *Journal of Systems and Software*, vol. 103, pp. 102–117, 2015.
- [200] M. L. Collard, “Addressing source code using srcml,” in *IEEE International Workshop on Program Comprehension Working Session: Textual Views of Source Code to Support Comprehension (IWPC’05)*, 2005.
- [201] “Replication package of paper G,” [http://models.cs.chalmers.se/oss/Downloads/JSS2019\\_SCAM\\_ReplicationPackage/](http://models.cs.chalmers.se/oss/Downloads/JSS2019_SCAM_ReplicationPackage/).
- [202] K. W. Bowyer, N. V. Chawla, L. O. Hall, and W. P. Kegelmeyer, “SMOTE: Synthetic minority over-sampling technique,” *CoRR*.
- [203] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and Regression Trees*. Monterey, CA: Wadsworth and Brooks, 1984.
- [204] A. Nurwidyantoro, T. Ho-Quang, and M. R. V. Chaudron, “Automated classification of class role-stereotypes via machine learning,” in *Proceedings of the Evaluation and Assessment on Software Engineering*. ACM, 2019, pp. 79–88.

- [205] B. W. Matthews, “Comparison of the predicted and observed secondary structure of T4 phage lysozyme,” *Biochimica et Biophysica Acta (BBA)-Protein Structure*, vol. 405, no. 2, pp. 442–451, 1975.
- [206] S. Boughorbel, F. Jarray, and M. El-Anbari, “Optimal classifier for imbalanced data using Matthews Correlation Coefficient metric,” *PloS one*, vol. 12, no. 6, p. e0177678, 2017.
- [207] K. A. M. Ferreira, M. A. S. Bigonha, R. S. Bigonha, L. F. O. Mendes, and H. C. Almeida, “Identifying thresholds for object-oriented software metrics,” *Journal of Systems and Software*, vol. 85, no. 2, pp. 244–257, 2012.
- [208] M. Aniche, C. Treude, A. Zaidman, A. Van Deursen, and M. A. Gerosa, “SATT: Tailoring code metric thresholds for different software architectures,” in *2016 IEEE 16th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, 2016, pp. 41–50.
- [209] F. Zhang, A. Mockus, Y. Zou, F. Khomh, and A. E. Hassan, “How does context affect the distribution of software maintainability metrics?” in *2013 IEEE International Conference on Software Maintenance*. IEEE, 2013, pp. 350–359.
- [210] H. Bagheri, J. Garcia, A. Sadeghi, S. Malek, and N. Medvidovic, “Software architectural principles in contemporary mobile software: from conception to practice,” *Journal of Systems and Software*, vol. 119, pp. 31–44, 2016.
- [211] M. Shahin, P. Liang, and M. A. Babar, “A systematic review of software architecture visualization techniques,” *Journal of Systems and Software*, vol. 94, pp. 161–185, 2014.
- [212] A. Blouin, N. Moha, B. Baudry, H. Sahraoui, and J.-M. Jézéquel, “Assessing the Use of Slicing-based Visualizing Techniques on the Understanding of Large Metamodels,” *Inf. Softw. Technol.*, vol. 62, no. C, pp. 124–142, jun 2015.
- [213] I. Jacobson, G. Booch, and J. Rumbaugh, *The Unified Software Development Process*. Boston, MA, USA: Addison-Wesley, 1999.
- [214] M. Staron, L. Kuzniarz, and C. Wohlin, “Empirical assessment of using stereotypes to improve comprehension of UML models: A set of experiments,” *Journal of Systems and Software*, vol. 79, no. 5, pp. 727–742, 2006.
- [215] M. Lanza and S. Ducasse, “Polymetric Views—A Lightweight Visual Approach to Reverse Engineering,” *Transactions on Software Engineering (TSE)*, vol. 29, no. 9, pp. 782–795, sep 2003.
- [216] S. Demeyer, S. Ducasse, and O. Nierstrasz, *Object-Oriented Reengineering Patterns*. Morgan Kaufmann, 2002.
- [217] “Replication package of paper H,” [http://models.cs.chalmers.se/oss/Downloads/MODELS2019\\_ReplicationPackage/](http://models.cs.chalmers.se/oss/Downloads/MODELS2019_ReplicationPackage/), accessed: 2019-03-09.

- [218] S. G. Hart and L. E. Staveland, "Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research," in *Advances in psychology*. Elsevier, 1988, vol. 52, pp. 139–183.
- [219] E. A. Bustamante and R. D. Spain, "Measurement invariance of the Nasa TLX," in *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 52, no. 19. SAGE Publications Sage CA: Los Angeles, CA, 2008, pp. 1522–1526.
- [220] NASA.
- [221] T. S. Tullis and J. N. Stetson, "A comparison of questionnaires for assessing website usability," in *Usability professional association conference*, vol. 1. Minneapolis, USA, 2004.
- [222] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics bulletin*, vol. 1, no. 6, pp. 80–83, 1945.
- [223] S. Anselm and J. Corbin, *Basics of qualitative research: techniques and procedures for developing grounded theory*. SAGE Publications, Thousand Oaks, USA, 1998.
- [224] R. A. Grier, "How high is high? A meta-analysis of NASA-TLX global workload scores," in *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 59, no. 1. SAGE Publications Sage CA: Los Angeles, CA, 2015, pp. 1727–1731.
- [225] J. Sauro, *A practical guide to the system usability scale: Background, benchmarks & best practices*. Measuring Usability LLC Denver, CO, 2011.