Master's thesis

On the need and value of trace link recovery in Model-Driven development

A thesis submitted for the degree Master of Science in Business Informatics

Author
Wouter van Oosten
5445205

Supervisors
Dr. Fabiano Dalpiaz
Dr. Sergio España Cubillo

February 24, 2023

Abstract

Managing requirements and their changes are important, as many examples exists of failed projects. Following the lifecycle of a requirement is known as requirements traceability. Traceability has been studied extensively, often with the focus to transform from manual trace practices to more (semi-) automated alternatives and their current status of use. Still, there is a gap between the tools that are proposed and their use in the industry.

Traceability for Model-Driven Development (MDD) has been studied less, though recent studies showed promising results by using ML to recover trace links between user stories and commits to software in MDD software projects. Despite the studies to (semi-)automated traceability, industry practitioners generally make use manual processes. Therefore, it is necessary to study the needs of traceability in Model-Driven software development and improve upon the ML model to better address those needs. To investigate this, we formulated our research question as; *What is the effectiveness of (semi-)automated requirements traceability in an MDD environment?*

We approach our research question from four perspectives. We start with a systematic mapping study, which provides us with the knowledge available in literature. Next, we perform a secondary analysis which looks into similarities and differences between software projects produced through MDD and non-MDD. Third, we experiment with techniques to improve upon our ML model regarding two trace scenarios. Finally, we conduct a focus group with experts to identify issues that the industry experiences in their traceability practices and discover opportunities.

Many opinions exist on tracing software artefacts, ranging from "a made up problem" to "worth its weight in gold". Studies reported that, often, traceability practices remained a static process that requires manual input. This view is confirmed by our focus group experts, though they find their practice to be low effort. To improve upon our ML model, we experimented with automated feature selection to reduce the feature space (n=131) of the original model. We tested four selection algorithms on the two trace scenarios and found that we, largely, were able to match or surpass (up to 16%) our baseline's performance while reducing our feature space ranging between 15 and 60 features. To better address the needs of the industry, we learned that traceability should not be limited to only tracing commits to Jira issues. Extending traceability practices to artefacts, such as testing, leads to more insight and a more transparent process. However, tracing those artefacts manually is not scalable, which suggest that transforming their process towards a semi-automated one, might be a necessary step. When discussing our two trace scenarios, the experts were hesitant on a system that establishes traceability without a developers interference. Actively participating in that process makes you are more aware of established links, implying a semi-automated approach would be more meeting a developers' needs.

Through our study, we filled the gap in literature by a combined study on both modern industrial practices and state-of-the-art information technology solutions. We gathered insights in where low-code platforms could improve upon to offer an even more complete solution for software development through MDD.

Preface

To finalise my two and a half year journey into the fields of business and information technology I dedicated myself to researching “the need and value of trace link recovery in Model-Driven development”. In front of you is my thesis, which is the result of the last nine months of my journey to fulfill the graduation requirements for the masters degree of science in Business Informatics.

It was the beginning of May last year when I was orienting myself to a candidate topic with which I wanted to close my time at the University Utrecht. A few months prior, I completed the course Requirement Engineering (RE) under supervision of Dr. Fabiano Dalpiaz from the Department of Information and Computing Sciences at Utrecht University, the Netherlands. I ended the course with great interest into the field of RE, resulting in me reaching out to Fabiano to discuss the possibilities for a thesis. We agreed upon a possible candidate for which I started to do some preliminary preparations, until Fabiano reached out to me to have an urgent meeting regarding another possibility that just popped up. I was asked to join an existing research group that was working on a extended paper on trace link recovery in Model-Driven development. I saw this as an unique opportunity to be working together with established academic professionals and have yet another experience in the scientific world before heading towards a future more focused on industrial appliances.

Therefore, I am grateful to my supervisor Dr. Fabiano Dalpiaz and would like to thank him for the opportunity he gave me and his excellent guidance during the past nine months. I always had the feeling that you gave me the freedom to design my own path and supported me whenever I needed it, while also challenged me to think outside the box. I also would like to thank my second supervisor, Dr. Sergio España Cubillo, assistant professor in Utrecht University, the Netherlands, who provided feedback on my research in various occasions. I am also especially thankful for sharing his expertise during his course Responsible ICT, in which I became more aware of the impact of ICT and the key role ICT can fulfil in taking responsibility for future generations. In addition, I would like to thank Toine Hurkmans from Mendix, who used his position within the company to find dedicated colleagues who were willing to share their expertise and help me to gather the information I was looking for. The experts, who will not be mentioned by name due to confidentially reasons, provided my with high value information that helped me to bring all ends together in my research.

I want to express my gratitude to my friends and family who were always interested in the process I was into and were there for me when needed it the most. For me it is important to highlight my grandfather particularly, who is not here anymore, but who always inspired and pushed me to go to the limits of my academic abilities. Finally, I am grateful to my girlfriend who was just as much part of my two and a half year journey as I and supported me from beginning to end.

Wouter van Oosten
Amersfoort, February 24, 2023

Contents

1 Introduction **1**
1.1 Problem statement . . . 2
1.2 Research questions . . . 2
1.3 Research objective . . . 3
1.4 Assumptions and expectations . . . 3
1.5 Research variables . . . 4

2 Research method **5**
2.1 Systematic mapping study . . . 6
2.1.1 Literature research protocol . . . 7
2.2 Secondary analysis . . . 9
2.2.1 Data sources . . . 10
2.2.2 Data quality assurance . . . 10
2.2.3 Analysis approach . . . 11
2.3 Experiment . . . 12
2.3.1 Scoping . . . 12
2.3.2 Planning . . . 13
2.3.3 Operation . . . 14
2.3.4 Analysis & interpretation . . . 15
2.3.5 Presentation & package . . . 15
2.4 Focus group . . . 15
2.4.1 Focus group discussion . . . 16

3 Literature review **17**
3.1 Traceability: a general perspective . . . 17
3.1.1 Defining traceability . . . 17
3.1.2 Traceability in practice . . . 18
3.2 Model-Driven development . . . 22
3.2.1 Introduction to MDD . . . 22
3.2.2 Traceability in MDD . . . 23
3.3 Baseline: Automated Traceability from Model Changes to Issues . . . 23
3.3.1 Case description . . . 23
3.3.2 Outcome . . . 24
3.4 Improving on ML classification . . . 25
3.4.1 Introduction to ML classification . . . 25
3.4.2 On the way to improvement . . . 27
3.4.3 A trace link recovery pipeline . . . 28
3.5 Lessons learned . . . 29

4 Secondary analysis **30**
4.1 Data description . . . 30
4.1.1 Data sources . . . 30
4.1.2 Project selection . . . 31
4.1.3 Data exploration . . . 31
4.2 Variables selection & visualisation . . . 32
4.2.1 Revision size . . . 32
4.2.2 Revision type . . . 34
4.2.3 Revision characteristics . . . 35
4.3 Lessons learned . . . 37

5 Experiment **38**
5.1 Data collection . . . 38

5.1.1 Treatment design 38
5.1.2 Experiment process 40
5.2 Models' performance 40
5.2.1 Selected features 41
5.2.2 mRMR and their values for K 41
5.2.3 Trace recommendation 43
5.2.4 Trace maintenance 46
5.3 Feature importance 48
5.3.1 Trace recommendation 48
5.3.2 Trace maintenance 50
5.4 Lessons learned 51

6 Focus group 52
6.1 Data collection 52
6.1.1 Focus group description 52
6.1.2 Data sources and preparations 52
6.2 Low-code and traceability 53
6.2.1 Processes and practices 53
6.2.2 View on value 55
6.2.3 Maturing the industry 55
6.3 Lessons learned 57

7 Discussion 58
7.1 Research questions and aim 58
7.2 Key findings 58
7.3 Triangulating our findings 60
7.3.1 About industrial needs 60
7.3.2 Improvements of the ML model 61
7.3.3 Improve trace recovery and link to the industry's needs 61
7.4 Implications and limitations 62
7.4.1 Implications 62
7.4.2 Limitations 62
7.4.3 Threats to validity 63
7.5 Future work 65
7.6 Conclusion 65
7.6.1 Sub-questions 65
7.6.2 Main question 66

Bibliography 72

Appendices 73
A Minimum attribute list 73
B Focus group discussion 74
C Summary of found studies in the literature review 76
D Test-retest study selection 77
E Literature quality assessment 78
F Data Extraction Form 80
G Source outlines for secondary analysis 82
H Distributions of revisions and JIRA issues 85
I Frequency on commit size histograms 87
J Types of change development 89
K Commits for (non-)MDD projects categorised 91
L Top 50 most frequent terms in commits 92
M Frequent term sets from commit messages 93
N Experimental method 94
O Baseline experiment 95
P Experiment aggregated results 96
Q Box and Whisker plots experiment 99
R Experiment feature importance results 103

S Empty Miro board structure 105
T Focus group transcriptions 106

List of Figures

1.1 Venn diagrams visualising the intersection between MDD and LCD. 4
1.2 Research variables 4
2.1 Empirical cycle 5
2.2 Venn diagram of research method triangulation 6
2.3 Analysis revision characteristics process visualisation 11
3.1 Example of a trace link between two artefacts 18
3.2 Traceability in multiple directions 19
3.3 Approaches to traceability visualisation. 21
3.4 Application domain of MDD 22
3.5 Example of a model as used in MDD 22
3.6 Process visualisation of supervised ML 25
3.7 Basic decision tree 26
3.8 Example of ROC graph showing three classifiers 27
3.9 ML pipeline architecture 29
4.1 Comparison of the distribution (revisions and Jira issues) between (non-)MDD projects 31
4.2 A comparison between MDD and non-MDD projects in revision sizes 33
4.3 Box and Whisker plots of revision size software projects 33
4.4 Visualisation of aggregated ratio's of commit sizes categories for all (non-)MDD projects. 34
4.5 A comparison between MDD and non-MDD projects in change types 35
5.1 FeatureWiz process visualisation 39
5.2 Box and Whisker plots of aggregated results for trace recommendation 44
5.3 Box and Whisker plots of aggregated results for trace recommendation 46
6.1 Responses on four statements on the experts view on traceability 55
B2 Informed consent form 75
F1 Data Extraction Form section I 80
F2 Data Extraction Form section II & III 81
G1 Structure of the Mendix data sets 82
G2 Database schemes of the SEOS33 data sets 84
H1 Data distribution visualisation of all Mendix software projects 85
H2 Data distribution visualisation of all SEOS33 software projects 86
I1 Histograms on the number of changes made per commit on MDD projects 87
I2 Histograms on the number of changes made per commit on non-MDD projects . . . 88
J1 Distribution of files being added, modified, or deleted to MDD software projects . . 89
J2 Distribution of files being added, modified, or deleted to non-MDD software projects 90
N Process-Deliverable Diagram of experiment operation 94
Q1 Box and Whisker plots for trace recommendation scenario 99
Q2 Box and Whisker plots for trace maintenance scenario 100
Q3.1 Box and Whisker plots for trace recommendation scenario using mRMR (FCQ) . . . 101
Q3.2 Box and Whisker plots for trace maintenance scenario using mRMR (FCQ) 101
Q4.1 Box and Whisker plots for trace recommendation scenario using mRMR (RFCQ) . . 102
Q4.2 Box and Whisker plots for trace maintenance scenario using mRMR (RFCQ) 102
S Empty structure of the Miro board used for the focus group 105

List of Tables

2.1 Overview of search queries. . . . 7
2.2 Overview of selection criteria 7
2.3 Data quality dimensions. . . . 10
4.1 Snippet most frequent terms present in commit messages for (non-)MDD projects . . 36
5.1 Summary of generated files during the experiment 40
5.3 Friedman test + Nemenyi Post-Hoc test for mRMR (RFCQ) 42
5.2 Friedman test + Nemenyi Post-Hoc test for mRMR (FCQ) 42
5.4 Macro-average results for trace recommendation 43
5.5 Summary of F_2-measures for trace recommendation for each project 44
5.6 Friedman test + Nemenyi post-hoc test for trace recommendation 45
5.7 Macro-average results for trace recommendation 46
5.8 Summary of $F_{0.5}$-measures for trace maintenance for each project 47
5.9 Friedman test + Nemenyi post-hoc test for trace maintenance 48
5.10 Aggregated feature importance results for trace recommendation 49
5.11 Feature importance top-5 for trace recommendation 49
5.12 Aggregated feature importance results for trace maintenance 50
5.13 Feature importance top-15 for trace maintenance 51
6.1 Overview of experts and their backgrounds that were involved in the focus group . . 52
6.2 Coding analysis and corresponding concepts of focus group 53
A Minimum attribute list 73
C Found studies in literature review 76
D Test-retest procedure results 77
E1 Description of literature quality assessment 78
E2 Literature quality assessment 79
G1 Mendix source summary 82
G2 SEOS33 source summary 83
K Commits for (non-)MDD projects categorised 91
L Top 50 frequent terms in commit messages for (non-)MDD projects 92
M1 Frequent term sets and their support from the seven MDD software projects 93
M2 Frequent term sets and their support from the seven non-MDD software projects . . 93
O1 Baseline performance results for trace recommendation and trace maintenance . . . 95
O2 Baseline feature importance results for trace recommendation and trace maintenance 95
P1 Aggregated results on performance (mRMR (FCQ)) 96
P2 Aggregated results on performance (mRMR (RFCQ)) 97
P3 Aggregated results on performance (RFECV) 98
P4 Aggregated results on performance (FeatureWiz) 98
R1 Aggregated feature importance results for mRMR (FCQ) 103
R2 Aggregated feature importance results for mRMR (RFCQ) 103
R3 Aggregated feature importance results for the RFECV feature selection algorithm . 104
R4 Aggregated feature importance results for the FeatureWiz feature selection algorithm 104

Acronyms

DEF	Data extraction form.
DL	Deep Learning.
EMA	European Medicines Agency.
FP	False positives.
ITS	Issue Tracking System.
LCD	Low-Code Development.
MDD	Model-Driven development.
MDE	Model-Driven Engineering.
ML	Machine Learning.
MQ	Main Research Question.
mRMR	Minimum Redundancy - Maximum Relevance.
PDD	Process-Deliverable Diagram.
RFE	Recursive Feature Elimination.
ROC	Receiver Operating Graphs.
RT	Requirements Traceability.
SDM	Software Development Manager.
SQ	Sub Question.
ST	Software Traceability.
TLR	Trace Link Recovery.
TP	True positives.
TPOT	Tree-based Pipeline Optimization Tool.
VCS	Version Control System.

Chapter 1

Introduction

With the increasing complexity of software systems, it is becoming more important for software development teams to manage the development of a software system, during and after deployment. From the start of the development of a system, the system itself and its requirements will evolve over time due to reasons from various origins (e.g., technical or legal restrictions). Poor requirements determination, requirements gold-plating (adding requirements not part of the original scope) [1], and poor (scope) documentation [2] have been identified as classic mistakes and often contribute to failing IT projects. Managing requirements and their changes are important and contribute to a project's success, as many examples exists of failed projects due to poor requirement management [3, 4]. The field that is concerned with aligning requirements throughout a project's lifecycle is called Requirements Engineering (RE). RE is a systematic process which holds techniques for multiple activities, such as domain analysis, elicitation, specification and evolution [5]. Where the first few activities focus on getting an impression of the system-to-be, the latter aims act on requirement changes and managing them.

Keeping track of these changes in requirements and the development of a software system is what is referred to as software traceability (ST) [6], or requirements traceability (RT) more specifically when one of the artefacts describe requirements. Formally, RT is defined as a relationship between a set of source and target entities [7]. Tracing requirements is a standard practice in the more traditional linear software development methods (e.g., waterfall model [8]) where there is a strict focus on documentation through the development cycle. On the other side of the spectrum, traces in agile development software development methods are often maintained in a less rigorous fashion, there is a loosened focus on documentation and more frequent changes are made in software [9]. Agile development methods (e.g., Extreme Programming and Scrum) have become the mainstream practice of software development in recent years [10]. Most agile methods follow similar principles, like being people oriented, focus on delivering immediate business value [11] rather than its documentation [12].

RT has proven its importance and effectiveness in various forms of software development methods [13], though less is known about traceability for Model-Driven Development (MDD) specifically. In MDD, models are the fundamental basis for the software-to-be. Instead of writing lines of code, models and model technologies are used to simplify and formalise software engineering, which then automatically generate the applications' source code [14, 15], or the models are interpreted in runtime. MDD is the foundation of another development, being the rise of low-code development (LCD) platforms. LCD platforms combine MDD with an automated cloud deployment. The platform provides the developer with pre-programmed user interface components which, together with the user-created models, allows the user to develop applications. Such platforms gained in popularity in recent years, who provide frameworks to build applications without the need of advanced knowledge of a programming language, while also aiming to reduce development time and increase product quality [16]. This brings the ability to create applications to a wide range of users who otherwise were not able to do so.

Previous researches mention the differences between RT in MDD and other software development methods [17, 18, 19] and what is necessary to successfully trace requirements in an MDD environment. A study by Walderhaug et al. on how traceability can be effectively used in MDD projects showed that trace data can be used to generate overviews of traceability information, providing support to the management of MDD projects [20]. Rasiman, Dalpiaz & España conducted research on how to apply Trace Link Recovery (TLR) between artefacts by the use of an automatic tracing tool with the use of machine learning (ML) [21]. This study aimed to boost the use of tracing tools, as they found that the industry often still applies manual tools. It showed the potential of such tools and highlights where opportunities lay, especially related to model performance and complexity.

1.1 Problem statement

Previous studies presented various insights in the differences between RT in different software engineering methods and what automatic tracing tools could mean for the industry. Simultaneously, automatically generating details on MDD projects through traces helps to keep an overview on the project. Though these studies all aim to make improvements in RT in the field of MDD, there are still some limitations.

Despite the various advantages of automated traceability, many companies often practise RT through manual [22] or semi-automated approaches [23]. Both studies acknowledge existing problems with automated approaches and to some extent provide context on how to make automated tools more appealing to the industry. Still, it would be interesting to see if progress has been made since these studies have been carried out considering their date of publishing. As agile software development is reaching higher adaptation, with less focus on documentation, automated approaches to traceability are expected to have developed as well. Besides that, the increased popularity of low-code platforms [24] seem to be a good candidate to support such automated tools because of their often less experienced users [25].

Though there are various approaches to automated tracing solutions, recent tools often makes use of techniques such as ML and deep learning (DL). Therefore, this research will focus on those techniques specifically. The performance of a ML classifying model is often described by a combination of metrics (e.g. accuracy, precision, recall, etc.). However, to make such a model to be used in practise, other aspects are evenly important. Think about the computational intensity of training a model and how that affects the time needed to accomplish those tasks, to give just one example. Other important, although often disregarded aspects, are the generalization of a model and if improvements can be made to reduce overfitting. It would be interesting to see if any improvements may be on the surface.

Considering all of the above, there is still a gap between research results and the needs of and impact on industrial settings. There is a need for a better understanding of what the industry requires considering the developments of the last decade in terms of automated RT. Recent research in automated RT for MDD has led to promising opportunities, but it is believed that more research is needed before it can be made practical for the industry [13, 21].

1.2 Research questions

This research aims to investigate the effectiveness of (semi-)automated traceability in a low-code, or MDD, domain. To do this, the following main research question (MQ) is formulated:

MQ: What is the effectiveness of (semi-)automated requirements traceability in an MDD environment?

To answer the MQ, a number of sub-questions (SQ) have been constructed to decompose the MQ. The first question (SQ-1) should help to get a thorough understanding of the application of traceability in the industry (both in MDD and other development methodologies). The answer to this question should provide a broad picture of current practices.

SQ-1: How do the industrial needs for software traceability in MDD environments differ to those in traditional software development?

Second, as previous research showed us, ML can be used to make improvements to automated traceability tools. The study also showed that still steps can be taken to further improve the model. Therefore, a deeper insight is needed of available techniques that could be of use to make these required improvements on performance aspects (e.g., performance metrics and run-time). Answering SQ-2 will deliver a selection of possible techniques that are available to this objective.

SQ-2: What can the existing state of the art techniques achieve to improve model performance aspects on a machine learning classifying algorithm?

For the third SQ, the aim is to combine the knowledge gathered in the previous two SQ. After SQ-1, we have a thorough understanding of the needs from the industry, where SQ-2 provide the tools to improve on existing TLR systems. Therefore, to answer SQ-3, we should connect the two and have them substantiate each other to get a clear picture of the road ahead to reach those improvements.

SQ-3: How to improve the state of the art in trace recovery to better address the needs of the industry?

1.3 Research objective

Answering the research questions help us to work towards achieving the research objective. The objective can be divided into two parts. The first objective is to have a clear and up-to-date identification of the problems that keeps companies to fully commit to (semi-)automated traceability tools, since a preliminary literature review showed that available literature seems to be obsolete in this domain. Second, this research aims to show how state of the art techniques can help to improve automated approaches for traceability, being a ML classification model for the purpose of this study.

To substantiate the objectives that are described above, the template for technical research problems [26] helps to identify the segments of our research problem.

How to redesign an (semi-)automated trace link recovery system
That satisfies the needs of software engineering practitioners
So that more semi-automated traceability is effective
In low-code development?

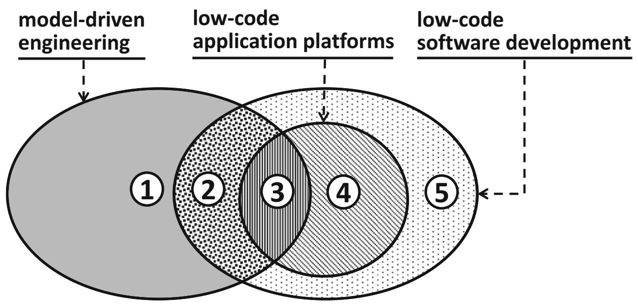
1.4 Assumptions and expectations

Before starting the research, it is important to state the assumptions that are made prior to the study. By doing so, it is clear for all stakeholders on what grounds the research is being performed. Therefore, it is assumed that:

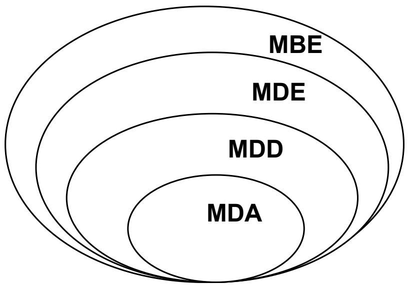
1. data is obtained from a third-party MDD company for the purpose of experimenting on ML models and the improvements that have to be made in that field;
2. the data are representative for other software development projects that are engineered through MDD.

Until now, the terms MDD and LCD are used to refer to similar concepts. Still, it is necessary to state that they cannot simply be used interchangeably when considering them outside this study. For the purpose of this study, when talking these concepts we refer to the intersection of MDD and LCD. Figure 1.1a presents the relationships between Model-Driven Engineering (MDE), low-code platforms, and low-code software development. MDE can be considered as a superset of MDD, where MDE goes beyond the model development activities (see Figure 1.1b). The figure shows 5 regions. Region 1 refers to MDE only. Region 2 are approaches that use models to reduce the amount of coding needed, without offering deployment and/or lifecycle management. The third region is similar to region 2, with the addition of built-in deployment and lifecycle management (i.e., LCD platforms). Region 4 and 5 can be considered similar to region 2 and 3, without being model-driven (i.e., region 4 includes deployment and region 5 does not). For this study, we look at the regions that are labelled as 2 and 3.

Besides stating the made assumptions, it is good practice to express the expectations of the outcome of the research. This helps to set a direction for all stakeholders and to be able to assess the initial intentions afterwards and, if there is a contradiction between expectation and outcome, to see how and where it occurred. Besides the general expectation to finish the research in time, it is expected that at the end of this study, we:



(a) Venn diagram relating to MDE and low code (applications/-software development) [27].



(b) MDE vs. MDD. Adapted from [28].

Figure 1.1: Venn diagrams visualising the intersection between MDD and LCD.

1. have a clear understanding where the opportunities lay for companies to implement or switch over to automated approaches for traceability;
2. have made improvements to the ML models measured across many angles, such as raw performance and speed.

1.5 Research variables

Visualising our research variables helps to illustrate the relationships between the different variables that are included in the study. The variables that are presented in Figure 1.2, which are derived from the research questions as introduced in Section 1.2.

Four types of variables are displayed in the model [29, 30]. The *dependent variable* is the outcome variable (i.e., the one where we see the change). In the case of the study, the effectiveness of automated traceability is evaluated. *Independent variables* are the ones that are used to influence the dependent variable. In other words, we try to influence the effectiveness of traceability by using the most recent available aspects present in academic research and industrial practices. An intermediate state of independent variables, a *mediating variable*, allows to better link the (in)dependent variables with each other. Finally, the model shows one *moderator variable* that alter the effect of other variables to alter the effect on the upcoming mediating or dependent variables.

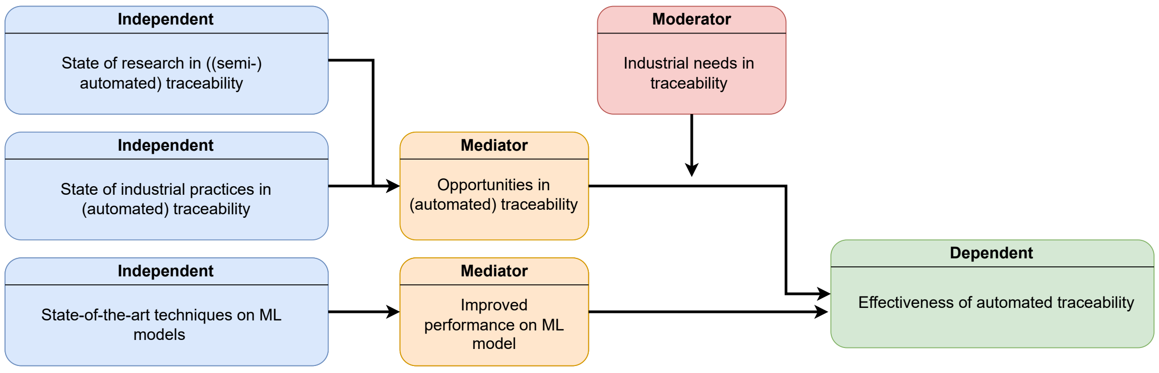

Independent
State of research in ((semi-) automated) traceability
Moderator
Industrial needs in traceability
Independent
State of industrial practices in (automated) traceability
Mediator
Opportunities in (automated) traceability
Dependent
Effectiveness of automated traceability
Independent
State-of-the-art techniques on ML models
Mediator
Improved performance on ML model

Figure 1.2: Research variables

Chapter 2

Research method

The research problem in this study is being approached as a knowledge question. The empirical cycle as introduced by Wieringa [26] aims to support researchers with a problem-solving process for doing design science in information systems . The cycle will be used to answer scientific knowledge questions.

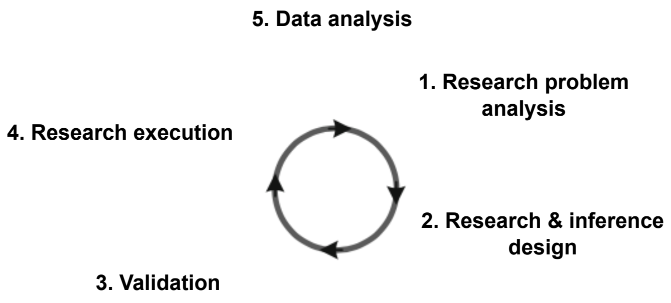


Figure 2.1: Empirical cycle. Adapted from [26]

The cycle has five main sections that provides a structure of a rational decision cycle (see Figure 2.1). The first step, **research problem analysis**, has started in the previous chapter where we discussed the conceptual framework and provided the questions to be answered in this study. Next, the **research design** is described in this chapter. A brief introduction on the design is given in the upcoming paragraph. The **validation of research** asks to validate if there is a proper match between research design and inferences from the data. Three main aspects are included in this part. First, we want to be sure the research does support the inferences. A validity evaluation will be included in the final report to elaborate on this. Second, the design must be specified in a way that the research is repeatable. This is supported by carefully going through every step of each research method that is described in this chapter while also reporting on various intermediate results (e.g., a literature quality assessment). In addition, the source code that used in this study is made publicly accessible. Finally, if, at any point, people are involved in the study (e.g., through treatments), ethical norms will be respected in relation to the participants. In this case, a consult with the supervisors of this study will point out if an ethical review (through the Ethics Review Board of the faculties of Science and Geosciences of the University Utrecht) shall be requested. **Research execution** happens according to the research design. It is possible that unexpected events occur and should be dealt with. It is required to report on unexpected events that are relevant to the interpretation of the results. **Data analysis** is done through the data synthesis that is described for each described method.

This research makes use of method triangulation to analyse the research problem from several perspectives. This means that various data collection methods will be used to gather the required knowledge and to be able to approach the problem from different angles. The methods used for this research are (1) a systematic mapping study, (2) a secondary analysis, (3) an experiment, and (4) a focus group. Each of them are discussed accordingly in the upcoming sub-sections. A visualisation of the mentioned research methods is presented in Figure 2.2.

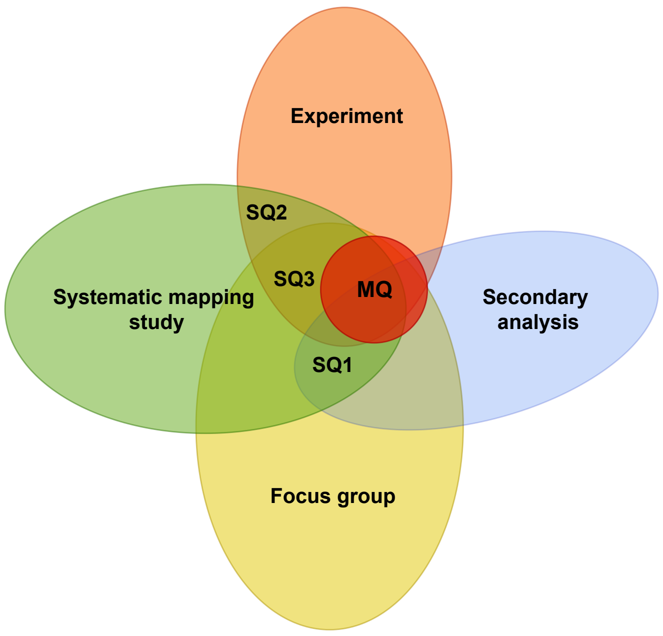


Figure 2.2: Venn diagram of research method triangulation

2.1 Systematic mapping study

To have a comprehensive picture on available literature related to the subject, a systematic mapping study will be executed. A review on the literature helps to gather all evidence related to the research questions and is done through a three-step process [31].
This method supports SQ-1, SQ-2, and SQ3 in the following manner. On the way to answer *SQ-1*, literature provides a first understanding of current practises in scientific and industrial settings. A preliminary review on literature already showed that enough literature should be available to establish that baseline. Knowledge that is gathered in this phase can then be used to have a more specific focus during data collection in the upcoming methods. For *SQ-2*, the search field directs towards the field of ML and especially state-of-the-art techniques that can support and improve processes in that field. The aim is to finalise this part with a good understanding of the possibilities and have a selection of candidate techniques that can be used later in the study to experiment to test their usability and performance regarding the research. Next, to support answering *SQ3*, literature should provide knowledge on how state-of-the-art in trace recovery can contribute to the needs of the industry in software development.

It is expected that, when finishing the literature review, there is a thorough understanding of:

1. the practices of manual and (semi-)automatic tracing tools, according to literature, in different industries of software development;
2. what keeps one to start using (semi-)automated traceability approaches in those industries;
3. how ML classification algorithms work and what techniques are available that contribute to a more efficient use of such algorithms;
4. the differences in software projects coming from (1) MDD specific software projects and (2) non-MDD specific projects;
5. what elements should are part of a ML pipeline and if they can be optimised;

and have a;

1. selection of where opportunities regarding (semi-)automated traceability for MDD environments lay;
2. selection of candidate techniques that can be used to set up an experiment to test the (different combinations of) techniques

2.1.1 Literature research protocol

The remaining part of this section can be considered as our literature research protocol for our systematic mapping study. The protocol is based on the steps as described by Kitchenham & Charters [31] and consist of the bulleted list below. Each of the items is discussed in the upcoming sections.

- Search strategy
- Study selection criteria
- Study selection procedures
- Study quality assessment checklist and procedures
- Data extraction strategy
- Synthesis of the extracted data

Search strategy

The search strategy relates to the plan where to search for to get the required literature. This includes two things, the resources where to search for and what the search strings should be.

Resources. Resources could be anything ranging from specific digital libraries down to a journal level. For the purpose of this literature study, Google Scholar will be used to gather the literature. This choice was made since most relevant journals can be accessed through the search engine in combination with the university's proxy. Other sources retrieved by the search engine, such as technical reports and theses, are accepted as well. In addition to the results that are returned by the search engine, *snowballing* procedures can be used to find additional literature. This could include both forward snowballing (i.e., looking at papers that cited the initially found paper) and backward snowballing (i.e., following references in the found paper) [32].

Search terms. By carefully considering the terms that will be used to feed to the search engine, it is possible to provide some direction to the results that will be returned. This is important, as just simply using words in the query could return a bunch of documents that are only related to just one of the terms that you put in the query. Therefore, the queries that are listed below are specifically chosen to gather the right information relating to the sub-questions that we would like to answer. The queries also make use of search operators (e.g. AND, OR, or the use of quotation marks) to have a more all-encompassing search query. An overview of the search queries can be found in Table 2.1.

Table 2.1: Overview of search queries.

SQ	Search query
SQ-1	- "Traceability" AND ("software development" OR "software engineering") - "Traceability" AND ("MDD" OR "Model-Driven Development") OR ("MDE" OR "Model-Driven Engineering") - ("Automated" OR "semi-automated" OR "manual") AND "traceability" AND (tools OR software OR industry)
SQ-2	- (Machine learning OR ML) AND "classification" AND (algorithm OR model) - (Machine learning OR ML) AND ("model" OR "algorithm") AND ("improvement" OR "performance") - ("trace" OR "traceability") AND "recovery" AND "machine learning" AND ("model improvement" OR "feature reduction")
SQ-3	- "Trace recovery" AND (industry OR industrial) AND needs - "Pipeline" AND "machine learning" AND (model OR classifier)

Selection criteria

Not every result that is returned by the search engine is automatically useful for answering the SQ's. Defining the inclusion criteria prior to the data collection helps to reduce bias, though slight deviations could become necessary while collection the data (e.g., during quality assessment). The inclusion and exclusion criteria are listed in Table 2.2.

Table 2.2: Overview of selection criteria

SQ	Criteria
SQ-1	**Inclusion**
	Any study that explains the general idea of traceability, or tracing of artifacts (e.g., requirements), in software development.
	Any study that either compares practise of traceability in MDD against non-MDD methods, or elaborates on one of them with a detailed exploration on that specific method.
	Any study that gives an elaborate overview of aspects regarding MDD in general
	Any study that reports on either how the industry uses one of the various tracing approaches, or a comparison of two or more approaches, not limiting to real-world applications.
	Any study that illustrates or clarifies the gap in the industry for a wider adoption of automatic traceability tools.
	Exclusion
	Studies that do not provide information on traceability, or when they do they are not linked to either requirements of software systems.
SQ-2	**Inclusion**
	Any study that report on detailed information regarding the classification algorithms, both theoretical and practical.
	Any study that proposes either opportunities for model improvements, or compares a selection of techniques that can be used.
	Exclusion
	Studies that compare algorithms other than the ones reviewed in this research or not those that are not specifically for classification purposes.
SQ-3	**Inclusion**
	Any study that proposes approaches or solutions regarding trace recovery to better address the needs of the industry for software development.
	Any study that presents a blue print or proposes a (partial) design of a pipeline regarding machine learning models.
	Exclusion
	Studies that do not involve aspects in the pipeline that are included of this study, such as training data and feature selection.

Selection procedures

The selected studies from the previous step will be subjected to another selection round which is based on more practical issues. This could be because the literature is too old to be relevant for this subject or the language of the paper is not in a language that is understood by the researcher. The studies will be selected according to the aspects that are listed below.

Language. The language used in the primary studies to have them included in this review should be either in English or Dutch. **Sources.** All sources that are relevant to the subject of research are allowed to be used in the review, meaning no specific sources (e.g., journals or authors) are listed to be excluded. Still, a list with excluded papers or other sources can be established if there is a reason to do so while assessing its quality. **Date of publication.** Generally, the more recent a study is the better, especially considering the subject of this research and its advantages over the past few years. Therefore, studies not older then 15 years will be included in the review with the exception of subjects that are only used to explain general (established) aspects relevant to the study. **Relevance.** Only studies that actually are related to the used search terms in the designated fields of interest are included for this review to prevent certain studies to be selected due to terms being homonyms.

Since the studies are selected by a single researcher, a test-retest procedure will be performed. This procedure is used to check for consistency of the inclusion and exclusion decisions, as proposed by Kitchenham et al. This means that a second selection procedure is performed on studies that are randomly selected to check for a consistent output of the study selection (see online appendix for the Python script that was used to support the random selection [33]). In the end, 40 percent of the selected studies are subjected to the test-retest procedure[1].

[1]Based on a literature study by Mahomed et al. [34] where the average of test-retest rate was taken {75, 8.5, 19.8, 60}% between subjects that were put through a the test-retest procedure over the total number of subjects. The authors reported on data from multiple studies.

Quality assessment

Besides having the selection criteria, it is good practise to assess the quality of the selected studies before reviewing them for this research. By assessing the quality it either helps to only include high quality literature or adjust the selection criteria concurrently to have a more strict selection for the continuation of the data collection. A checklist, based on the procedure as used by Fliefel [35], is used to assess the quality of the found literature and can be found in Appendix E1. For the purpose of this research, the procedure is adapted, where a study is assessed on twelve points across six categories. In general, the checklist aims to evaluate the credibility of the research method and data, the generalisability of the research, and if the right conclusion have been taken.

Each of the studies passing the selection procedure is subjected to the quality assessment. The output of the assessment is a quality score for each assessed study and based on that score, studies can be allocated to a quality coefficients. Fliefel makes use of three possible quality coefficients: scores that fall in the range of 0.8-1 are labeled as *excellent*, between 0.5-0.8 studies are of *average* quality, and all lower than 0.5 is reported as *poor*. Studies that are labeled as poor will be excluded from the review.

Data extraction

The extraction of data regards to the collection of the information that are of value for the objective of this study. This part of the process is well documented through Data Extraction Forms (DEF) to reduce the opportunity for bias and to establish the reproducibility of the review. The DEF include three sections, being (1) general information and study related details (e.g., name or reviewer and date of extraction, and authors), (2) study related specifics (e.g., study design), and (3) the outcomes of the study (e.g., outcomes and measurements).

The DEF that is used for for this review is presented in Appendix F, which is composed with inspiration from others [31, 36, 37].

Data synthesis

Data that has been collected is summarised and reported on to be of use in this research. This may also include graphical representations if this contribute to providing a clear picture on the literature. Graphical representations can be either a direct copy from the source or one that is adapted to better fit its purpose.

Protocol summary

The literature review is conducted through a systematic mapping study. The aim of the review is to find answers to the research questions to be able to compare those to the findings in the other research methods linked to those SQs. To do this, the scientific search engine of Google (i.e., Google Scholar) is used and is fed several search queries (see Table 2.1). This provides us with a selection of potential sources, which can be extended when applying snowballing procedures. These studies are subjected to selection criteria (see Table 2.2), that will filter the selection of studies. Next, the studies that passed the previous step are subjected to a selection procedure and a quality assessment to ensure the review involves studies that are both relevant and of high quality. Data is extracted from the remaining studies by the use of a DEF. The extracted data is then synthesised to report on it.

2.2 Secondary analysis

The next step in this research is performing a secondary analysis. This type of research method is concerned with reusing existing research data with the purpose answering different questions when compared to the original work [29]. As described by Verhoeven, this type of data collection has several advantages and disadvantages that need to be addressed. Most importantly, it saves time and money for this research to reuse data. Simultaneously, it is expected that it concerns data that has already been cleansed which again saves time and effort. The down-side to this is the lack of control over the data and errors could have occurred during the data cleansing that cannot be detected anymore. These disadvantages have to be considered before choosing to use that specific data set. To ensure a high-quality data set, a framework for quality assurance has been established. This is further explained in the designated section (see Section 2.2.2).

Using secondary analysis helps to get a step closer to answering our research questions, which will be primarily be focussed on SQ-1 and SQ-2. In that order, the data (which is described in the upcoming subsection) allows us to understand the differences between MDD and non-MDD software projects. It is expected that this comparison will either substantiate the findings from the literature study regarding this subject, or can be used to explain the possible gap in the literature in this aspect. Secondly, the data is used for experimentation purposes to study the effect of the proposed candidate techniques (again coming from the literature study) and see if applying one or a combination of multiple techniques can lead to the desired performance improvements.

It is expected that, when finishing the secondary analysis, there is a thorough understanding of:

1. what kind of attributes in the data are present and related to MDD software projects compared to non-MDD software projects;

and have a;

1. selection of MDD specific data sets that are suitable for the experimentation phase of the research;
2. positive or negative confirmation of what has been found in the literature of the difference between MDD and non-MDD software projects

2.2.1 Data sources

Two data sources have been identified for the purpose of this secondary analysis. Each source does supply multiple data sets which relate to different software projects. The choice is made to select these sources, because (1) they both contain similar details on the projects (being project issue tracking and project version control details) and (2) where one source is limited to projects in an MDD context, the other is not.

The first source is a company called Mendix, which is the developer of a similarly called low-code development platform Mendix Studio (Pro). The data sets were obtained through a collaboration between the company and the previously mentioned study by Rasiman, Dalpiaz & España. For their study, they used data on four projects with an additional three projects which they got for an extended study [38]. These data sets were made available for this research as well. The second source relates to a study by Rath & Mäder, wherefore they collected data on 33 open-source projects [39]. For the purpose of this research, not all projects will be included or processed to be analysed because a selection of those will already meet its goal. Note that these two sources are quite imbalanced in size. After the data quality check, if necessary, a selection is made for the data sets that will be used for the continuation of this study.

2.2.2 Data quality assurance

To ensure that the data sets coming from secondary research meets the quality that is required for this study, a quality assurance framework is established. This framework is based on a workshop given by the European Medicines Agency (EMA) in 2022 [40]. In this workshop, a number of dimensions were opted for consideration when data quality needs to be assessed. The more dimensions a framework covers, the more extensible the framework is for other data types. After a preliminary review on the data, a selection is made that provides us with a ‘good enough’ dimension set.

The considered dimensions and their meaning can be found in Table 2.3. The meaning of one or more dimensions can be different compared to the ones introduced at the workshop by EMA, as an alternative interpretation make them fit better to the data used in this research.

Table 2.3: Data quality dimensions.

Domain	Meaning
Uniqueness	Data is unique in the sense that a data set cannot be a subset of another data set
Completeness	The minimum required data attributes are present in the data set[2]
Consistence	Conformance of data values to other values in the data set
Relevance	The data is relevant to the purpose of this research
Timeliness	The data must be captured recently, or at least in the same time period across different data sets

2.2.3 Analysis approach

To perform the secondary analysis, a four step approach is followed. Each step is based on published work and then combined for the purpose of this analysis, which consists of the steps below. What follows is an explanation of each step and its activities.

1. Data description
2. Variable selection and visualisation
3. Revision characteristics
4. Presentation and conclusion

Data description. The first step is to describe and thoroughly understand the data that will be used for the secondary analysis [41]. To get to this understanding, several activities are performed. First, both sources are studied. This should provide us more context about where the data is coming from. After that, the priory described data quality check is performed to filter out data sets that cannot be of use. Simultaneously, the structure of the data is analysed and documented.

Variable selection and visualisation. After the data structure has been analysed, variables can be determined. This includes the generation of operational definitions of the outcome and exposure variables [42]. Since data is coming from multiple sources, this step helps to establish uniformity. The next step is to use descriptive statistics to quantify the data and the variables in it. Frequency tables should help to further visualise the data and provide information about the presence of the variables.

Revision characteristics. To better understand the characteristics of commits made to an MDD software project and compare these to their non-MDD counterparts, we want to see if we can discover differences in terms of the vocabulary used in those commits. To do so, we use an approach by Alali, Kagdi & Maletic [43] who studied the vocabulary across nine open source (non-MDD) software projects and compared this to three size-based characteristics (i.e., number of files, lines, and hunks). However, for the purpose of this analysis, we focus primarily on the former and use their approach to make a comparison between MDD and non-MDD projects.

The approach by Alali, Kagdi & Maletic consist of the seven steps as displayed in Figure 2.3. The steps are discussed in more detail in the Section 4.2.3.

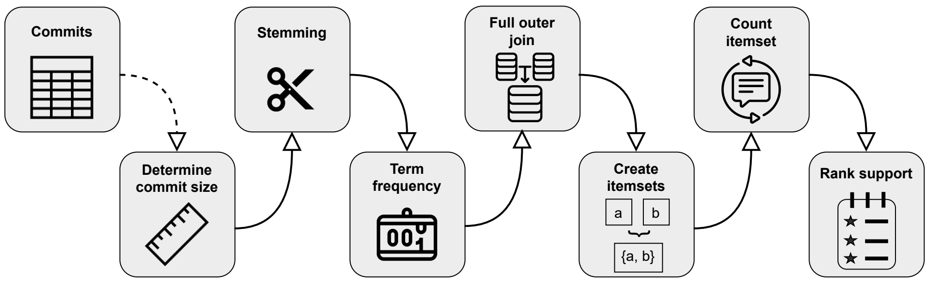


Figure 2.3: Analysis revision characteristics process visualisation (left to right)

[2]Minimum attribute list (see Appendix A)

Presentation and conclusion. The results are presented in a tabular fashion and graphical interfaces which includes standardised metrics, that allows us to carefully consider the rules that were mined. It is expected that from these results, we can distinguish certain aspects between the two development methods.

2.3 Experiment

The data that has been collected in the previous phase can now be used for the experiment. The goal of the experiment is to test a selection of candidate techniques that are identified in the literature study and their effect on the performance of the ML model. The experiment continues on the results from Rasiman et al., where he identified the best configurations for the two traceability scenarios (i.e., trace recommendation and trace maintenance). The experiment will be executed by following the steps as mentioned by Wohlin et al. [32]. The steps are listed below and will be further elaborated on in the upcoming subsections.

1. Scoping
2. Planning
3. Operation
4. Analysis and interpretation
5. Presentation and package

It is expected that, after conducting the experiment, there is a thorough understanding of:

1. the effect on the performance of each, or a combination, of the candidate techniques
2. the (dis)advantages of each of the candidate techniques after using it on actual data

and have a;

1. optimal technique available for the purpose of improving the classification model by Rasiman [21]

2.3.1 Scoping

The scope of an experiment is about defining the purpose of the experiment and its goals. To define the scope of the experiment, the goal template by Basili & Rombach is used [44]. Similar to the template for technical research problems (Section 1.3), the template helps to define the scope in a way that is quick to read but covers all important aspects. First, each component is discussed and afterwards a brief summary is given.

Object of study. The object of study are the candidate techniques that were identified as potential for improvements to the classification model through the literature study.

Purpose. The purpose of the experiment is to test the ability of the candidate techniques to boost the performance of the ML classification model. The aim is to see if the feature space can be reduced while keeping the models' performance equal or higher compared to the performance as measured by Rasiman et al. [21]. Studies have identified that a large feature space increases training time (i.e., cost), increases noise, and could introduce the curse of dimensionality [45, 46]. Therefore, this experiment aims to assess whether one or a combination of techniques can play a role in improving the model by increasing its performance or by making it more efficient (similar or higher performance while being faster).

Perspective. The perspective is from the point of view of the researchers and developers who work with MDD. The previous study already showed promising results for low-code development platforms and this experiment provides more insight in possible future improvements. The experiment only considers the two traceability scenarios as result of previous research by Rasiman.

Quality focus. The effect studied in the experiment will be in the performance of each studied technique. The overall performance is measured in a 2-dimensional space, where the overall performance is determined by performance metrics (i.e., metrics based on recall and precision) and the number of features left in the feature space. The latter is on its own correlated with the time it takes for a model to perform its task, making it relevant in terms of efficiency.

Context. This experiment is run in the context of ML classification models. The candidate techniques will be applied to the ML algorithm, which is run in a programmed Python notebook as a result from the study by Rasiman et al. The experiment is focused on Gradient Boosted Machine algorithms with combinations of rebalancing strategies, again based on the results of previous research.

Summary of scoping

Analyse the candidate techniques
for the purpose of evaluation
with respect to their performance and efficiency
from the point of view of the researcher and developers in MDD
in the context of the execution of ML classification models.

2.3.2 Planning

To plan the experiment, various steps have to be taken. Similar to scoping the experiment, each individual step is discussed accordingly.

Context selection. The context of the experiment is a ML classification model that uses Gradient Boosted algorithms. The experiment is conducted by a single graduate student for the purpose of a master's thesis and is run off-line, and the experiment is specifically focused on data from MDD software projects. A real problem is being addressed through this experiment, which is the lack of efficiency and explainability of the created model. This is especially relevant for developers in the context of traceability in MDD. The use of the models is still in an experimental phase, as it yet has to focus on the implementation for the targeted systems, such as low-code development platforms.

Hypothesis formulation. To state formally what is going to be evaluated with this experiment, the following hypothesis is formulated.

> Null hypothesis (H_0): There is no effect on the performance of the ML classification model when applying the different candidate techniques.
> H_0: $\mu_1 = \mu_2 = ... = \mu_n$.
> Alternative hypothesis, H_1: At least one of the treatment means is different.

Variables selection. Choosing the right variables allows us to control and measure the effects within the experiment. The *independent variable* is the classification model improvement technique, and each of the candidate techniques found through our literature study is a treatment. The effect of the treatment is measured in the *dependent variable*, which is defined as the performance of the classification model.

Selection of subjects. To select the subjects, or in the case of this experiment, to select the software projects to conduct the experiment with a sample is taken from the data sets that passed the quality check (see Section 2.2.2). *Convenience sampling* is then the technique used to select the projects that will be subjected to the experiment.

Experiment design. The experiment is setup as an *one factor with more than two treatments* design. This type of design allows to compare the different treatment means of the dependent variable for each treatment.

Instrumentation. Instrumentation relates to three specific types, being *objects*, *guidelines*, and *measurement instruments*, which are discussed respectively. Since this experiment is conducted solely by a computer system, Instrumentation *objects* are limited to the computer system that is running the program itself. For the experiment, a single computer is used to reduce the risk of variations in performance (more on this in the upcoming paragraph). Similarly, since the experiment does not involve human subjects, no guidelines need to be prepared to guide participants through the experiment. The measurements that are produced by the experiment are being collected and saved by the *measurement instruments* from the used Python program. This includes an evaluation

Python script (created by the author of previous research, Rasiman et al.[21]) and should be slightly revised for the purpose of this experiment. Necessary revisions include changes, such as removing of unnecessary activities on unused classifiers and slight improvements on the notebook itself (as found during work on [38]). The revised Python program is the experimental environment for this experiment.

Validity evaluation. To determine the generalisability of the experiment's results, it should be evaluated how valid the results coming from the experiment are. The validity, according to Wohlin's taxonomy [32], can be harmed by four types of validity threats, with each of them discussed below.

With *Conclusion validity* we want to make sure that there is a statistical relationship between the treatment and outcome. Ten data sets are used in the experiment, which contributes positively to the statistical power of the experiment. Yet, the data sets all come from the same company. A positive aspect here is that the data sets come from three separate software development teams, though a more heterogeneous sample would be more beneficial. To counter the threats, the experiment for each treatment is conducted ten times.

Internal validity aims to ensure that the treatment is indeed the variable that causes the outcome of the experiment. To be as sure as possible that the treatment causes the outcome, the experiment is conducted with the use of a single computer to exclude variations in hardware and/or software. The experimental environment is designed to deliver results without interaction from the researcher in between a run of the experiment. There is a risk concerning the data, because it is provided by an external company. This means that the researcher has no influence on the data that is collected and there is a risk of bias on the company's selected data sets. However, the attributes used from the data are carefully considered and only attributes that are common across the data being subjected to the experiment. There were no attempts for repairs to reduce the risk of potential bias.

Construct validity refers to the generalisability of the results of the experiment in relation with the concepts or theory behind it. A threat belonging to this category is the risk of insufficient defined constructs before translating into measures. To measure the results of the experiment, metrics are used that are widely used in literature. These metrics ($F_{0.5}$ and F_2), that combines the metrics precision and recall, are also used by others [21, 47] who conducted similar experiments to research the same traceability scenarios. However, other F-metrics could be more appropriate for these studies scenarios. Also, since single treatments are possibly combined in this experiment to test if they can together contribute to better performance, there is an interaction between treatments. This could make it hard to determine which treatment causes which portion of the effect. To mitigate this risk, these combinations are limited to two treatments, with each treatment also subjected on its own. This allows us to distinguish the different treatments in this experiment.
External validity regards to threats that limit our ability to generalise to industrial practice. The results from this experiment mostly hold for the software projects at low-code development platform Mendix. More data from other sources (from other low-code development platforms or clients) in the same context would be beneficial to minimise the risk of the results not being representative.

2.3.3 Operation

The operation of an experiment consists of three tasks, being preparation, the actual execution and data validation. Each step is discussed accordingly in the next few paragraphs.

Preparation. Prior to the execution of the experiment, three things need to be done to prepare the experiment. First, we need to *populate the features* on the 10 software projects that were delivered by Mendix, so that they can be subjected to the feature selection algorithms. These populated features are then run through the classification evaluation notebook to *create a baseline of the performance of the classifiers* on the total feature set as well. The results from this allow us to compare the effect on the reduced feature set for each data set. The populated features are then carefully named and organised in the directory, so that the Python notebook is able to retrieve the data sets for the experiment. Finally, *Python notebooks must be prepared* to facilitate the execution of the experiment.

Execution. The execution of the experiment is supported by the priory mentioned Python notebook. Each run includes applying one of the feature selection algorithms and running the evaluation script on that subset of populated feature data. After each run, the output of the evaluation is properly documented in the directory, so that we have everything organised for future steps of the experiment.

Data validation. After each run, the data is validated to make sure the output is reasonable and collected properly. The Python program includes some tools to support a quick validation, like clearly identifying the used technique within each file so that it reduces the risk of mixing data of various runs. Also, since the output of each run does not include extensive set of data, it is straightforward to identify outliers that could scent errors in the evaluation script.

2.3.4 Analysis & interpretation

Following Wohlin's approach, this part follows three steps to be able to draw valid conclusions. The first step described the data through descriptive statistics. For the second step, the data is cleansed from false and abnormal data points. The final step consists of hypothesis testing, where the hypothesis (see Section 5.2.3) is statistically evaluated. Each step is described in more detail.

Descriptive statistics. The data is described with the use of descriptive statistics. To describe the measure of central tendency, the mean and range are metrics that sufficiently provide a first understanding of the data sets. In addition, the standard deviation describes the dispersion from the mean using the same dimensions as the data itself. The findings on the analysis on these metrics are visualised through Box and Whisker plots to see the characteristics of the data. This also helps to identify outliers, that should be opted for (possible) removal. Furthermore, the data is displayed in a tabular fashion with colourised table cells to highlight the important aspects.

Data set reduction. Although data set reduction generally leads to higher quality data sets to use in experiments, for the purpose of this experiment it is decided not to perform this step. As said in the validity evaluation of the experiment (see Section 2.3.2), no attempts to improve the data set will be made as it would increase the risk of bias.

Hypothesis testing. To test the hypothesis as stated in Section 5.2.3, the Wilcoxon Signed-Rank test and Friedman test will be considered [48]. While both are non-parametric tests, the Friedman test is an extension of the Wilcoxon test. However, since we have more then two treatments in this experiment, the Friedman test is preferred over the Wilcoxon test. If chosen for the Wilcoxon test, the test must be performed several times which increases the risk of a Type-I error. The Friedman test, as alternative to the ANOVA test, allows us to test differences between treatments. This test is sufficient for this experiment, as we have results for every treatment on all ten data sets. Considering only ten data sets, we are limited to only non-parametric tests. Because we have more than three treatments in the experiment, a post hoc test is required to study the effects between the treatments. The Nemenyi test is used as post hoc test, which utilises the standard error metric to test the difference between rank sums.

2.3.5 Presentation & package

The results of the experiment are presented as part of a final report, which includes a discussion on the results and its data analysis. The results will compared to other findings in this research as well. All source code that was received as a result of previous research, including necessary modifications, are included in the final deliverable.

2.4 Focus group

As visualised in Figure 2.2, a focus group is used to gather data to answer SQ1 and SQ3. A focus group is a qualitative research method that aims to collect rich and detailed information from experts [49]. The structure of a focus group may differ, but often it entails a semi-structured (interview) group session hosting between 6 and 12 experts.

It is expected that, after the focus group, there is a thorough understanding of:

1. the issues that industry experts run into when practising traceability;
2. what industry practitioners value and that contributes to more efficient traceability practices;
3. where we can find opportunities to further improve the proposed scenarios.

2.4.1 Focus group discussion

Through interviews, we want to get an understanding of present-day issues according to industry experts and then identify where value may be that could lead to improvements for those issues. The group discussion is done through a single focus group, that means we want to establish an interactive discussion about our topic between all participants. The focus group discussion consists of the four steps as described by Nyumba [50], being (1) research design, (2) data collection, (3) analysis, and (4) reporting. Each step is discussed below.

Research design. The structure of the group discussion will be semi-structured. To assemble a structure, we state the research objectives that we aim to accomplish through the discussion.

The objectives for the discussion are:

1. Understand the (multiple) viewpoints on traceability in MDD amongst industry experts
2. Know what issues are there that keep developers from properly utilising traceability
3. Understand the needs and the value of traceability practices in MDD environments
4. Know what is necessary according to industry experts to increase traceability practices

Based on those objectives, a list of questions is prepared as guidance for the discussion session. The structure of the discussion and the questions can be found in the interview protocol (see Appendix B1). The focus group consist of experts regarding our field of study from both Mendix and its parent company Siemens. The discussion session will be held online and will take approximately 1-1.5 hours. Prior to the discussion, a short presentation is given, so that all experts are up-to-date on the current standings of the research.

Two important roles are introduced to guide the session, being a first- and second moderator. The first moderator facilitates the discussion process, the second moderator makes sure the session is properly recorded and can take notes during the session [51]. The second moderator can monitor if all participants are equally given time to speak as well. Prior to the session, all participants are given a consent form that they need to sign (see Appendix B2). This form makes sure the participants understand aspects, such as the intent of the session, that they can stop participating whenever they want, and what the data is used for.

Data collection. Data will be collected in multiple ways. For one, the session is being recorded. This will result in a transcribed (and anonymised) data set. Notes can be made during the discussion, but it is assumed that this is only used to better facilitate discussions rather than to be used for data analysis. During the session, the experts are asked to rate their view on presented statements through a Likert-scale (through a questionnaire or supporting platform like Wooclap).

Analysis. The analysis uses both quantitative as qualitative approaches. The answers through the Likert-scale questions are especially suitable for quantitative analyses. This will mainly include derivatives based on the data being visualised. For the qualitative data, the analysis is approached by applying content analysis. By systematical coding the data, we aim to discover patterns that are undiscoverable by only reading the discussion transcripts.

Reporting. After analysing all the data, the results will be presented into a coherent report. The results are also being discussed and compared to findings from the other research methods to see if these support or contradict each other.

Chapter 3

Literature review

All information as disclosed in this literature review is a result of the execution of the steps as described in the literature research protocol (see Section 2.1.1). Prior to presenting the findings of the review, the intermediate results are presented in the paragraph below.

Sources that were returned by the search queries were subjected to both **selection criteria** and an additional **selection procedure**. Studies that met the selection criteria as presented in Section 2.1.1 and thereafter was found sufficient based on the additional procedure. A complete overview of the found literature can be found in Appendix C, with the results of the test-retest procedure presented in Appendix D. The table that displays the studies in Appendix C are the studies that are subjected to the **quality assessment**. The results of the quality assessment are displayed in tabular fashion (see Appendix E2). The studies that passed the quality assessment were opted for **data extraction**. For each eligible study (with a quality coefficient of average and higher), a DEF is completed and each of them can be found in a combined file in the online appendix [33].

This review presents what has been written in literature. Section 3.1 explores the ins and outs of traceability, its use in practice and visualisation techniques. Section 3.2 starts with explaining the concepts of MDD and then goes into detail about linking it to traceability. Next, an elaborate overview is given on recent work in automating trace link recovery (see Section 3.3). Based on the techniques used there, Section 3.4 presents insights and opportunities on how to improve the used approach presented in the section before. The final subsection summarises all that we have learned through the systematic mapping study (see Section 3.5).

From this point forward, the review will focus on two traceability scenarios, being trace maintenance and trace recommendation as the main scenarios that we are interested in for this study. These scenarios are inspired from the studies by Rath et al. [47] and Rasiman [21]. Both scenarios aim to recover trace links between artefacts coming from commit data and issue tracking systems. The trace maintenance scenario is an approach where trace links between artefacts are recovered (and establishes valid traces) in an automated fashion. Trace maintenance is something that could be done outside office hours, where the system would link artefacts to support the developers with their traceability practices. In this scenario, one would require the system to be as precise with the recovered links as possible because of this automated nature. Trace recommendation is about offering a developer potential trace links when a commit is made, so that the developer can make the valid trace links themselves (i.e., semi-automated). Since this scenario requires human intervention on possible trace links, recall is more important to offer multiple highly potential links to the developer. This scenario applies whenever developers are making commits.

3.1 Traceability: a general perspective

Traceability can be defined in numerous ways for various fields each with a slight different meaning, so let us first establish a definition of the term based on what is said in literature. To come to this definition, we first explore traceability in both RE and software engineering. This means that when we discuss traceability, we primarily focus on traceability in the discussed fields of interest and exclude others (e.g., food or money traceability).

3.1.1 Defining traceability

A definition for *requirements traceability* has been proposed by many. A frequently cited definition [21, 52, 53, 54] is from Gotel and Finkelstein [55] who define RT as: "the ability to describe and follow the life of a requirement, in both forward and backward specification, to its subsequent deployment

and use, and through periods of ongoing refinement and iteration in any of these phases". A second interpretation that is cited multiple times [21, 54] is that by Pinheiro [56]: "the ability to define, capture, and follow the traces left by requirements on other elements of the software development environment and the traces left by those elements on requirements". This suggests that Pinheiro only considers traceability from the point of formally documented requirements.
Traceability in software engineering (referred to as *software traceability* (ST)), as used in research [13, 57], is described as: "the ability to relate artefacts created during development of a software system to describe the system from different perspectives and levels of abstraction with each other". A similar understanding of the term has been given by De Lucia et al. [58], where it is referred to as: "the ability to describe and follow the life of an artifact (requirements, code, tests, models, reports, plans, etc.) developed during the software lifecycle in both forward and backward directions". A third definition is provided by the Center of Excellence for Software and Systems Traceability (CoEST) [59], an organisation with researchers and practitioners that work together to achieve traceability solutions, that extents the definition by Gotel and Finkelstein to include software traceability: "encompassing multi-directional traceability centered around diverse artifacts".

We can see that both terms share similar grounds. This does not necessarily imply that they can be used interchangeably, but one can argue the opposite. The definition by Pinheiro already speaks of a relationship between requirements and the software development environment. In publications by Sundram et al. [57] and Spanoudakis and Zisman [13], both use either one of the two terms (RT or ST) as article keywords but use the other to define traceability in the aticle itself. This is substantiated by work from Santiago et al. [17] where take a more general definition for *traceability* as: "keeping track of the relationships between requirements, design artefacts, source code, test cases, etc.". Finally, De Lucia et al. [58] actually applies the definition by Gotel and Finkelstein for RT, but replaces 'requirement' for 'artefact' to define the term 'software artefact traceability'. Besides, requirements are also referred to as a software artefact [13, 58].

Considering all the given arguments, for the purpose of this review and continuation of the study, we will take the adjusted definition by De Lucia et al. on the originally proposed definition by Gotel and Finkelstein. Hereafter, when referring to (software) traceability, we include requirements as a software artefact as addition to other artefacts (e.g., source code).

3.1.2 Traceability in practice

With the definition set, we can build a more specific understanding of the thing, what it is build from, and the different flavours it comes with. First we define the concept of traceability, followed by what is known of its current academic and industrial setting.

To continue with terminology, Rasiman [21] visualised the relationships between the elements that potentially establishes traceability (based on the work by Gotel and Finkelstein). As described conceptually by Sundaram et al. [57], traceability is the notion of two artefacts that are linked to each other. If there is a link between them, one may speak of a trace. The artefact can be any unit of data, ranging from a requirement from the SRS to a class in the source code of some software application. Figure 3.1 shows an example of this (based on the made example), where there is a link established between two artefacts.

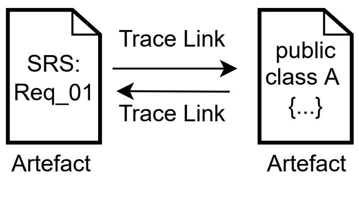


Figure 3.1: Example of a trace link between two artefacts. Adapted from [21].

In a survey by Winkler and Von Pilgrim [54], they identified three aspects of traceability (see Figure 3.2). *Forward* and *backward traceability* has been mentioned as one of the differentiating factors. *Forward traceability* concerns following the trace link from the source to the place where it transformed into a software artefact. *Backward traceability* is then the reversed form, where the (software) artefact is traced back to its original source. For the second aspect, there is *horizontal* versus *vertical traceability.* The difference between them is about a shift in the level of abstraction. If it concerns a link between artefacts of the same level (e.g., between SRS and interview notes), one speaks of horizontal traceability, whereas a link that spans across multiple abstraction levels (e.g., between SRS and a class in source code) is referred to as vertically. The final distinction the author makes is pre- and post-SRS traceability (originally suggested in the paper by Gotel and Finkelstein), and refers to the creation of traces between those two phases.

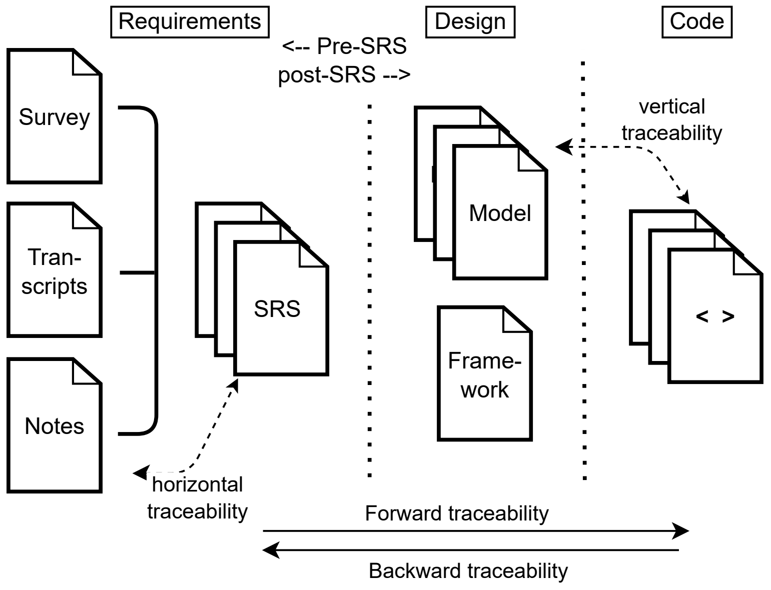


Figure 3.2: Traceability in multiple directions[1]. Adapted from [54, 60].

Manual and (semi-)automated traceability

According to Spanoudakis and Zisman [13], there are three approaches to traceability in software engineering, being manual, semi-automated, and automated traceability. As the authors identified in their research in 2005, the majority of the tools that aim to support both requirements engineering and traceability are limited in their actual support of traceability.

Manual traceability. Numerous approaches and techniques that are available for this purpose rely purely on manual input in terms of the creation of trace relationships. Independent of the approach one uses, trace links can only be used effectively when the link is strongly defined [61] (hereby leaving no gap for ambiguity). Where Spanoudakis and Zisman state that the creation of trace links manually is known to be error-prone and complex, Cleland-Huang et al. found that this is still a contemporary way of work almost a decade later [62]. Research by Neumüller and Grünbacher provided one possible clarification for this, which is that most commercially available tools only aim to support the management of trace links [22] rather than defining them.

Automated traceability. Where manual approaches require a human to create and manage trace links, automated traceability aims to cover that very part of the process. The creation of trace links, is one view where automated traceability can play its part. Another view is an after-the-fact approach, where a system would try to create the links afterwards, referred to as trace link recovery (TLR) [21]. Different techniques are available, and researched, to support this idea. Information retrieval (IR) techniques (e.g., probabilistic [63] and vector space [64]) have been used for this purpose for instance by determining the similarity between for instance requirement documents

[1]Listed elements are only used as an example, other elements may be appropriate as well given the context.

and source code [13]. Other approaches make use of semantics, like Latent Semantic Indexing) [65]. Semantics is used to identify syntactically related terms in documentation [66] as well. Studies also combined IR to the state-of-the-art in computer research, where they applied techniques from ML and DL. A recent study provides a case study where they applied a ML approach to recover trace links. A more elaborate reflection on this is presented at a later stage (see Section 3.3).

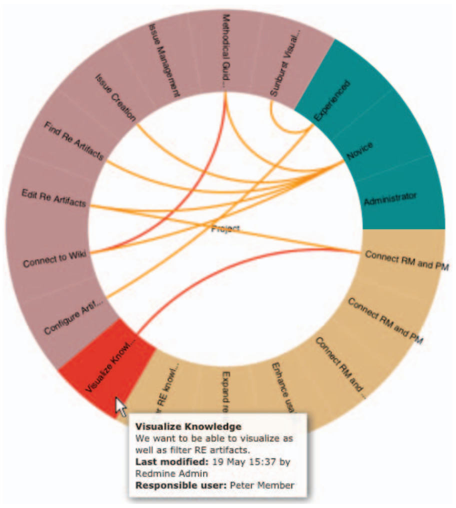
Semi-automated traceability. For semi-automated traceability, there is a trade-off made between the previous two approaches. Spanoudakis and Zisman [13] identified two families that relate to semi-automated traceability. There are *pre-defined link* approaches that apply user-defined rules to create the links between certain software artefacts. *Process-driven* approaches makes use of software that monitors the development of the software project to generate traceability relations. Sundaram et al. [57] adds a third family to the ones introduced by Spanoudakis. They speak of third (*collaborative*) approach that involves automated traceability tools to some extent. In this approach, the automatically created links have still to be evaluated by a human before they can be of use. Chen and Grundy [64] found techniques that they identify as semi-automated, other then the ones as previously introduced, although there is some overlap between the discussed approaches.

Studying the value behind (semi-)automated approaches to traceability has a long history through academics. To cite a relevant statement by Thayer and Dorfman from over 30 years: “automated traceability would be worth its weight in gold” [67]. Since then, the potential of automated tracing has been recognised by several, including the fact that it yet has to achieve the expectations as mentioned by Thayer and Dorfman [13, 57, 58]. Still, latest advantages in automatically discovering artefact relationships provide new opportunities, especially in the field of MDD [17]. Literature that elaborates specifically on traceability in MDD is presented in Section 3.2.

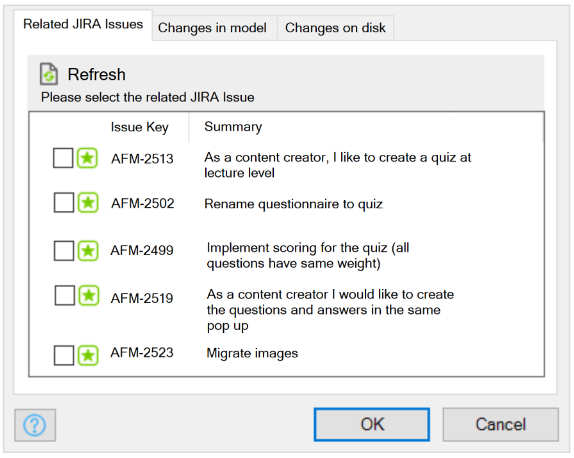
Visualisation of trace links
Neumüller and Grünbacher [22] implemented a traceability environment to automatically acquire traces for a software company, and their case study presented two relevant findings. First, the system allowed the developers to focus on utilising the trace links and second, an attractive visualisation of the trace links contribute to a wider adoption. The visual aspect of traceability has also been studied by others [62, 68]. They found that most commercially available tools use, amongst others, tabular and graph-based visualisations of traceability information. Simultaneously, they also identify that continuous growth of software projects (e.g., in size and complexity) demands the development of new visualisation tools to prevent visual clutter. According to the authors, at the moment of publishing, visualisation of traceability is still findings its way to the field (with tree visualisations or interactive approaches as recent examples).

Such visualisations are dependent on the context of its use case. Figure 3.3 shows two approaches on how trace links can be presented to a user, though both have a different utilisation. Merten, Jüppner and Delater proposed a view (see Figure 3.3a) that allows one to visually identify problems in the links across a project [68]. The view is interactive and can be filtered for the desired artefact(s) and can be helpful to correct false links. The authors state that a filter option is mandatory to prevent similar clutter issues as is the case for more traditional visualisation techniques. A second approach was opted in research by Rasiman. Here, two traceability scenarios were studied based on work by Rath et al. [47], with one of them focusing on a trace recommendation system. Rasiman provided a mock-up design for this scenario (see Figure 3.3b). This implementation has a different goal compared to the first approach that is discussed, where both could coexist. In case of trace recommendation, as a developer, you would be presented with open issues from the project issue tracking system that could be linked to the made changes inside the project. In this case, the goal is to maintain the trace links and keep it all up-to-date.

Traceability in the industry
Cleland-Huang et al. studied both industrial studies and questioned practitioners (IT project managers) to quantify the current state of practice [62]. They found that, from the questioned practitioners, a majority do not understand traceability and often see it as an: “unnecessary evil” or a “made up problem”. Their findings are similar to the findings by Neumüller and Grünbacher, who stated a lack of widespread adoption by the industry [22]. There are, however, industries (where safety critical systems are involved) where traceability is governed by standards. Cleland-Huang



(a) Netmap view for artefacts [68].



Please select the related JIRA Issue

Issue Key	Summary
AFM-2513	As a content creator, I like to create a quiz at lecture level
AFM-2502	Rename questionnaire to quiz
AFM-2499	Implement scoring for the quiz (all questions have same weight)
AFM-2519	As a content creator I would like to create the questions and answers in the same pop up
AFM-2523	Migrate images

(b) Mock-up of a trace recommendation system [21].

Figure 3.3: Approaches to traceability visualisation.

at al. found that this established a better understanding of the concept, but also acknowledged that this does not necessarily led to a wider or more efficient implementation of it. The authors do mention studies that successfully integrated traceability [69, 70], with the practitioners seeing the most value of tracing in relation to testing and regulatory compliance.

In the context of a case study, Torkar et al. [71] held interviews and distributed questionnaires at two companies to learn more about the industrial practices in traceability. Torkar et al. discovered differences between the two. One company has employees dedicated to RT (no clarification about the use of ST) and they see value in these activities. This is mostly due to improved customer satisfaction as the product development can be tracked more easily. The company also uses various tools, including an automated management tool. Though they describe the tool to be automated, the input comes from the other tools (for requirements elicitation), which are not labelled as automated. Each requirement has attributes that are used in tracing. The company did identify that they look for other solutions, as the maintenance cost of the automated tool is very high. The second company does not use ST either and only use traceability for requirements and testing phases. The company uses two separate tools for this, which are not linked. These processes are static, and though the company identified it as very efficient, it requires manual input to work with the tools.

Others conducted research to find and solve the barriers companies walk into when implementing traceability. In the study by Regan et al. [72] three viewpoints are discussed. First, they set apart *management* related *issues*. This includes aspects, like cost (related to tools and training) and traceability decay (decay of traces when not maintained). In essence, establishing traceability and maintaining it starts with proper management that is dedicated and see value in the cause. The authors propose solutions like developing a good user interface that reduces the time needed for training, or taking ownership. Second, they found *social issues*, that mainly concerns different viewpoints from stakeholders and internal politics (e.g., extra work because of documentation). To counter these issues, one could think of establishing better organisational policies and a better integration of traceability tasks. Finally, *technical issues* acts as a barrier. There are numerous tools available, but choosing the right one difficult and time consuming. Also, especially the tracing of non-functional requirements is experienced as highly complex. To tackle these technical issues, opted solutions are to have dedicated people or departments that can handle these technical aspects, and do not limit yourself to a one-solution-fits-all (as using a combination of techniques could deliver the required solution).

A more elaborate example of traceability in an industrial (MDD) setting is provided in an upcoming *section* by reviewing a case study at Mendix.

3.2 Model-Driven development

To review how traceability and MDD come together, first an understanding of MDD is established in Section 3.2.1. Next, we see what has been written about how traceability is making its way into MDD according to academic literature and see a recent industrial example of it (see Section 3.2.2).

3.2.1 Introduction to MDD

MDD is described by Brambilla, Cabot and Wimmer [28] as: “a development paradigm that uses models as the primary artefact of the development process. Usally, in MDD the implementation is (semi-)automatically generated from models”. To state a relevant phrase from the work by Wenzel [19]: “Models are no longer only the documentation or specification of the system. They *are* the system”.

When considering MDD in the application domain, three steps can be identified (see Figure 3.4). Following a clarification by Brambilla et al., the first step is to define the models. These models are then fed to mapping system that transforms the model into executable code based on transformation rules. The last level is where solutions are realised based on the automatic generated code. These steps are often executed iteratively, as the models are rarely designed in one go (or corrections are required) [19].

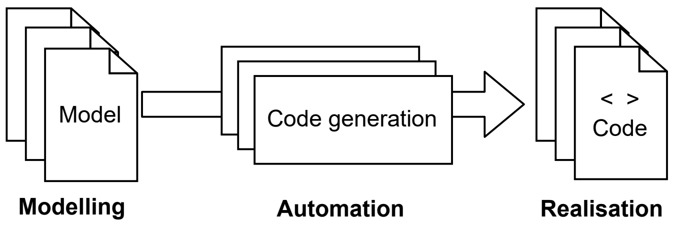


Figure 3.4: Application domain of MDD. Adapted from [28]

Models are a collection of classes, attributes, and the relations between them. An example of such a model as used in MDD is displayed in Figure 3.5. Without going to much into detail, the model includes three classes (i.e., Session, Feedback, and Attendee) with the relation (and cardinality) in between. Each class has a selection of attributes related to that class, including the data type of that specific attribute. During the transformation, all these elements are used to transform the model into working code. In practice this is an iterative process, where models are modified or extended based on evaluations on the realised code.

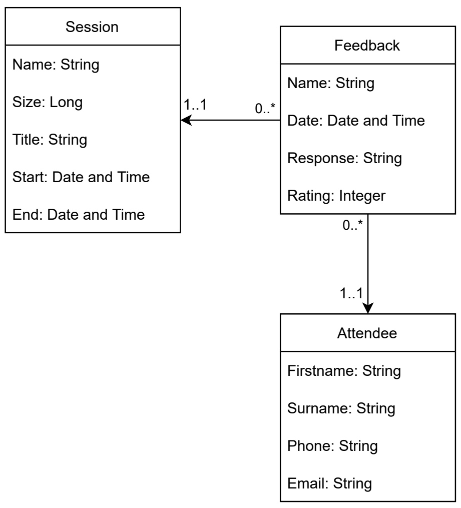


Figure 3.5: Example of a model as used in MDD. Adapted from [28, 73]

3.2.2 Traceability in MDD

Traceability has been researched for some time, but only recently has it became subject of research in the field of MDD [54].

As described by Santiago et al., automated traceability approaches are especially suited to facilitate traceability in MDD. He arguments that, since there is already a large automation aspect in MDD by means of model transformation, it becomes more feasible to automatically generate traces [17]. In their research, they acknowledge that one approach would be to derive relationships at a meta-model level, so-called traceability meta-models. This idea has also been reviewed by others [19, 54, 74], where they focused on model-to-model traceability. During the development of a model, when a model element is created, it is assigned an identifier. This identifier is then tracked throughout the project, including revisions and relations to other classes.

Wenzel [19] identified various approaches in literature where class models are linked to source code. One approach essentially uses an algorithm to identify elements that changed (from model to source code and vice versa) and link them. These elements were then marked and had to be reevaluated by someone (i.e., semi-automated). As upside, since the approach is based on XML representation for models and source code, it allows nearly any kind of application that can be transformed to XML to use this. Wenzel also suggests the use of techniques as discovered *earlier*, such as IR-based and similarity-based approaches.

Traceability in an industrial setting
A case study that was conducted at Mendix [21], a low-code software platform developer, provided insight in present-day example of traceability used in the industry. The company develops their own platform (Mendix Studio) that allows users to develop their own applications in a low-code manner (that utilises the principles of MDD). The company's own developers create applications with the use of this software as well. According to the explanation by Rasiman, when developing a software application, a developer commits its made changes to the application model towards a team server. Developers add certain information to that commit, like what has been added or modified. They also are required to add the Jira issue identifier (ID) to the commit message. Atlassian Jira is a tool the company uses that allows them to keep track of their development progress through user stories and bug reports. By providing the issue ID, developers establish a trace link between the commit and the Jira issue. This makes the current practice of traceability to be labelled as a semi-automated approach. As a result of the case study, Rasiman came with a proposal to go from a semi-automated approach to an automated approach, which is elaborated on in Section 3.3.

3.3 Baseline: Automated Traceability from Model Changes to Issues

In this section, the proposed automated approach by Rasiman is reviewed in more detail. The goal of his research was to recover trace links with the use of ML for two traceability scenarios; trace maintenance and trace recommendation. To find the best model for these scenarios, an experiment was conducted to see what combinations of variables lead to the most optimal solution. For his research, Mendix provided him with data sets on four software projects that they developed in-house. For each project, one data set that included Jira issues and one data set with revision history was received.

In Section 3.3.1, some more context about his approach is described (e.g., data, techniques, and metrics). After that we go into details about the results obtained through the study (see Section 3.3.2). This section also includes a comparison with other studies in this field allows us to finalise a description of the state-of-the-art in TLR with the use of ML.

3.3.1 Case description

As priory mentioned, Rasiman identified two scenarios based on the work by Rath et al [47]. The recommendation scenario asks, for example, for a presentation of a list to a developer whenever a commit is pushed, to give a possible implementation. The system then provides options with related issues to assign the commit to the related issue. The maintenance scenario asks the system to link the different artefacts (i.e., Jira issues and revision history) to each other in a completely automated manner.

Variables
Data sets. For each project, two data sets were obtained. The Jira data sets included details about the Jira issues which included aspects, like a summary (description of the issue), issue key (reference ID), assignee (person responsible for implementing the Jira issue) and dates (i.e., created, resolved and updated). The revision history included aspects as revision number, author (committee), and a log (i.e., commit message). *Features.* To represent candidate trace links between the artefacts, Rasiman engineered 131 features across 4 so-called feature families combined from multiple academic sources. The first family are **process-related** features (n=4), relating to the authors of artefacts or timestamps. Then, **document statistics**, use seven features that focuses on document characteristics (i.e., number of terms or overlap between artefacts). **IR** features (n=18) apply IR techniques to calculate semantic similarity between artefacts. Finally, for **query quality** (n=101), we make a distinction between thee sub families (specificity, similarity, and term relatedness). The trace links were then subjected to different rebalancing strategies[2] (none, over, under, and 5050) to deal with imbalanced data (as result of the constructed trace links). *Classifiers.* Based on previous research, three classifiers were identified as possible candidates to experiment with (Random Forests, XGBoost and LightGBM).

The pipeline
The experiment is performed through a pipeline. First, the two raw data files are provided to the program. These files are then cleaned to make them of better use, which includes activities as stemming of textual attributes and the creation of corpora. What follows is pre-processing the files. The Cartesian product is calculated (i.e., multiplying all possible combinations of data points between the two data sets) and based on that, all features are populated. The output of this part is one file that contains all output values of the calculated features. This file is then subjected to the next activity, that is about running multiple evaluations on the different configurations (where a configuration refers to a combination of a classifier and a rebalancing strategy).

The performance on different configurations was determined based on standardised metrics (i.e., precision and recall). Based on literature, for the trace maintenance and trace recommendation scenarios, $F_{0.5}$ and F_2 were the chosen metrics respectively. This means that the configuration that scored the highest was opted as the configuration to go with, for each scenario. The ideas behind the chosen metrics is further explained in Section 3.4.1.

3.3.2 Outcome

Based on the experiments Rasiman did, two configurations were selected as optimal considering the available options. For trace recommendation, this resulted in an F_2-score of 73.48% for the configuration of the LightGBM classifier on a 5050 rebalancing strategy. For trace maintenance, a $F_{0.5}$-score of 77.32% was obtained for the XGBoost classifier without applying a rebalancing strategy.

One outcome of these experiments is for evaluated scenarios, at least one of the proposed configurations did outperform the classifier (i.e., Random Forests) that was used in the work by Rath et al. Some key differences between the studies should be noted. For one, Rath et al. only included 18 features across two feature families, where Rasiman included 154 in his original work. Since the studies are completely different (e.g., MDD vs. non-MDD projects and 154 features vs. 18 features), it can be interesting to do some high-level comparison. The studies do show similar performance, and although the data sets that were used can be very different on a project-level, it is interesting Rath et al. is able to reach such performance with the number of features that they use compared to Rasiman.

[2]Details on the strategies and other aspects that are not discussed now can be found in the original work [21]

3.4 Improving on ML classification

The foundation of the tool, resulting from the study by Rasiman, is based on ML techniques. In this section, we explore literature that offers opportunities in this field specifically that potentially allows us to make further improvements to the tracing tool as delivered by the author. These improvements must be compatible with the optimal configurations as concluded in the original work. This means that the focus is on making improvements in terms of trace maintenance (LightGBM - 5050 rebalancing) and trace recommendation (XGBoost - no rebalancing).

To explore the opportunities, first a brief introduction is given to ML classification to understand the basics of it (see Section 3.4.1). Next, Section 3.4.2 presents various options that could potentially deliver the desired improvements.

3.4.1 Introduction to ML classification

Machine learning has many applications nowadays and is often the go to as it comes to high dimensional data [45]. Novaković et al. describes ML as: "the discipline that studies the generalization and construction and analysis of algorithms that can generalize". Others describe ML as the study of computational algorithms that are automatically enhanced and its application goes from automatically detecting features in images to speech recognition [75]. ML is often applied on cases where traditional algorithms are difficult or impractical to implement. Research speaks of two distinct ML problems, being regression and classification [45, 75, 76] with the latter being the focus of this review.

Supervised learning
The algorithms that are under review for this study (i.e., decision trees) can be categorised as supervised learning algorithms. The primary goal of such algorithms, as stated by Charbuty and Abdulazeez [75], is to: "learn a model that creates the same labeling preferably for the data offered and popularizes well on unseen data". Supervised ML is the process of learning a set of rules that come from instances of a data set (provided through a training set), according to Kotsiantis [77]. He describes six steps of such a process as a practical example (see Figure 3.6).

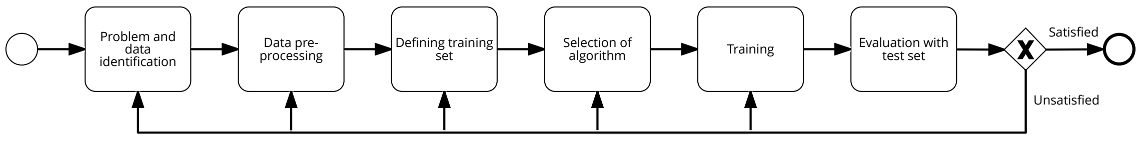


Figure 3.6: Process visualisation of supervised ML. Adapted from [77]

As a brief summary, the first step the author describes is the identification of the problem and the data that can be used to solve that problem. Next, the data needs to be pre-processed. This includes handling of missing data and transforming data attributes in a form that can be worked with. The data is then split into a training and test set (for the study by Rasiman, this resulted in a 80:20 split). An algorithm, one that is appropriate for the problem we aim to solve, is selected (e.g., classification or regression) which is then trained by using the training set. To test if the algorithm produces satisfying results, an evaluation is run with the use of the test set. If the performance is insufficient, one or multiple steps need to be redone until satisfactory results before the classifier is ready to be used.

Decision trees relate to a family of non-parametric classification algorithms, that classify instances by making a decision based on an attribute's value [77]. Figure 3.7 presents a basic representation of a decision tree, that has a flowchart-like structure. The figure shows four nodes (attributes) and even more branches. When we reach a point where a node does not again makes a split, we reach the leaf of the tree (which is the end point). The leaf represents then a class label, which is the decision that is made based on all the considered features.

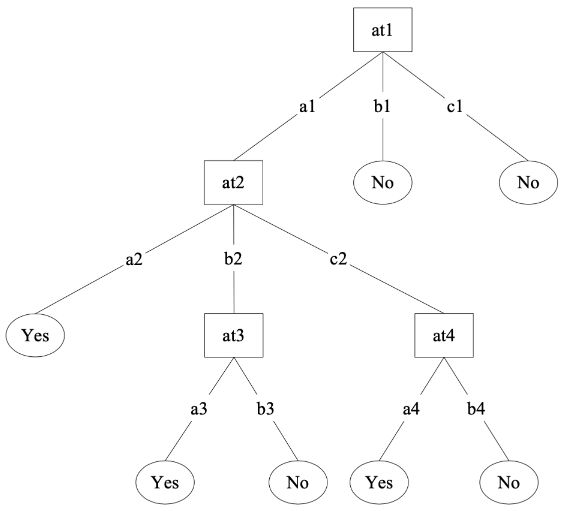


Figure 3.7: Basic representation of decision tree [77]

Evaluation of classification models
The evaluation of models refer to the degree a suggested classification by a model corresponds to the actual classification [78]. However, there are different methods that can be used to measure the performance of such models (e.g., accuracy, metrics based on precision and recall, and DiffAR) [21, 47, 78]. As identified by Shin et al. [79], metrics related to precision and recall are often the chosen metrics for model evaluation purposes. Based on the metrics used by others, the focus will be on the metrics that are a trade-off between precision and recall.

Before the specifics of these metrics are explained, first we break down the two elements that are used in the trade-off based on the explanation of both Rasiman and Novaković. To speak in terms of trace links, **precision** is defined as the percentage of valid retrieved trace links (see Equation 3.1). **Recall** is then denoted as the number of valid retrieved trace links across the total number of valid trace links present, meaning how much recalled from all the valid traces that are there (see Equation 3.2).

$$Precision = \frac{correct\ retrieved\ trace\ links}{correct\ retrieved\ trace\ links + incorrect\ retrieved\ trace\ links} \tag{3.1}$$

$$Recall = \frac{correct\ retrieved\ trace\ links}{correct\ retrieved\ trace\ links + correct\ unretrieved\ trace\ links} \tag{3.2}$$

In the Mendix case description (see Section 3.3.1), we mentioned the terms $F_{0.5}$ and F_2. These F-measures refer to the trade-off between precision and recall, where the number states the importance of recall over the precision. For instance, for F_2, the recall is more important compared to precision (with $F_{0.5}$ vice versa). One could, if justified, also use another balance between precision/recall (dependent on the use case). The general form of the F-measure (defined as F_β) is presented in Equation 3.3. The equations for $F_{0.5}$ and F_2 can be found in Equation 3.4 and 3.5 respectively. As explained by both Rath et al. and Rasiman, these metrics were chosen based on the two scenarios that were studied. In the case of trace maintenance, one would require more precision because of the absence of human validation, where recall would be more important for trace recommendation (as it simply provides various potential options to a user who will pick the correct one).

$$F_\beta = \frac{(1+\beta^2)\ \times\ Precision\ \times\ Recall}{(\beta^2\ \times\ Precision)\ +\ Recall} \tag{3.3}$$

$$F_{0.5} = \frac{(1+0.5^2)\ \times\ Precision\ \times\ Recall}{(0.5^2\ \times\ Precision)\ +\ Recall} \tag{3.4}$$

$$F_2 = \frac{(1+2^2)\ \times\ Precision\ \times\ Recall}{(2^2\ \times\ Precision)\ +\ Recall} \tag{3.5}$$

Novaković also addresses another way to test the performance of a classifier, which are Receiver Operating Characteristics (ROC) graphs. He cites one author [80] who explains the graph as follows: "ROC graphs are two-dimensional graphs in which the TP (True Positive) rate is plotted on the Y axis and the FP (False Positive) rate is plotted on the X axis". Such graphs allows us to visually compare the characteristics of different classifiers. When plotting discrete classifiers (resulting in a class label, being either valid or non-valid), the output is just a single point. Figure 3.8 shows an example of an ROC graph including three discrete classifiers. In the graph, classifiers I, II, and III are plotted. Classifier I shows an almost perfect classification. Classifier II performs way worse, reported many false positives. As explained by Novaković, a classifier that is located at the diagonal has no information about its class. A classifier below the diagonal does have information, but uses it incorrectly.

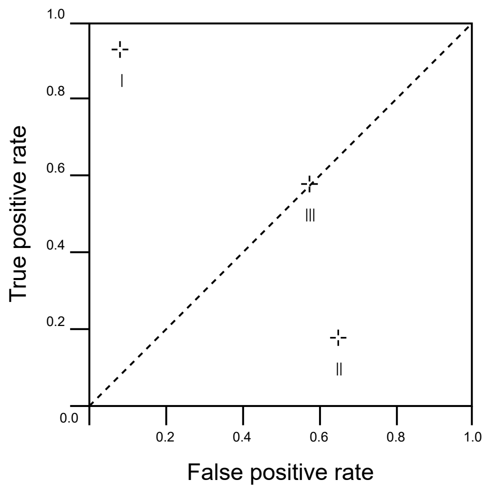


Figure 3.8: Example of ROC graph showing three classifiers (I, II, III). Adapted from [80]

What evaluation method is the most appropriate depends on the data set you use for the classification [81]. In case of a heavily imbalanced data set or you care mostly about the positive class (in case of binary classification), it is recommended to use F-measures over ROC graphs.

3.4.2 On the way to improvement

The literature found for this review provided two possible solutions for model improvements. First, optimisations through parameter tuning is explained, which is then followed by approaches to systematically selecting features from a feature space.

Parameter tuning
In the search of improving classification models, a first step could be tuning a classifiers' parameters. Often, but for one classifier of greater impact then for the other, one could change the default parameters of the algorithm. Studies identified parameter tuning as something that could be needed to get the desired performance from a model [21, 45, 76, 77, 82]. Though they mention the option to make improvements, most of these studies do not go into detail to quantify this effect.

Rasiman [21] did experiments with various parameter tuning for the previously mentioned scenarios he studied. For both scenarios, he tested the effect on the performance of five parameters (the parameters are not equivalent between the scenarios). These parameters came from non-academic sources and included aspects as setting the maximum number of leaves and the learning rate. However, based on statistical analyses, he did not find a significant effect on the performance scores.

Feature selection
The number of features that are used in ML models is increasing and is leading to problems with high dimensional data [45]. Various studies have identified problems that may occur that relates to the use of many features in ML models [45, 82, 83, 84, 85] with many proposals to tackle the issue.

Mills et al.[82] states that each introduced feature includes the possibility of introducing errors and increase both computational time and the chance of overfitting. It is also possible to include features that are correlated or redundant [45].

To decrease the number of features, the cited authors all steer towards (automated) feature selection. The term is quite self-explanatory, but it refers to applying techniques to (automatically or not) reduce the number of features used in a model. Jiarpakdee et al. [83] makes a distinction between three feature selection types. *Filter-based* techniques search for subsets based on evaluation criteria. This includes techniques based on correlation or information gain (i.e., selecting features from a feature set that score (based on correlation or gain) above or below a certain threshold). This selection type is also independent of the learning process. *Wrapper-based* techniques use classification techniques to come to an optimal feature subset (again, based on evaluation criteria). Often, candidate subsets are generated which are then subjected to a classification model. The candidate subset that shows the best performance is then opted as the feature subset to be of use. Recursive Feature Elimination (RFE) is a wrapper-based technique, where the total feature set is reduced feature-by-feature based on an importance score. Jiarpakdee et al. make the distinction between the two as filter-based approaches being low cost and more widely used compared to wrapper-based approaches (more computationally intensive [46]). Finally, there are combinations of the two types, called *hybrid-based techniques*. Filter-based and wrapper-based techniques are then combined to benefit from the advantages each type has. One example the authors provide is that the initial feature subsets are created (e.g., correlation based) using filter methods and then a classifier is used to identify the optimal set of the subset that was created using the filter method.

It should be noted that the techniques described by Jiarpakdee et al. often evaluate on a feature-level (e.g., the feature that scores the highest is added to a subset). Peng, Long, and Ding [85] mention in their study that: "individually good features do not necessarily lead to good classification performance". In a previous study [46], they came up with a filter method (minimum redundancy - maximum relevance (mRMR)) where they want to find features that (1) maximise the dissimilarity between features and (2) maximise the relevance to the target. In their follow-up study, they expanded the feature selection approach of mRMR by enhancing it with a wrapper method. As a result, the wrapper method can be used more effectively (which was the main goal of the research).

3.4.3 A trace link recovery pipeline

In their study towards creating a ML pipeline as part of a smart electrical grid, Krishnadas and Kiprakis [86] describe their approach from problem identification towards the deployment of the ML models in a production environment. The activities they describe, up to the point of the creation of a ML pipeline, follow similar steps (e.g., algorithm selection and model evaluation) as by the study of both Rasiman [21] and Rath et al. [47]. For deployment, they consider two ML workflows that distinguishes two sets of tasks that are both necessary to be in place. One set regards to data pre-processing, training, and forecasting, which should be performed on fixed intervals. These intervals depend on the computational power that is available and economically viable. The other refers to tasks (model selection, evaluation, and redeployment) that need to be executed only when the performance of the model declines. The execution of these workflows are managed through workflow management systems, where they propose two options.

Sugimura and Hartl [87] designed a four stage system architecture (see Figure 3.9), which again follows a similar structure as previously described. They start with a Data layer that is responsible for retrieving the data to be fed to the system. The features layer then generates features from the data provided from the previous layer. The scoring layer is assigned to perform necessary data transformations, training the classifier and defining the model. Finally, the evaluation layer is there to monitor the model to see if it still achieves the required performance.

Olson and Moore [88] propose a tool called Tree-based Pipeline Optimization Tool (TPOT) in their research that can be used to optimise ML pipelines. For this purpose, they use a method called genetic programming that is used to evolve tree-based pipelines. One possible implication in their approach however is that it evaluates pipelines based solely on accuracy, which can be misleading. As explained by Zeya [89], accuracy is the: "the portion of correct predictions generated by a model".

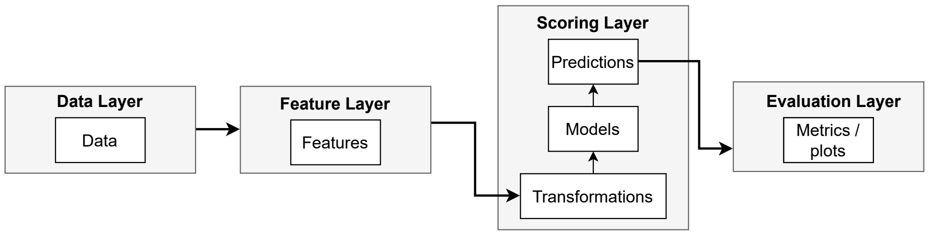


Figure 3.9: ML pipeline architecture. Adapted from [87]

So, accuracy only considers the correct positives and the correct negatives. This means that, if a model generates the same amount of correct positives and false positives, accuracy does not take the false positives into account. When solely considering accuracy, you could end up with a model that does not classify the classes correctly. If one would consider such an optimisation tool as by Olson and Moore, more aspects should be considered to properly evaluate a pipeline.

3.5 Lessons learned

Traceability has a rich academic history in requirement engineering and software, where many see value in (semi-)automated traceability approaches. We have accepted the definition for traceability as: "the ability to describe and follow the life of an artifact developed during the software lifecycle in both forward and backward direction". Artefacts can be requirements written down in natural text or source code. If two artefacts are connected, you can speak of a trace link between the two artefacts. Manual traceability approaches are commercially available with some variations (i.e., manual management and/or creation of traces). This type of traceability is known to be error-prone and complex. Another approach is automated traceability, and though recent studies show techniques that could substantiate to this automation, no industrial applications were found in literature. An intersection of the two approaches is semi-automated traceability, where a human is still required to intervene (often to validate) in the process. The adoption of traceability practices can benefit of a proper visualisation of trace links. Though traditional visualisation techniques often clutter, recent proposals try to prevent this through filter options and dynamic approaches.

Model-Driven development (MDD) has been identified as being suitable to facilitate automated traceability, because of parts of MDD already being automated. In MDD, applications are developed by the creation of models, that are then automatically transformed to source code. Researchers that studied traceability in MDD found various options to apply automated traceability in the field, though it is still in an academic context. With a case study, a (semi-)automated traceability approach has been studied in a low-code development environment. This study applied ML techniques to establish a (semi-)automated approach to traceability in MDD. Two scenarios were studied (i.e., trace maintenance and trace recommendation), which led to two optimal configurations (including classifier and data rebalancing strategies) for those specific scenarios. Though the results looked promising, optimising the tool may contribute to industrial applications.

The ML techniques used in the previously mentioned case study fall under the category of supervised learning. That means that a model is trained with data and evaluated to see if the model is able to correctly assign correct class labels. The evaluation of such models is often done through metrics based on precision and recall, though a Receiver Operating Characteristics (ROC) has been proven to be effective in model evaluation as well. Literature provides two options for model improvements. The first is through parameter tuning, which allows us to fine tune certain parameters that could outperform the default settings of the classifier. Secondly, systematically selecting features from a defined feature space could help to boost performance, improve the interpretability of models, and reduce computational complexity. Tools that helps to optimise ML pipelines are available, but would need improved evaluation techniques to ensure the models are correctly judged on their performance.

Chapter 4

Secondary analysis

In this chapter, we present the results of a secondary analysis on software projects that originated from two different software development paradigms. The chapter follows the structure as outlined in Section 2.2.3, where the analysis' approach has been described. As a result, Section 4.1 presents our first step to thoroughly understand the data, including activities such as a data quality check (as described in Section 2.2.2). Next, we define variables in the data and visualise them using descriptive statistics (see Section 4.2), which helps us to better frame the data that we are working with. The chapter is concluded by a short summary with all the valuable lessons that we have learned (see Section 4.3).

4.1 Data description

To describe the data, we first explore the data of both sources (see Section 4.1.1) and make a selection of which software projects to include in this analysis (see Section 4.1.2). In Section 4.1.3, we briefly describe the data sets that were selected in the previous section.

4.1.1 Data sources

Our first step is to study the data we gathered from both sources (i.e., Mendix and SEOSS33 [39]). To do so, we explore each source and describe the software projects present in each source.

MDD
The data set from Mendix consist of 7 software projects. As described in the literature review (see Section 3.3.1), the data consist of Jira issues and revision history for each project. The Jira issues and revision history are delivered as separate files, as a Microsoft Excel and .txt file respectively. An outline of the Mendix data set can be found in Appendix G1. Table G1 displays for each software project the number of months the project data encompasses, submitted Jira issues, revisions, and how many revisions have been traced (or not traced) to Jira issues. Note that, for *Control* and *Learn*, the table displays complete projects as well as partitions that are used in our experiments in Chapter 5. For this chapter, we want to capture and characterise the entirety of a project and therefore will consider the complete files only and ignore the partitions.

Overall, the table shows that revisions are often traced to at least one Jira issue, though there are some variations to discover. We see that revisions are, with the exception of *Service* and *Store*, traced by at least 70% for all projects. *Store* is an extreme case with only 29% of its traces linked. However, the project also has the longest time period (starting in 2018) compared to the other projects (which all started after that year). This could be an indication that establishing traces became more important during development after that period. The time period does vary significant, with some projects providing records of only three months, while others well over a year.

The complete structure of the data set can be found in Figure G1.

Non-MDD
As a representative data set for development projects that do not employ MDD, we chose the SEOSS33 [39]. The data set includes 33 open source software projects compared to the seven MDD projects that was received. Each software project is covered by one database file (i.e., SQLite), with tables included for both the issue tracking (by Jira issues) and revision history (through Git, compared to the Subversion system used by Mendix). Similar to the table provided for the MDD data set, an outline of the non-MDD data set is provided in Table G2, with an structured overview displayed in Figure G2.

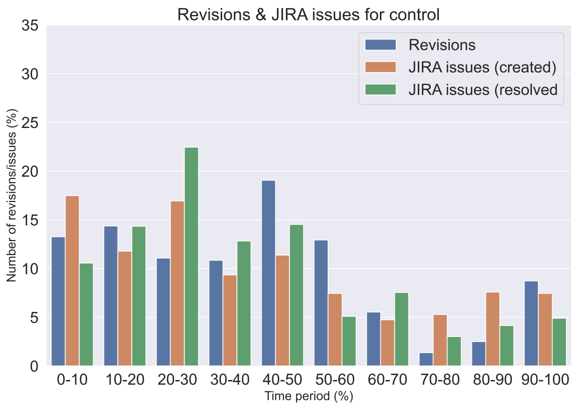
First, we see a large difference in the time periods compared to the MDD projects, with an average period of 133 months. This logically allows for more issues and revisions to be included in the projects. Further more, we see quite some differences in terms of traced revisions. Only a few projects (*Hadoop, Hive*, and *Zookeeper*) have a trace percentage that is similar to the trace rates as recorded for the projects from Mendix.

4.1.2 Project selection

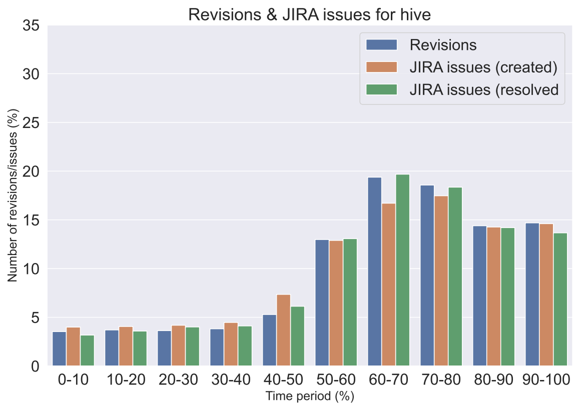
We will be comparing the different software projects from both sources. To do so, we will use all seven projects from the Mendix data set. For the projects coming from SEOSS33, we take a sample so that we match the same number of projects as for the Mendix data set. A simple Python script is used to take this random sample, which can be found in the *Secondary Analysis* folder in the online Appendix [33]. This resulted in the following projects to be included for the continuation of this analysis: *Cassandra, Hadoop, Hive, JBehave, Lucene, Resteasy,* and *Weld.* As said, all the Mendix projects are included in the analysis, which are all listed in Table G1.

4.1.3 Data exploration

To get an idea on how the revisions and Jira issues are distributed amongst the selected projects, we briefly explore the data in those aspects. First, we review MDD projects, followed by non-MDD projects. To get an idea of the distribution of revisions and Jira issues across these projects, bar plots have been created. Figure 4.1 displays a bar plot for both an MDD and non-MDD project, showing on the vertical axis, the percentage in relation to the sum of revisions, created Jira issues, and resolved Jira issues, projected over the projects time span expressed as percentage (at a 10% interval). In these two plots, we can identify two different developments in terms of active participation on the project (in terms of revisions and created/resolved Jira issues). The bar plots for all other MDD projects can be found in Figure H1, with the non-MDD projects displayed in Figure H2.



(a) Distribution for an MDD project (*Control*)



(b) Distribution for a non-MDD project (*Hive*)

Figure 4.1: A comparison of the distribution of revisions and Jira issues between an MDD project and non-MDD project.

MDD projects. The graphs (see Figure G1) show different distributions for each of the three traces (i.e., commits, Jira created, and Jira resolved). Projects *Control, Learn, Portfolio*, and *Service* show are quite clean process, with some busier periods on occasions (e.g, halfway through the *Learn* project). Throughout the projects, there are always commits being made and Jira issues created and resolved. For other projects, we see some more fluctuations during the development. For instance, the project *Company*, starts with two active periods of development with gaps in between. The reason behind this can be numerous things (e.g., extended functionalities), which are outside the scope of this analysis. In the final stages of the time span that is covered by the data, it seems that the developers do some fixes as we see some revisions and resolved Jira issues. *Data* also shows another pattern, where we have a small initial project start that is continued months later (as Table G1 shows an absolute time span of the project of only 13 months). For *Store*, we see again a different pattern. This time, we see two active periods, in terms of commits, in the first half of the project while almost no Jira issues are being created or resolved. This is in contrast to the second half, which shows are more continuous process as we saw for the first four projects that were discussed in this paragraph.

Non-MDD projects. When scanning over the bar plots as displayed in Figure H2, we see three different flows where projects are either increasing or decreasing in productivity over time, or we see a flatter distribution throughout its process. No matter the distribution, revisions and Jira issues (created and resolved) are often well represented in every period of development. For two projects (*Hadoop* and *JBehave*), we see a deviating start of the project in comparison to the others. For these two projects, the first period recorded many created and resolved Jira issues in contrast to the number of revisions. Such a pattern is not visible in any period for other projects that are visualised in the figure.

On this first, high-level, view on the projects, we see some clear differences between the MDD and non-MDD projects. Where the non-MDD projects seem to follow a more continuous process, the MDD projects show, while also recording more stable processes, more fluctuating periods of active development. Also, these projects do not show a clear flow of development as we saw for the non-MDD projects, where projects either increase or decrease in the amount of work being done. These clear flows we see for the non-MDD projects might be explained when we consider the absolute time span of the projects. Since the average time span of the non-MDD projects are 133 months, this could flatten or filter out diverging periods compared to the short development time of 14 months for the MDD projects. A commonality of all analysed projects is that we do not know if the development on these projects all started and ended within the coverage of the data sets. We do not know what happened before or what happened after. Because of this, we cannot draw decisive conclusions at this point.

4.2 Variables selection & visualisation

After this first exploration, we we select variables from existing or derivable attributes from the data that might be of interest. In Section 4.2.1, we will elaborate on the size of a revision. After that, we briefly look into the three different types of commits being made (see Section 4.2.2). Finally, in Section 4.2.3, we take a deeper dive into the data to the level of a single commit and categorise the commits' log messages on size and extract frequently used terms.

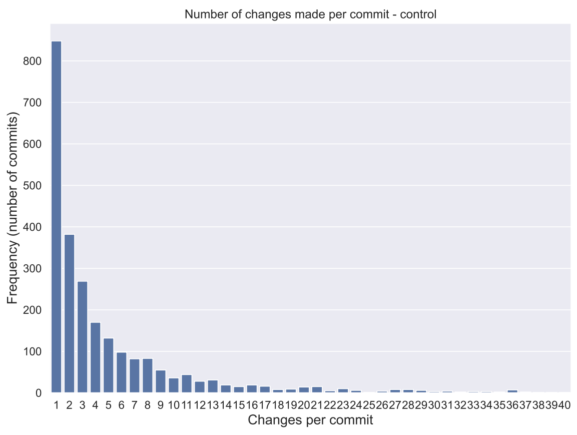
4.2.1 Revision size

The number of changes that are made in a single commit helps us to quantify the manner or followed structure during software development. Size differences in commits can relate to numerous reasons. Larger commits, for instance, are likely to relate to code management activities, while smaller commits are more likely to be produced due to development activities [90]. By visualising this, the intent is to discover differences in these aspects between (non-)low code applications.

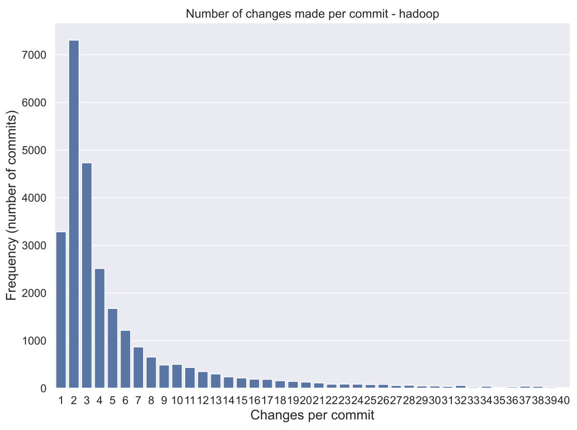
Figure 4.2 displays two histograms, one MDD project (see Figure 4.2a) and one non-MDD project (see Figure 4.2b). Histograms that cover all projects can be found in Appendix I. Figure 4.2a shows that commits often contain only one or a few revisions per commit, resulting in a negative exponential looking trend. This trend is shared amongst all projects, both MDD and non-MDD, with the exception for *Hadoop* (as displayed in Figure 4.2b).

If we want to have a deeper understanding of the size of a revision, we need a more extended view on each project. Figure 4.3 presents the revision size on each project as Box and Whisker plots. These plots are the first step to categorise the commits.

First, we briefly recall how such plots should be interpreted. The size of the box (coloured part) indicates the interquartile range (IQR), which relates to 50% of the data values. One could say, the size of the box shows how centered the values are (a smaller IQR means values are centered, which could indicate being more reliable scores). The whiskers (top and bottom bars) both indicate the outer ranges (both 25%) with the horizontal bards as minimum and maximum values. The bar inside the middle of the IQR represents the median. A median in the lower-section of the IQR



(a) Revisions sizes on MDD project (*Control*)



(b) Revisions sizes on non-MDD project (*Hadoop*)

Figure 4.2: A comparison between MDD and non-MDD projects in terms of revision sizes.

means that the distribution is positively skewed, with a negatively skewed distribution when a median is in the upper-section. Data points above or below whiskers can be identified as outliers (which are left out of Figure 4.3 for readability purposes).

In the first place, the plots in Figure 4.3 do not show a clear distinction between MDD projects compared to their non-MDD counterparts. We do see that most projects coming from MDD are located more to the second half of the Figure, meaning that they often include more changes in a revision compared to non-MDD developed projects. However, the coloured plots are still too mixed to make concrete conclusions at this point.

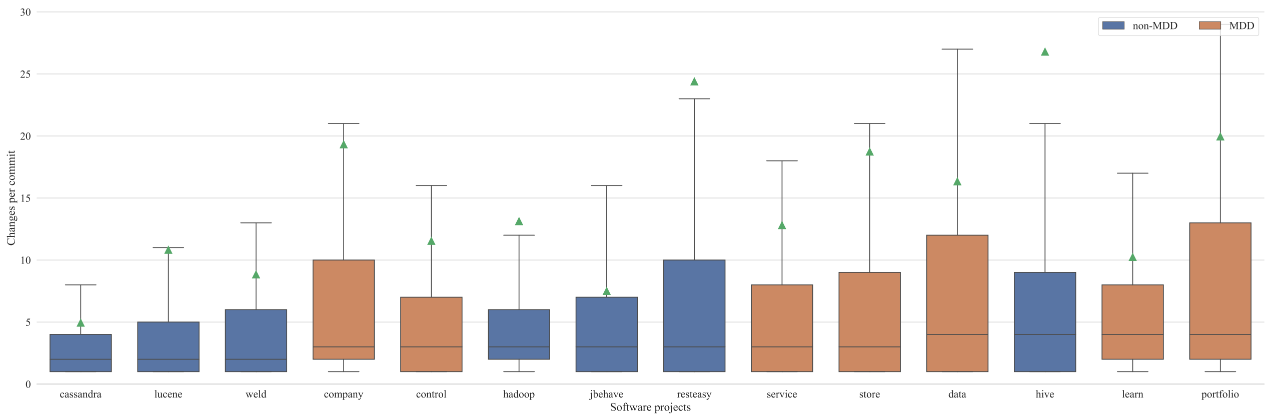


Figure 4.3: Box and Whisker plots presenting the distribution of the number of changes per commit for all (non-)MDD software projects. The plots are ordered ascendingly by their median values. The average number of changes that are made to a project is indicated by a green triangle (▲).

We can expend our insights by first categorise the commits based on their size. For this, we borrow a scheme from a study by Alali, Kagdi & Maletic [43]. In their research, they use a 5-point scheme that allows them to categorise the commits into five size regions based the projects own measures. Their scheme uses the following data points: (1) the minimum value (Q_0), (2) the lower quartile (Q_1), (3) the median (Q_2), (4) the upper quartile (Q_3), and (5) the maximum value (Q_4). With these points, the different size regions are defined as such;

1. $[Q_0,\ Q_1)$
2. $[Q_1,\ Q_3)$
3. $[Q_3,\ Q_3 + 1.5 \times IQR)$
4. $[Q_3 + 1.5 \times IQR,\ Q_3 + 3 \times IQR)$
5. $[Q_3 + 3 \times IQR,\ Q_4]$

Table K displays, for each (non-)MDD project, a categorisation of the commit sizes based on the prior mentioned scheme. This categorisation then allows us to understand what types of changes are usually included in what size category and see if there are certain differences between MDD and non-MDD projects. When we visualise the data from Table K and compare the aggregated commit category sizes (see Figure 4.4), we see that both share a similar distribution across the category sizes.

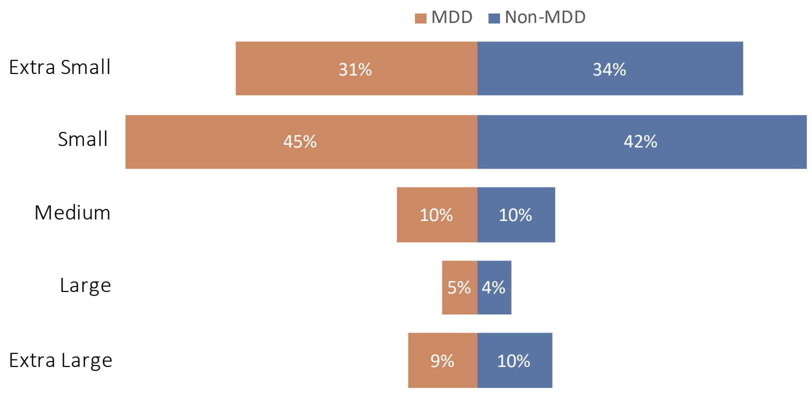


Figure 4.4: Visualisation of aggregated ratio's of commit sizes categories for all (non-)MDD projects.

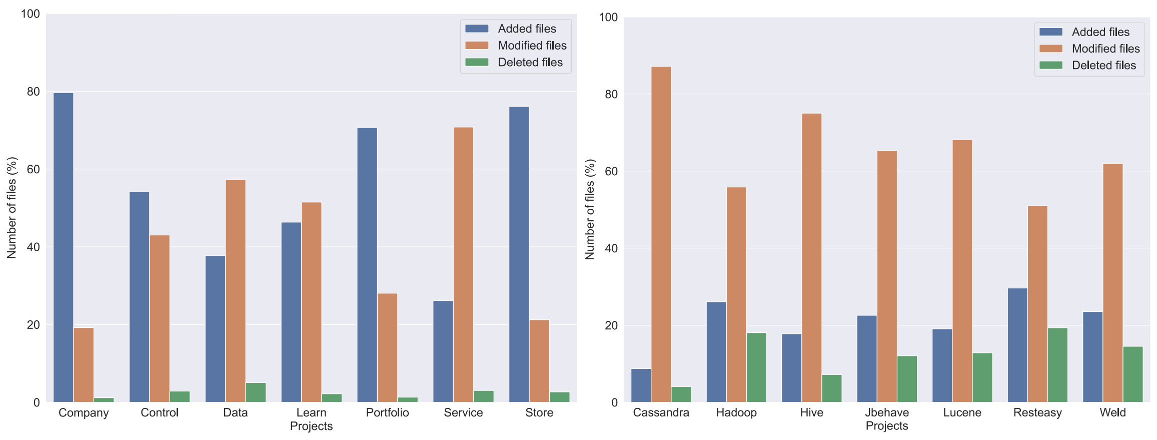
Making a comparison of revision sizes (as in Figure 4.4) does show us something about how the commits are distributed across the size categories, however not so much about revisions characteristics. What is of interest is to see if we can find differences or commonalities between MDD and non-MDD when we decompose each commit message and categorise them according to their commit size. To do so, we follow the steps as described by Alali, Kagdi & Maletic (see Section 4.2.3).

4.2.2 Revision type

When we look at the commits being made to each project and make a distinction in the type of revision that is made, we can see some differences between the majority of the MDD and non-MDD projects. Figure 4.5 displays the different types of changes that are made to a software project, where we make a comparison between MDD projects (see Figure 4.5a) and non-MDD projects (see Figure 4.5b).

When we consider the number of **added** files, the plots in Figure 4.5 show there are difference between MDD and non-MDD projects. In terms of MDD projects, for six out of seven, most of the commits are about adding new files to the project. We can see a distinction for *Portfolio*, that shows a similar distribution as the non-MDD projects where every time, way more files are **modified** instead of added. We also see more files being **deleted** in non-MDD projects compared to MDD, though these always a minority in contrast to the other two traces.

Of course, we need to take into account that the average time span for the non-MDD projects is longer than the included MDD projects. This might suggest that, over time (especially after implementing the initial functionalities), there is an increasing focus on modifying the system and fixing errors. Appendix J presents the progress of added, modified, and deleted files over a projects time span. For most non-MDD projects (see Figure J2), we can see the phenomenon that we suggested, where over time the number of added files (slightly) decreases in favour of modified or deleted files. For MDD projects (see Figure J1), we cannot discover such a pattern. This might be due to various reasons, such as, the MDD projects still being developed.



(a) Types of changes for MDD projects (b) Types of changes for non-MDD projects

Figure 4.5: A comparison between MDD and non-MDD projects in terms of the number of *added, modified* and *deleted* files relative to the total number of changes that have been made.

4.2.3 Revision characteristics

To better understand the types of changes that are made to an application, one approach is to analyse the comments or logs that are added at a commit. More specifically, can we discover differences in these comments (that contain details on software changes that are included in the revision) when we compare commit messages made to MDD projects to non-MDD projects? And, how does the size of a commit relate to these types of changes?

To analyse the commits this way, it requires us to map the vocabulary used in every single commit message (before relating it to the revisions size). As described in Section 2.2.3, we use the approach by Alali, Kagdi & Maletic [43] to compare the revisions of the MDD and non-MDD projects. The steps as presented in Figure 2.3 are described in more detail below, with the exception of the first step (i.e., dashed-arrow) in the figure (which is a step that was done prior to produce the tables in Appendix K).

Stemming

The first step is to collect all commit messages from each project and process each message by removing stop words and reduce inflected words to their word stem (i.e., stemming). The box below shows an example what this step entails for a commit message from the *Hadoop* project, using the Porter stemming algorithm.

Complete message: ‘Added functionality for schedulers to kill all applications in a queue’

Processed message: [‘add’, ‘function’, ‘schedul’, ‘kill’, ‘applic’, ‘queue’]

Term frequency

The next step is iterate through all commits within a project and count the frequencies of the stemmed terms. If we then take the full outer join of all projects, we have a complete frequency list for the (non-)MDD projects. Dividing the frequency by the number of commits (MDD: N = 8,700; Non-MDD: N = 104,776) results in a ranked term frequencies. Table 4.1 shows the top-10 most frequent terms, with the complete top-50 displayed in Appendix L.

The table shows some commonly used terms for both MDD and non-MDD, such as ‘add’, ‘fix’, and ‘update’. We also see terms that are more specific to the respective development method, like ‘Microflow’ for the Mendix platform.

Table 4.1: A snippet (top-10) of the most frequent terms present in commit messages for both MDD and non-MDD projects, all seven projects are covered in each respective column.

	MDD		**Non-MDD**	
	Terms	**Average rank**	**Terms**	**Average rank**
1	add	42.87%	fix	14.02%
2	chang	32.45%	add	11.92%
3	page	22.73%	merg	11.91%
4	updat	21.44%	test	9.18%
5	merg	19.19%	branch	8.74%
6	remov	15.18%	use	6.45%
7	microflow	12.87%	trunk	5.72%
8	fix	12.12%	remov	5.28%
9	creat	9.65%	updat	4.24%
10	modul	9.40%	chang	3.86%

Frequent term combinations

The 50 most frequent terms (see Appendix L), a threshold by the original authors, are transformed into item sets by making sets of sizes two and three. Identifying collocations helps us to explain the nature of a commit. We then go over the all the commits once again, now counting the frequency of the item sets and we link those to the commits size category that we defined in Section 4.2.1. As a final step, we remove any subsets for which a superset exists that includes all three elements of those subsets (for each size category). This way, we end up with an aggregated view on frequent item sets and their support[1] of each set for both MDD and non-MDD (see Appendix M).

When we consider the frequent item sets for MDD in Table M1, often item sets include the term 'add'. Across the size categories, item sets {add, user}, {update, microflow}, {add, check}, {add, new}, {add, modul}, {page, updat}, {chang, review}, and {add, attribut} occur the most (≥ 3 commit size categories). However, when we take their support into consideration, {change, review}, {add, module}, and {add, user} make up our top-3 item sets.

For the non-MDD projects (see Table M2), we find more frequent sets being present across the different sizes (20 sets ≥ 3 categories for non-MDD compared to 8 sets for MDD). We often see sets on {test, use}, {fix, use}, and {commit, revert}, as well as sets related to merging branches (e.g., {merge, branch, trunk}. Merging branches are also by far the most frequent when we look at their aggregated support, followed by {add, support} and {test, use}.

When we compare the frequent item sets for both MDD and non-MDD, we see quite little overlap between the included item sets. Only sets {add, new}, {add, use}, and {add, chang} are common in both. We also see that sets related to branch merging are occurring in the Extra Large commits in MDD projects, while the top-1 spots for non-MDD are always taken by an item set related to branch merges. The data also shows that non-MDD projects include more often revisions with the aim to fix something (e.g., 'fix error', 'fix merge', and 'fix bug') compared to its counterpart. The same is true for MDD, that includes often more revisions where something is 'added' compared to non-MDD projects. This supports our findings in Section 4.2.2, where MDD showed term sets that were about adding something and commit messages from non-MDD projects more often was about fixing things. Besides, the item sets in MDD includes more often object-oriented terms, such as {'chang', 'page'} and {'add', 'button'}, which is not the case for non-MDD projects.

Comparing our results of identified non-MDD term sets to the results by Alali, Kagdi & Maletic [43] shows us that for both, often term sets relate to fixes to the code. We see shared sets, such as {'fix', 'use'} and {'file', 'fix'}. However, we also see differences, mainly in the absence of branch related term sets in the results by Alali, Kagdi & Maletic.

[1] $Support = \frac{Number\ of\ commits\ in\ which\ the\ set\ appears}{Total\ number\ of\ commits}$

4.3 Lessons learned

With a secondary analysis on software revisions from both MDD and non-MDD origins, we got a step closer to understand their commonalities and differences. The analysis includes seven software projects from Mendix and seven randomly selected projects from the SEOSS33 data set [39].

Through a brief data exploration, we identified various patterns in development (see Appendix H). For non-MDD projects, active development participation either showed a calm start with an increase towards the end, a rapid initial start with a decrease in development towards the end or a stable pattern from beginning to end. For MDD, there was no commonly shared process to discover, with some projects showing multiple periods of almost no work being done. Still, projects Control, Learn, Portfolio, and Service do show similar patterns as their non-MDD counterparts.

Visualising the revisions' sizes showed that both MDD and non-MDD often entail a single change, with few exceptions (see Appendix I). Also, when the number of changes per commit are plotted and sorted by their median value (see Figure 4.3), it is not possible to differentiate between MDD and non-MDD projects. Although projects produced through MDD are located more to the right, the plots are still too mixed to be able to make a conclusion. Next, based on the work by Alali, Kagdi & Maletic [43], we divided each project into five size categories. For the categorisation, we used the projects own measurements and aggregated the results to make the comparison between (non-)MDD projects. This results in a similar distribution across the different sizes for MDD and non-MDD based projects, as can be seen in Figure 4.4.

When then compared the types of changes that are made to a project (see Figure 4.5), being either files are added, modified, or deleted through a commit. For both MDD and non-MDD, files are always added and modified more then they are deleted. For most MDD projects, more files were being added to the project then being modified, relatively speaking. For non-MDD projects, files are way often being modified, especially towards the end of a project.

Finally, we followed the approach by Alali, Kagdi & Maletic to get a better understanding of a projects' revision characteristics. Through various activities (see Figure 2.3) we identified frequent term sets for both MDD and non-MDD projects in the five prior defined size categories (see Appendix M). For MDD, this showed us that commits often relate to adding new functionalities (e.g., {add, user} or {add, module}). When we compare this to non-MDD projects, we see far more commits being focused on fixing artefacts or branch merging. Besides, the commits from MDD projects seem to be more objected-oriented compared to its non-MDD counterpart.

Chapter 5

Experiment

This chapter describes the experiment's execution and its results as outlined in Section 2.3. Section 5.1 presents how data is collected during the experiment. From Section 5.2 onwards, we present the results of the experiment. Section 5.2 gives an overview of the results that relate to the models' performance. This is then followed by the results regarding feature importance, where we elaborate on the features that were selected by the algorithms (see Section 5.3). Section 5.4 provides a summary and the most important aspects coming from the experiment.

5.1 Data collection

By conducting this experiment, we aim to collect value data on different techniques with the goal to boost the models performance and reduce the feature space (for more details see Section 2.3.1). In this section, the selected candidate techniques are presented (see Section 5.1.1) and the data collection process (see Section 5.1.2) is discussed.

5.1.1 Treatment design

Through the literature review, we got an impression of techniques available to us. Based on that, a selection is made to experiment on four candidate techniques across the three feature selection types (i.e., filter, wrapper, and hybrid). The techniques are listed below and is followed by a justification for the selected techniques.

1. Filter (mRMR (FCQ))
2. Filter (mRMR (RFCQ))
3. Wrapper (Recursive Feature Elimination (RFECV))
4. Hybrid (FeatureWiz)

mRMR. mRMR belong to the filter-based feature selection approaches. Based on a study by Zhao, Anand, and Wang [91], who evaluated seven variants of the mRMR feature selection technique, two techniques were selected to be subjected to this experiment. From the seven variants that Zhao et al. evaluated, the *F-test correlation quotient (FCQ)* and *random forests correlation quotient (RFCQ)* variants produced the least correlated feature sets. They measured the time for computing and ranking the features based on computation speed metrics, with *FCQ* being the fasted, closely followed by *F-test correlation difference (FCD)* and *RFCQ*. They tested the techniques on both synthetic data and real data, for which *FCQ* and *RFCQ* showed the best performance compared to other variants in terms of computation times and AUC scores.

As described by Zhao et al., the *FCQ* uses the F-statistic to score the relevance and the correlation to score the redundancy. The formula for the *FCQ* scheme is presented in Equation 5.1.

$$f^{FCQ}(X_i) = \frac{F(Y, X_i)}{\frac{1}{|S|}\sum_{X_s \in S} \rho(X_s, X_i)}, \tag{5.1}$$

where the F-statistic is denoted as $F(Y, X_i)$ and the Pearson correlation as $\rho(X_s, X_i)$.

The *RFCQ* is an extension on mRMR proposed by the authors to determine the relevance based on the random forests feature importance score. The formula for *RFCQ* is provided in Equation 5.2.

$$f^{RFCQ}(X_i) = \frac{I_{RF}(Y, X_i)}{\frac{1}{|S|}\sum_{X_s \in S} \rho(X_s, X_i)}, \tag{5.2}$$

where the relevance score is denoted by $I_{RF}(Y, X_i)$ and the denominator is equal to the one given in Equation 5.1.

As the mRMR algorithms need the number of features it needs to keep in the feature space, we let the algorithms to compute feature sets for thee subset sizes (i.e., k=40, k=50, and k=60). Because we create additional sets for the same algorithm, we will test if there is a statistical significance in the set sizes in case the mRMR algorithms show the best performance than the other selection techniques.

RFECV. As explained in the literature review, RFE (as our wrapper approach) recursively eliminates the least important feature from the total feature space. Dependent on the choice of implementation, the iterations stop when it either accomplishes a certain score (e.g., based on a F-measure, number of features left in the feature space, or iteration rounds). RFE, however, can be negatively influenced by the ranked attributes and is therefore susceptible to training data as it produces attribute rankings with high variation [92]. They suggest to incorporate cross-validation (CV) to the RFE method to overcome these limitations and have the algorithm more stable and reliable. The python package from Scikit Learn provides an RFECV[1] method. A few parameters need to be configured, which are stated below.

1. Classifier: Random Forests
2. Cross-validation: StratifiedKfold (5 times)
3. Steps (number of features to remove per iteration): 1
4. Scoring: F1 score

Featurewiz. To use a hybrid approach to feature selection, first a filter method (based on the mRMR algorithm) is applied to remove redundant features, which is followed by a wrapper method to search for the best subset using RFE. In the approach we will be testing is offered through an open-source Python package called FeatureWiz[2].

Figure 5.1 provides a visualisation of the FeatureWiz process. It starts with providing the algorithm with a data set (i.e., populated features) and defining the target variable. As described by De Carvalho and Victor [93], first a method named SULOV (based on the described mRMR algorithm) is applied where the algorithm searches for correlated features based on a correlation threshold. When a pair of features is identified as correlated, the feature that has the highest MIS[3] (Mutual Information Score) is kept in the feature space. After this first stage, we are left with a reduced feature space. The second stage applies RFE in combination with the XGBoost algorithm to determine the best feature set from the reduced feature space. The features are recursively passed through the algorithm, given a target variable, where models are created and variables are eliminated that increase the error of those models. The output is a populated feature data frame with only data on the selected features.

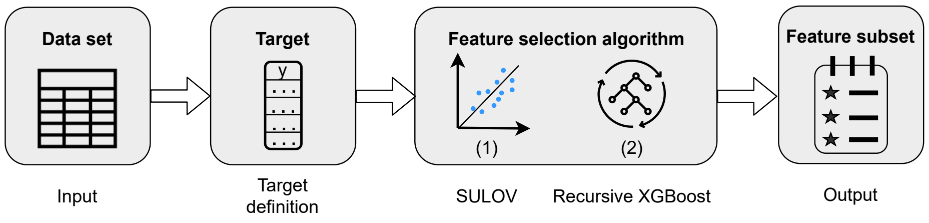


Figure 5.1: FeatureWiz process visualisation

A hybrid method can be created through various combinations of filter- and wrapper methods, though the choice was made for this specific implementation because it combines the other discussed techniques. This allows us to test if these other techniques can strengthen each other. As researchers identified, starting with removing redundant features through correlation is a good first step, as other

[1]More details on `https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFECV.html`

[2]More details on `https://github.com/AutoViML/featurewiz`

[3]MIS is a non-parametric scoring method that is capable of quantifying the uncertainty in variables and scaling the shared information among variables [94].

techniques (e.g., based on information gain) could select features that are effective, but contain features that are too similar (leading to overfitting) [46].

5.1.2 Experiment process

This subsection describes the execution of the experiment's operation activities, such as the preparation of the data sets and running the feature selection algorithms. The process follows the three steps, as described in Section 2.3.3, consisting of preparation, execution, and data validation. An overview of the complete experimental method can be found in Appendix N.

Preparation. In Section 2.3.3, we described the steps necessary to prepare for the experiment. As a result, an overview of the baseline performance results can be found in Table O1, that displays the relevant performance scores on the 10 software projects using all 131 features. The baseline results regarding feature importance is displayed in Table O2. The Python notebooks that are used to facilitate the experiment can be found in the online Appendix [33]. Note that these include parts of the notebooks that can be found on the RELab Github[4].

Execution. All 10 projects are subjected to the 4 feature selection algorithms. The experimental environment (i.e., experiment_pipeline.ipynb in the online Appendix [33]) is designed to take one project at a time, while for each run the notebook executes the selection algorithms and runs the model evaluation script for each created feature subset. This means that the notebook is run ten times, producing the output files as listed below (see Table 5.1).

Table 5.1: Overview of the files that were generated during the experiment. These files can be found in the online Appendix [33]. The values within the parentheses are the number of files for all 10 software projects combined.

Generated files	File type	Algorithms mRMR (FCQ)	mRMR (RFCQ)	RFECV	FeatureWiz
Subset of populated features	.xlsx	3 (30)	3 (30)	1 (10)	1 (10)
Performance evaluation					
Trace recommendation	.csv	3 (30)	3 (30)	1 (10)	1 (10)
Trace maintenance	.csv	3 (30)	3 (30)	1 (10)	1 (10)
Feature importance					
Trace recommendation	.csv	3 (30)	3 (30)	1 (10)	1 (10)
Trace maintenance	.csv	3 (30)	3 (30)	1 (10)	1 (10)
Sub total		15 (150)	15 (150)	5 (50)	5 (50)
Total					40 (400)

To support the interpretability of the output files, a second notebook is used to aggregate over the performance evaluation files (i.e., generating average, standard deviation, and maximum values from the raw files) and displays the results in tabular format. This notebook can be found in the online Appendix [33] by the name process_experiment_results.ipynb.

Data validation. The output files of the feature subsets and the model evaluations were reviewed. No abnormalities have been found in the output. All files that were generated in the experiment can be found in the online Appendix [33].

5.2 Models' performance

With all data from the experiment collected and documented, we can start analysing the results. Similar to the previous section, this section follows the steps as described in Section 2.3.4. First, we discuss the number of features that were selected by the algorithms (see Section 5.2.1). Thereafter, we will look into the effect of the different values for K for both mRMR algorithms (see Section 5.2.2). Thirdly, the results in terms of performance of each scenario are presented by the use of descriptive statistics, visualisation techniques, and statistical tests (see Section 5.2.3 and 5.2.4). Then, in Section 5.3, we will present the results in terms of feature importance that is gained by the selection algorithms.

[4]More details on `https://github.com/RELabUU/LCDTrace`

The results from the experiment (processed by the priory described notebook) can be found in Appendix P, where for each feature selection algorithm the results are displayed in tabular fashion. Appendix Q visualises the results in Box and Whisker plots.

5.2.1 Selected features

The tables in Appendix P present, besides performance related scores, the number of features that were selected by each algorithm for every project. For the mRMR algorithms, these were predefined (and set to K=40, K=50, K=60) as required by the algorithm. For the other algorithms, it was up to the algorithm to come up with an optimal number of features for each project. In Tables 5.4 and 5.7, we see that RFECV has on average selected 15 features, though when we look at Table P3, we see a wide range of subset sizes (ranging from 3 selected features to 36 features). FeatureWiz (see Table P4) seems to be more stable, which produced subset sizes in between 30 and 40 features. When considering characteristics of the projects, such as the dimensions of each data set (Jira issues $\times$ revisions), there seem to be no relation between the characteristics and the number of features selected by the RFECV algorithm. However, we further look into the selected features in terms of feature families when we go into feature importance (see Section 5.3).

5.2.2 mRMR and their values for K

Before we compare the results of all feature selection algorithms, we first present the results of both mRMR algorithms and discuss the effects of the different subset sizes given to the algorithms. We first discuss the algorithm that uses the FCQ scheme, which is followed by the RFCQ scheme.

mRMR (FCQ)
When comparing the results in Table P1 for *trace recommendation*, the thing that stands out is that most highest values are recorded for subset size K=40 or K=60. By just looking at the table, there is no value for K that clearly outperforms the others. When comparing the mean F_2, we see that across projects varying ranges of the scores are being reported. For example, *Portfolio* ranges between 55.96% and 57.03% where *Store* ranges between 62.19% and 73.45%. This is clearly visible in Figure Q3.1 as well. Other than that, the only thing we can distinguish is that performance often alternates between the values for K.

For *trace maintenance*, again looking at Table P1, the highest values seem to be scored for K=60. The macro averages show scores that are very similar across the values for K. On a project level, we see a similar distribution as for trace recommendation, where *Store* seems to have larger differences across the metrics compared to the other projects. When considering $F_{0.5}$ in Figure Q3.2, no clear winner can be distinguished, as the performance seems to shift when considering the projects.

To statistically test the effect of the different values for K, we used Friedman test with Nemenyi's post-hoc analysis (as substantiated in Section 2.3.4). A notebook is used to perform the test, which can be found in the online Appendix [33]. The hypothesis for this test is stated as follows:

> Null hypothesis ($\mathbf{H}_0$): There are no differences in the performance of the ML classification model for subset sizes K=40, K=50, or K=60.
>
> Alternative hypothesis, ($\mathbf{H}_1$): There is a difference in the performance of the ML classification model for subset sizes K=40, K=50, or K=60.

The results of the test can be found in Table 5.2. We interpret the results from the table as follows. A Friedman test was conducted to determine whether using different subset sizes (K=40, K=50, K=60) in the mRMR algorithm with FCQ scheme results in a significant effect on the performance (in terms of precision, recall, and F-measure) of the ML classification model. The results indicated that there was a significant difference for the different values for K for *trace recommendation* in terms of precision, $\chi^2(2) = 7.800$, p = 0.020. A post-hoc comparison was conducted using Nemenyi's test. The analysis of this test indicated that there was a significant difference between K=40 and K=60, with $p \leq 0.05$. We therefore reject the null hypothesis for precision and conclude that there is a difference in precision using K=40 and K=60 in the trace recommendation scenario. For the other metrics and trace maintenance, we fail to reject the null hypothesis.

Table 5.3: Statistical test for mRMR (RFCQ), using Friedman test followed by Nemenyi's post-hoc test. The coloured cells indicate a statistical significance with p ≤ 0.05.

Metric	Trace recommendation				
	Friedman		Nemenyi (post-hoc)		
	χ^2	p	K=40 vs. K=50	K=40 vs. K=60	K=50 vs. K=60
Precision	.800	.670	.632	.888	.888
Recall	3.231	.199	.900	.214	.373
F_2	7.800	.020	.760	.109	.020

Metric	Trace maintenance				
	Friedman		Nemenyi (post-hoc)		
	χ^2	p	K=40 vs. K=50	K=40 vs. K=60	K=50 vs. K=60
Precision	1.400	.497	.900	.632	.504
Recall	.600	.741	.760	.760	.900
$F_{0.5}$	.800	.670	.632	.888	.888

Table 5.2: Statistical test for mRMR (FCQ), using Friedman test followed by Nemenyi's post-hoc test. The coloured cells indicate a statistical significance with p ≤ 0.05.

Metric	Trace recommendation				
	Friedman		Nemenyi (Post-Hoc)		
	χ^2	p	K=40 vs. K=50	K=40 vs. K=60	K=50 vs. K=60
Precision	7.800	.020	.760	.020	.109
Recall	1.800	.407	.373	.760	.760
F_2	2.400	.301	.373	.900	.373

Metric	Trace maintenance				
	Friedman		Nemenyi (Post-Hoc)		
	χ^2	p	K=40 vs. K=50	K=40 vs. K=60	K=50 vs. K=60
Precision	.800	.670	.888	.632	.888
Recall	2.600	.273	.504	.261	.888
$F_{0.5}$	1.400	.497	.900	.504	.632

mRMR (RFCQ)

The results of the mRMR algorithms using the RFCQ scheme can be found in Table P2. The values in the table for *trace recommendation* indicate a possible preference for smaller subset sizes, as most high values are recorded for K=40 and K=50. Also when comparing the macro averages for recall (i.e., dominant metric in this scenario), a smaller subset size seem to be more favourable. For F_2, the differences are smaller. It is likely that this is a result of higher precision scores for K=60, which works through in more balanced F-measure across the values for K. These characteristics can also be seen in Figure Q4.1. However, the figure also indicates that only in a few occasions K=60 is on the same level as the others. Often it is outperformed, although the differences seem to be small.

For *trace maintenance*, the results from Table P2 seem to be more balanced across the values for K (with small differences in performance). Again, like for the FCQ scheme, the box plots (see Figure Q4.2) show varying performance by the values for K across the projects. Therefore, we do not expect to detect a significant difference when we run statistical tests on this.

Still, to substantiate our findings through descriptive statistics, we again perform the statistical tests as we did for the FCQ scheme. The hypothesis as presented for FCQ are tested also for the RFCQ scheme, because the setup of the test is equal to the one used for the FCQ scheme.

The results of the test can be found in Table 5.3. We interpret the results from the table as follows. A Friedman test was conducted to determine whether using different subset sizes (K=40, K=50, K=60) in the mRMR algorithm with RFCQ scheme results in a significant effect on the performance (in terms of precision, recall, and F-measure) of the ML classification model. The results indicate that there is a significant difference for the different values for K for *trace recommendation* in terms of the F_2-measure, $\chi^2(2) = 7.800$, p = 0.020. A post-hoc comparison was conducted using Nemenyi's test. The analysis of this test indicated that there was a significant difference between K=50 and K=60, with p ≤ 0.05. We therefore reject the null hypothesis and conclude that there is a difference in F_2 using K=50 and K=60 in the trace recommendation scenario. For the other metrics and trace maintenance, we fail to reject the null hypothesis.

How the different values for K for both the mRMR algorithms perform in relation to the baseline is presented in their designated sections (see Section 5.2.3 for trace recommendation and Section 5.2.4 for trace maintenance). It will especially be interesting to see if the values for K=50 and K=60 differ in those tests.

5.2.3 Trace recommendation

Before we go more in depth and see the performance on a project level, let us first set our viewpoint on a higher level. From the results as presented in Appendix P, Table 5.4 provides a summary in tabular fashion, with Figure 5.2 providing a visualisation of these results.

Table 5.4: Mean and standard deviation for precision, recall, and F$_2$-measure for **trace recommendation** across the four feature selection algorithms. The values present aggregated results across all ten data sets. The colored cells highlight which selection algorithm has the highest metric, which is shown in green for the trace recommendation scneario

		Trace recommendation								
Feature selection		Precision			Recall			F_2		
algortihm	K	$\overline{x}$	$\overline{\sigma}$	$\overline{Max}$	$\overline{x}$	$\overline{\sigma}$	$\overline{Max}$	$\overline{x}$	$\overline{\sigma}$	$\overline{Max}$
mRMR (FCQ)	40	41.03	3.45	46.44	71.61	5.96	79.56	61.60	4.46	68.04
	50	41.85	3.33	47.71	69.37	5.81	79.01	60.50	4.23	67.39
	60	44.08	4.33	51.19	69.84	5.40	78.36	61.83	4.59	69.24
mRMR (RFCQ)	40	42.91	4.10	49.33	68.17	6.55	77.74	60.56	5.24	68.84
	50	46.20	4.49	52.97	67.81	5.65	76.50	61.59	4.65	69.04
	60	45.72	4.57	53.11	65.47	7.32	77.69	59.78	5.98	69.68
RFECV	15	32.18	2.96	37.50	74.12	5.49	83.10	55.38	3.97	61.33
FeatureWiz	35	47.79	4.44	55.40	67.15	5.87	76.12	61.88	4.73	68.97

For trace recommendation, we can clearly identify the more important metric in the F_2-measure in this scenario (as recall scores higher overall compared to precision). In Figure 5.2, all algorithms show a comparable scores for precision, with the exception of RFECV (which performs the least). The plots do indicate larger differences between the maximum precision. The scores for recall show more stable maximum values and a similar range of average recall across the algorithms. RFECV does show the best performance for recall. The plots also show more outliers in the lower half of the figure. This could indicate that either the performance regarding recall is varying a lot, or the number of evaluation rounds (currently set at 10) is too small (in combination with these variations). When considering the F_2-measure, we see a stable average result in the figure. As presented in Table 5.4, the average F_2-measure across the algorithms are ranging in between 59.78% and 61.88%. With an average F_2-measure of 60.94% by using the complete feature set, this falls right in the middle of that range. All the algorithms do have large differences between its whiskers ($\geq$ 40%). When only considering the mRMR algorithms (FCQ and RFCQ), there is no indication of a clear winner for this scenario when looking at aggregated results (as we saw in Section 5.2.2). As Figure 5.2 indicates similar F_2 performance across the algorithms, it would be interesting to see what the differences are in terms of the specific features that the algorithms selected (see Section 5.3.1).

When we evaluate the results on a project level, we get a more in depth picture. Figure Q1 displays the F_2-measures that resulted from the 10 evaluation rounds on each classifier, for all projects across the four feature selection algorithms.

When comparing the plots for each project individually, we can see some differences. Looking at the plots in Figure Q1, there is no selection algorithm that is able to always score better or similar compared to the baseline. The performance by FeatureWiz and the mRMR algorithms does seem to be similar to the baseline (while varying between projects to be better or worse than the baseline). We can also see that RFECV does score the lowest for *Control_1, Learn_1, Portfolio, Service, and Store*. Knowing this, and looking at the number of features that were selected by the algorithm for those projects (see Table P3), it is likely these low scores are caused by the low number of features in those sets (i.e., 6, 8, 3, 6, and 5 respectively). Another thing that we see in the figure is that for both *Learn_2* and *Store*, the box plots are larger (i.e., larger range) compared to other projects. Table G1 might provide an explanation to this. When we take the number of issues over

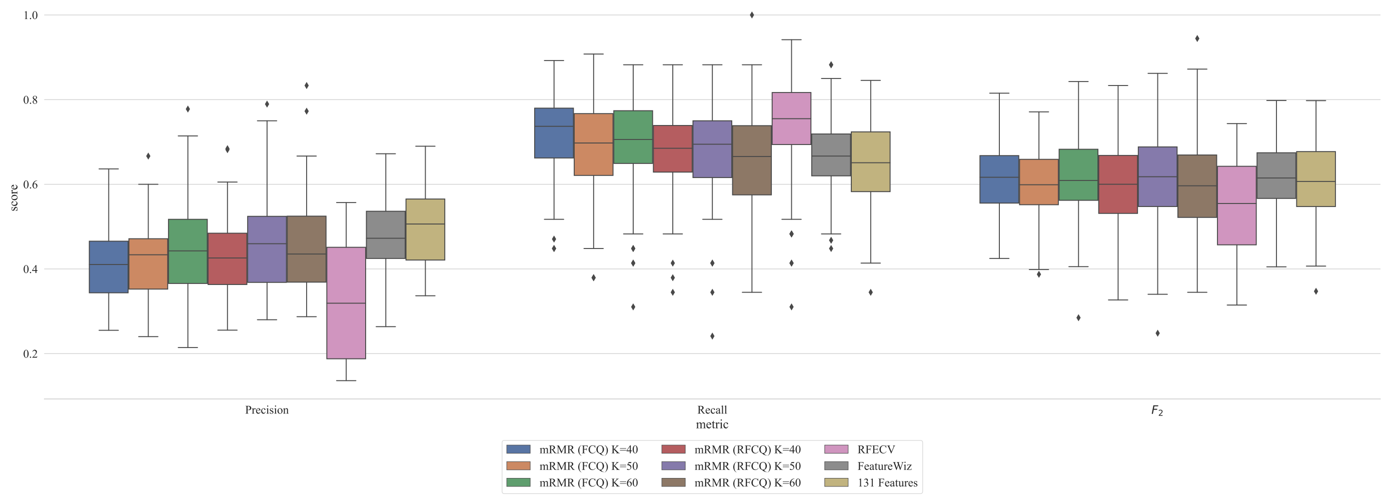


Figure 5.2: Box and Whisker plots of aggregated results for trace recommendation for the four feature selection algorithms for all ten data sets.

the number of revisions, *Learn_2* and *Store* have ratio's of 0.55 and 0.89 respectively. For other projects, these numbers are at most 0.22 (*Learn_1*), with many scoring ratio's well below 0.10.

When looking at actual numbers, Table 5.5 shows the performance in terms of the F_2-measure across all software projects, resulting from Tables P1, P2, P3, and P4. Often the subsets are not performing worse than the baseline of 131 features and often the baseline is outperformed by one of the selection algorithms. For projects *Control_1, Control_2, and Control_3*, this means that the baseline is outperformed by one of the selection algorithms at least 75% of the time. For three additional projects (*Data, Learn_2, and Store*) this happens at least 50% of the time. We can also see that no algorithm outperforms the baseline for every project.

Table 5.5: Summary of F_2-measures for trace recommendation across the selection algorithms (including baseline) for each project. Cells in orange indicate that the baseline is outperformed, with green cells indicate the highest score for that project (i.e.. horizontally oriented). The change column displays the percentage change[5] between the highest value with respect to the baseline, with plus and minus signs indicating its difference with regards to the feature subsets.

Project	Baseline	Change	Feature selection algorithm							
			mRMR (FCQ)			mRMR (RFCQ)			RFECV	Feature-
		%	K = 40	K = 50	K = 60	K = 40	K = 50	K = 60		Wiz
Company	63.69	+2.78	64.06	61.94	59.99	62.54	65.46	62.20	61.83	62.39
Control_1	50.41	+7.25	54.07	52.21	52.79	52.17	52.40	49.23	44.49	51.01
Control_2	55.81	+10.88	59.39	57.60	61.89	58.34	60.24	55.46	61.00	61.32
Control_3	66.39	+6.11	68.28	70.08	66.70	69.12	68.54	68.21	66.66	70.45
Data	72.94	+1.35	72.33	73.20	73.62	62.94	73.93	73.49	70.86	72.41
Learn_1	57.01	+2.59	57.65	56.85	57.16	53.89	56.34	54.94	41.45	58.49
Learn_2	48.27	+13.56	52.65	52.03	46.76	51.46	46.50	45.95	45.38	54.81
Portfolio	58.74	−0.34	55.96	56.66	57.03	58.54	57.67	57.05	51.33	57.67
Service	68.69	+0.30	64.75	62.19	68.89	65.85	63.57	64.80	55.39	63.40
Store	67.43	+8.93	66.86	62.19	73.45	70.70	71.25	66.47	55.46	66.87

Hypothesis testing

To strengthen the presented results, a statistical analysis is performed to see if a significance can be detected between the baseline and the feature selection algorithms. Similar to the statistical tests for the mRMR algorithms, we use the Friedman non-parametric test in combination with Nemenyi's post-hoc analysis. However, this time we make the comparison between the baseline and the selection algorithms, rather than a comparison between the algorithms. For completeness, the hypothesis that we aim to test is stated below.

Null hypothesis ($\mathbf{H}_0$): There are no differences in the performance of the ML classification model between the baseline and the treatments.

Alternative hypothesis, ($\mathbf{H}_1$): At least one of the treatments is significantly different in terms of performance of the ML classification model compared to the baseline.

The results of the test is displayed in Table 5.6. We interpret the results from the table as follows. A Friedman test was conducted to determine whether using feature selection algorithms results in a significant effect on the performance of the ML classification model compared to our baseline that uses the complete feature set.

Table 5.6: Friedman test + Nemenyi post-hoc test for trace recommendation. The coloured cells indicate a statistical significance with p $\leq$ 0.05.

Metric	Friedman		Nemenyi (post-hoc)			
	χ^2	p	Comparison			p
Precision	46.507	1.904×10^{-7}	Baseline vs.	mRMR (FCQ)	K=40	.001
					K=50	.004
					K=60	.257
				mRMR (RFCQ)	K=40	.050
					K=50	.809
					K=60	.562
				RFECV		.001
				FeatureWiz		.900
Recall	32.167	8.693×10^{-5}	Baseline vs.	mRMR (FCQ)	K=40	.005
					K=50	.279
					K=60	.063
				mRMR (RFCQ)	K=40	.562
					K=50	.809
					K=60	.900
				RFECV		.001
				FeatureWiz		.760
F_2	22.320	0.004[6]	Baseline vs.	mRMR (FCQ)	K=40	.900
					K=50	.900
					K=60	.900
				mRMR (RFCQ)	K=40	.900
					K=50	.900
					K=60	.900
				RFECV		.150
				FeatureWiz		.900

For **precision**, the results indicate that there is a significant difference between the baseline and at least one of the treatments, $\chi^2(7) = 46.507$, p = 1.904×10^{-7}. A post-hoc comparison was conducted using Nemenyi's test. The analysis indicated that there was a significant difference for both mRMR (FCQ) K=40 and K=50, and RFECV, with p $\leq$ 0.05. We therefore reject the null hypothesis and conclude there is a difference between the baseline and multiple selection algorithms.

For **recall**, the results indicate that there is a significant difference between the baseline and at least one of the treatments, $\chi^2(7) = 32.167$, p = 8.693×10^{-5}. A post-hoc comparison was conducted using Nemenyi's test. The analysis indicated that there was a significant difference for mRMR (FCQ) K=40 and RFECV, with p $\leq$ 0.05. We therefore reject the null hypothesis and conclude there is a difference between the baseline and multiple selection algorithms.
For $\mathbf{F_2}$, the results indicate that there is a significant difference between the baseline and at least one of the treatments, $\chi^2(7) = 23.320$, p = .004. A post-hoc comparison was conducted using Nemenyi's test. However, the analysis indicated that there was a no significant difference between the baseline and one of the treatments. We therefore fail to reject the null hypothesis and conclude there is no difference between the baseline and the selection algorithms.

With both the results in Table 5.5 and our statistical analysis we can conclude the following. From our statistical test, we saw that the RFECV is significantly different to the baseline in terms of precision and recall. Considering the results regarding F_2, we can conclude that the RFECV algorithm is likely to produce worse results than the baseline. For the mRMR (FCQ) algorithm we can state the opposite. The precision and recall are again statistically significant, but when we consider its results

[5]Calculated by $\frac{New\ number - Old\ number}{Old\ number} \times 100\%$

[6]The Friedman test detected a significance in F_2 (as reported in the table). However, the post-hoc test found that this significance is between two feature selection algorithms, which is outside the scope of this analysis.

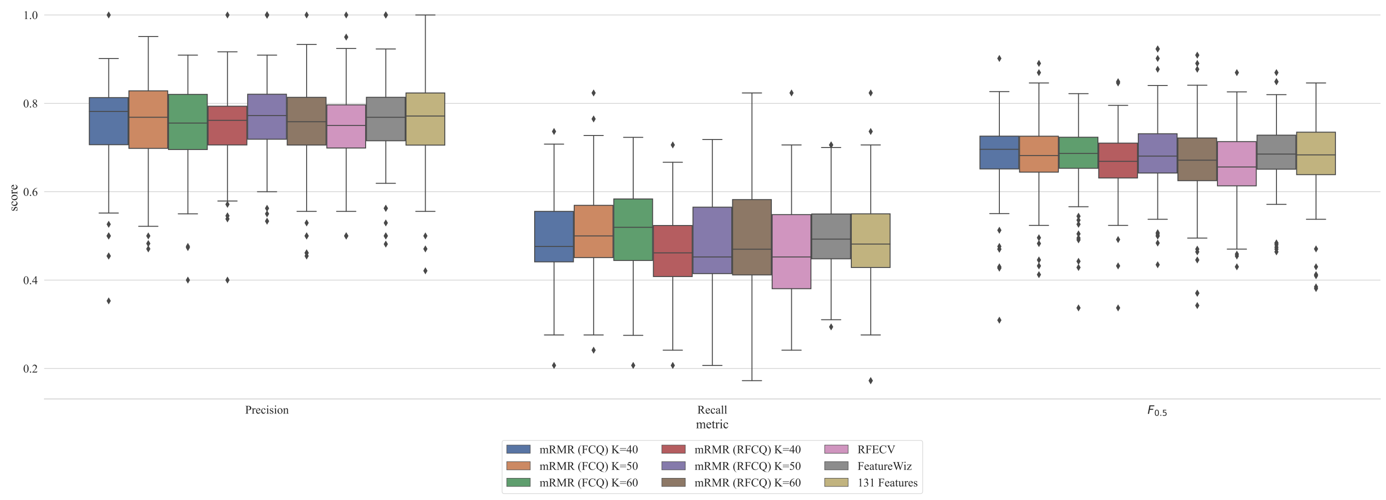


Figure 5.3: Box and Whisker plots of aggregated results for trace maintenance for the four feature selection algorithms for all ten data sets.

it shows improvements over the results by our baseline (though not indicated by the statistical test in terms of F_2).

5.2.4 Trace maintenance

For trace maintenance, we again first take a look at the aggregated results (see Table 5.7). The table presents the results, as a summary from Appendix P, with Box and Whisker plots to present the visual representation of the results (see Figure 5.3).

Table 5.7: Mean and standard deviation for precision, recall, and $F_{0.5}$- for **trace maintenance** across the four feature selection algorithms. The values present aggregated results across all ten data sets. The colored cells highlight which selection algorithm has the highest metric, which is shown in orange for the trace maintenance scenario.

Feature selection algorithm	K	Trace maintenance								
		Precision			Recall			$F_{0.5}$		
		$\overline{x}$	$\overline{\sigma}$	$\overline{Max}$	$\overline{x}$	$\overline{\sigma}$	$\overline{Max}$	$\overline{x}$	$\overline{\sigma}$	$\overline{Max}$
mRMR (FCQ)	40	75.00	5.65	84.40	49.40	5.35	58.55	67.62	4.80	75.27
	50	75.61	6.03	86.22	50.96	5.62	59.61	68.48	4.91	76.33
	60	74.63	6.25	85.13	51.12	6.53	60.77	67.91	5.26	75.19
mRMR (RFCQ)	40	74.98	6.83	85.30	46.49	6.37	55.83	66.31	5.93	74.16
	50	77.07	6.23	86.24	48.17	5.06	55.18	68.33	4.96	75.31
	60	74.85	5.71	83.56	48.55	6.11	57.87	67.05	5.21	74.84
RFECV	15	74.62	6.78	87.14	47.54	5.75	56.27	66.37	5.34	75.15
FeatureWiz	35	75.85	5.94	84.74	49.67	5.22	58.47	68.19	4.54	75.86

For trace maintenance, the differences between the algorithms seem to be even smaller compared to the previous scenario. Obviously, precision and recall swapped their importance as Figure 5.3 indicates (in comparison with Figure 5.2). For precision, the average scores range between 75.00% and 77.07%. It should be noted that the figure does show many outliers for precision. The mRMR algorithms that use the FCQ scheme seem to score better compared to the others when looking at recall. The same can be said about the scores of the $F_{0.5}$-measure (see Table 5.7), though the scores are very close, ranging between 66.37% and 68.48%. Similar to trace recommendation, the average score achieved by using the complete feature set is with 67.59% right within that range. The figure does indicate that the scores for $F_{0.5}$ are less scattered (made visually due to the IQR being small), making the results more stable. However, each plot does show many outliers. If these were not identified as outliers and were included in the default plot, it would look very different.

The results of the trace maintenance scenario, in terms of the $F_{0.5}$-measure, can be found in Figure Q2. The results (of $F_{0.5}$) as presented in tables of Appendix 5.2 are summarised in Table 5.8. We can see a few things when we look at the figure. For one, the baseline seems always to be outperformed in terms of $F_{0.5}$ by at least one of the mRMR algorithms. For the other algorithms, this is varying a lot per project. We also see a similar trend as we saw for trace recommendation, being that the size of the boxplot's IQR differ a lot between the each project.

When looking at average numbers (see Table 5.8), we get a stronger picture of the performance of the selection algorithms compared to the baseline. Though, sometimes the difference is small (e.g., for *Control_1, Control_3, and Data*), the baseline is always outperformed by one of the algorithms. Especially for the subsets K=50 and K=60, that were selected by the mRMR algorithm with FCQ scheme, outperform the baseline 80% of the time. The other algorithms only occasionally outperform the baseline.

Table 5.8: Summary of $F_{0.5}$-measures for trace maintenance across the selection algorithms (including baseline) for each project. Cells in orange indicate that the baseline is outperformed, with green cells indicate the highest score for that project (i.e.. horizontally oriented). The change column displays the percentage change[7] between the highest value with respect to the baseline. with plus and minus signs indicating its difference with regards to the feature subsets.

Project	Baseline	Change	Feature selection algorithm							
			mRMR (FCQ)			mRMR (RFCQ)			RFECV	Feature-
		%	K = 40	K = 50	K = 60	K = 40	K = 50	K = 60		Wiz
Company	67.20	+5.08	69.28	70.04	67.24	68.87	70.61	66.86	69.90	66.74
Control_1	66.91	+1.29	67.15	66.21	67.78	64.40	64.96	66.67	61.75	65.70
Control_2	64.43	+3.54	62.41	64.73	65.02	65.56	61.86	66.71	65.88	64.26
Control_3	70.83	+0.79	66.79	67.73	68.99	66.31	69.61	67.05	67.40	71.39
Data	78.39	+1.92	77.27	78.62	79.89	69.10	76.99	78.59	77.83	75.86
Learn_1	69.19	+4.89	72.57	70.45	69.62	67.39	68.17	65.74	66.68	69.11
Learn_2	49.20	+16.46	47.23	51.28	52.29	54.37	57.30	47.00	53.25	52.61
Portfolio	66.07	+5.36	69.61	67.40	68.48	66.81	63.78	64.50	60.24	68.05
Service	69.02	+5.52	70.17	69.05	71.54	71.09	67.99	68.54	66.14	72.83
Store	74.62	+9.94	73.74	79.33	68.28	69.23	82.04	78.83	74.63	75.32

Hypothesis testing

We now perform statistical tests to substantiate the findings that were found for the trace maintenance scenario. Again, the Friedman test is used together with Nemenyi's post-hoc analysis. The hypothesis that we want to test is equal to the hypothesis as presented for trace recommendation (see Section 5.2.3).

The results of the test is displayed in Table 5.9. We interpret the results from the table as follows. A Friedman test was conducted to determine whether using feature selection algorithms results in a significant effect on the performance of the ML classification model than our baseline that uses the complete feature set.

For **precision**, the results indicate that there is no significant difference between the baseline and at least one of the treatments, $\chi^2(7) = 2.747$, p = .949. We therefore fail to reject the null hypothesis and conclude there is no difference between the baseline and multiple selection algorithms.

For **recall**, the results indicate that there is a significant difference between the baseline and at least one of the treatments, $\chi^2(7) = 22.640$, p = .004. A post-hoc comparison was conducted using Nemenyi's test. However, the analysis indicated that there was a no significant difference between the baseline and one of the treatments. We therefore fail to reject the null hypothesis and conclude there is no difference between the baseline and the selection algorithms.

For $\mathbf{F_{0.5}}$, the results indicate that there is no significant difference between the baseline and at least one of the treatments, $\chi^2(7) = 7.280$, p = .507. We therefore fail to reject the null hypothesis and conclude there is no difference between the baseline and multiple selection algorithms.

[7]Calculated by $\frac{New\ number\ -\ Old\ number}{Old\ number} \times 100\%$

Table 5.9: Friedman test + Nemenyi post-hoc test for trace maintenance. The coloured cells indicate a statistical significance with p $\leq$ 0.05.

Metric	Friedman χ^2	Friedman p	Nemenyi (post-hoc) Comparison			p
Precision	2.747	.949	Baseline vs.	mRMR (FCQ)	K=40	.900
					K=50	.900
					K=60	.900
				mRMR (RFCQ)	K=40	.900
					K=50	.900
					K=60	.900
				RFECV		.900
				FeatureWiz		.900
Recall	22.640	.004[8]	Baseline vs.	mRMR (FCQ)	K=40	.900
					K=50	.302
					K=60	.150
				mRMR (RFCQ)	K=40	.900
					K=50	.900
					K=60	.900
				RFECV		.900
				FeatureWiz		.900
$F_{0.5}$	7.280	.507	Baseline vs.	mRMR (FCQ)	K=40	.900
					K=50	.900
					K=60	.900
				mRMR (RFCQ)	K=40	.900
					K=50	.900
					K=60	.900
				RFECV		.900
				FeatureWiz		.900

5.3 Feature importance

Feature importance provides some explanatory power to the ML models. Feature importance is a technique where we can assign a score to each feature that is used in the model based on its prediction value. Recall that in the baseline used 131 features. As briefly discussed in our literature review (see Section 3.3.1), the features are part of four feature families. First, we provide a bit more insight in feature importance as a metric before presenting our results. Thereafter, the results of feature importance for both trace recommendation (see Section 5.3.1) and trace maintenance (see Section 5.3.2) are presented and analysed.

The raw results produced by the experiment (as reported in Table 5.1) are processed by a jupyter notebook, which can be found in the online Appendix [33]. The notebook assigns the results of each feature to its feature family and calculates descriptive statistics. The outputs for trace recommendation and trace maintenance can be found in Appendix R.

Information gain
Feature importance can be expressed in different ways, such as coverage (i.e., relative number of observations related to a feature) or weight (i.e., number of times a feature occurs in the trees of a model) [95]. The evaluation script that is used in the experiment expresses the feature importance in terms of information gain. As explained in the work by Rasiman [21], information gain is the default metric used by both the classifiers, though on a slight different level. This means that for trace recommendation (which uses the LightGBM classifier), the feature importance is reported in terms of total gain, whereas trace maintenance in its turn uses the metric in terms of average gain (the default metric for the XGBoost classifier).

5.3.1 Trace recommendation

Similar to the previous section, we first present the results for trace recommendation from a high level before we go more in depth. The results of all feature selection algorithms for trace recommendation, as presented in Appendix R, are summarised in Table 5.10.

[8]The Friedman test detected a significance in recall (as reported in the table). However, the post-hoc test found that this significance is between two feature selection algorithms, which is outside the scope of this analysis.

First, we look at the baseline results (more details are available in Table O2). We see that *process related* features are representing the most (over two-thirds) in terms of feature importance across all feature families. This is followed by *QQ-specificity* features, with the remaining families contributing very little to the overall performance of the model.

When we compare the importance of each feature family across the selection algorithms, we see that *process related* features are by far the most influential (similar to the baseline). This is followed by QQ-specificity features and *IR* related features closely behind. For the RFECV algorithm (in comparison with the others) we see an increase for *process related* features by approximately 9%. However, we also know from Section 5.2.1 that the RFECV algorithm selects 15 features on average. With fewer features in the feature space and taking into account that the four *process related* features are essentially always selected by the algorithms (according to Tables R1, R2, R3, and R4), this 9% increase is not surprising. On the other side of the spectrum, when we know that the four features of the *process related* family are often included and seeing their share than the other families, it seems that these features have a large impact. We also see that *document statistics, QQ-similarity, and QQ-term relatedness* are not contributing a lot across the algorithms. These three families ($\approx$ 28% of all features), on average for all selection algorithms combined, make up only 7.2% of feature importance. Finally, the second family in terms of importance (i.e., *QQ-specificity*) does report a quite low per-feature importance {0.74, 0.69, 3.78, 0.87}%, with an exception for RFECV. However, this feature family is the largest in terms of the number of features when we again consider the tables from Appendix R, which explains the low average importance in relation to their sum.

Table 5.10: Feature importance results in terms of the sum and the average importance (aggregated over all software projects) for **trace recommendation**, shown as percentage and divided by feature family. For both mRMR algorithms, the macro average is shown in the table. Note that for these algorithms, because of the taken macro average, the sum does not necessarily sum up to 100% for the feature families together.

			Feature selection algorithm							
	Baseline		mRMR (FCQ)		mRMR (RFCQ)		RFECV		FeatureWiz	
Feature Family	Sum	Avg	Sum	Avg	Sum	Avg	Sum	Avg	Sum	Avg
Process related	64.55	16.14	65.95	18.54	65.66	18.50	74.72	23.64	65.36	28.27
Document Statistics	1.12	0.16	2.24	0.51	1.97	0.81	0.44	0.74	1.32	0.53
Information Retrieval	6.89	0.38	14.27	1.04	8.48	1.49	9.13	4.13	9.78	2.41
QQ-Specificity	20.32	0.28	13.58	0.74	17.17	0.69	11.77	3.90	15.49	0.87
QQ-Similarity	4.2	0.23	2.16	0.55	2.99	0.51	3.28	3.78	4.73	1.26
QQ-Term Relatedness	2.93	0.24	1.80	0.63	3.73	0.72	0.67	2.23	3.33	1.06

With that we can conclude that, by comparing the baseline results to the results by the feature selection algorithms, we see similarities between the baseline and the mRMR algorithms in terms of distributions and average importance across the feature families. Next, we present the results on a feature-level to see if we can discover something there.

Table 5.11 presents the top 5 features for the baseline and selection algorithms. The table provides a better understanding of the strong performance by the *process related* feature family, as the top 3 for the columns in the table below all belong to that family. The most important aspect we learn from these results in comparison to the baseline is that none of the feature selection algorithms failed to include F4 in their feature subset, which is by far the most dominant feature. This indicates that, although different underlying techniques (i.e., based on correlation variants, MI-score, and F1 score), all produce quite similar subsets. We can clearly state that for each of the treatments, F4 (i.e., the assignee is the commiter) contributes for approximately 50% of the performance for trace recommendation.

Table 5.11: Feature importance top-5 for **trace recommendation** in terms of average gain, aggregated over all 10 software projects. The list is computed according to a threshold of features that have an importance $\geq$ 2%. For the mRMR algorithms, an average was calculated for the three values for K to compute its top-5.

			Feature selection algorithm							
Rank	Baseline		mRMR (FQC)		mRMR (RFCQ)		RFECV		FeatureWiz	
	F	Avg	F	Avg	F	Avg	F	Avg	F	Avg
1	F4	51.27	F4	48.05	F4	48.58	F4	53.21	F4	48.99
2	F3	6.86	F1	7.24	F3	8.93	F3	11.91	F1	9.54
3	F1	4.83	F3	6.20	F1	5.15	F2	5.24	F3	5.61
4	F73	2.54	F2	4.46	F2	3.00	F1	4.36	F106	1.81
5	F2	1.59	F12	3.39	F12	2.83	F28	2.99	F23	1.34

5.3.2 Trace maintenance

For trace maintenance, the results regarding feature importance are summarised in Table 5.12. When comparing the distribution of importance between the feature families of the baseline, it immediately stands out that there is quite a difference between the two trace scenarios. The features that belong to the *QQ-specificity* family contribute the most the performance, with *IR* related features on the second place. The remaining features families all have a similarly sized importance in the performance.

If we examine the different treatments, in general, we see a similar distribution to some extent across the families as for the baseline. We see that *QQ-specificity* is the most influential family, with an exception for the RFECV algorithm. For RFECV, again, *process related* features are the most dominant, with high average values for most other feature families. This is likely due to the small number of features that the algorithm selects (see Table R3). When we compare the mRMR algorithms to each other, a few differences can be seen. Especially the *IR* related features are twice as important for the FCQ scheme as for RFCQ. The algorithm using the RFCQ scheme makes up for this in other feature families (i.e., *process related, QQ-specificity, and QQ-term relatedness*). When we look at Tables R1 and R2, we can compare the different subset sizes for the mRMR algorithms individually. The tables show us that differences exist, though they are small (at most 6.66% and 2.91% difference for a specific family between the values for K for the FCQ and RFCQ scheme respectively).

Also, though not limited to trace maintenance only, whenever the subset size increases, the *QQ-specificity* family gains the most in number of features within that family. However, what we see is that when the family size increases, the size in terms of the sum (the importance of that family in relation to the total importance) decreases. This actually makes sense as the features that are added with each increase in K are of less importance (else they would have been chosen in the smaller subset size), while the denominator for calculating the sum increases.

Table 5.12: Feature importance results in terms of the sum and the average importance (aggregated over all software projects) for **trace maintenance**, shown as percentage and divided by feature family. For both mRMR algorithms, the macro average is shown in the table. Note that for these algorithms, because of the taken macro average, the sum does not necessarily sum up to 100% for the feature families together.

			Feature selection algorithm							
	Baseline		mRMR (FCQ)		mRMR (RFCQ)		RFECV		FeatureWiz	
Feature Family	Sum	Avg	Sum	Avg	Sum	Avg	Sum	Avg	Sum	Avg
Process related	6.58	1.65	10.20	2.80	13.14	3.56	42.07	13.41	12.62	4.95
Document Statistics	6.80	0.97	10.10	2.28	7.11	2.79	4.37	7.28	6.85	3.31
Information Retrieval	21.10	1.17	30.74	2.09	17.37	2.54	23.35	9.08	14.34	3.84
QQ-Specificity	49.73	0.69	38.11	1.98	45.91	1.81	24.08	10.02	45.15	2.53
QQ-Similarity	7.67	0.43	5.09	1.24	7.28	1.35	5.28	6.84	10.21	2.66
QQ-Term Relatedness	8.12	0.68	5.77	2.17	9.18	1.83	0.86	2.85	10.83	2.90

The top 15 features across the treatments for trace maintenance are displayed in Table 5.13. The table provides us a different picture compared to the uniform top 5 representation we saw for trace recommendation. We see that the average importance on a feature level are far denser, with more features having a relative importance above our 2% threshold. Again, we see that F4 is the most influential feature across the treatments (and baseline), however the remaining features are more diversified compared to trace recommendation (considering the differences between Tables 5.12 and 5.4). We also find a limited selection of features present across the algorithms, being F1 and F10, except for RFECV (which shows a different ranking compared to the others). When we compare the results to the baseline, we see that especially the mRMR (FCQ) algorithm has a similar ranking for the first seven features. That said, mRMR (RFCQ) shows the smallest set of features above our set threshold. This is also the only algorithm that selects features based on its feature importance (see Section 5.1.1).

Table 5.7 showed that most treatments have *QQ-Specificity* as the most influential family (i.e., F30-F101). However, when we consider the results on a feature level (see Table 5.13), we the family being represented only once above the threshold for our mRMR algorithms. For RFECV, we see 25% of its top-15 being represented by *QQ-Specificity* features, though not being its most dominant family. The algorithm also shows high average importance's for its first three spots, which is again likely to be caused by its average number of selected features.

Table 5.13: Feature importance top-15 for **trace maintenance**, aggregated over all 10 software projects. The list is computed according to a threshold of features that have an importance ≥ 2% (with each column complemented to the longest ranking column). For the mRMR algorithms, an average was calculated to compute the top-15.

Rank	Baseline		Feature selection algorithm							
			mRMR (FQC)		mRMR (RFCQ)		RFECV		FeatureWiz	
	F	Avg	F	Avg	F	Avg	F	Avg	F	Avg
1	F4	2.97	F4	4.11	F4	5.30	F4	19.82	F4	5.35
2	F29	2.28	F29	2.88	F3	3.20	F3	11.94	F1	4.55
3	F23	2.17	F23	2.85	F1	3.04	F2	7.50	F10	3.87
4	F10	1.68	F10	2.80	F10	2.19	F28	4.98	F17	2.46
5	F1	1.65	F1	2.74	F93	2.07	F61	3.68	F129	2.34
6	F129	1.56	F25	2.58	F83	1.93	F16	3.57	F100	2.33
7	F25	1.54	F129	2.42	F17	1.79	F25	3.11	F128	2.30
8	F61	1.49	F17	2.30	F11	1.75	F16	2.81	F93	2.24
9	F27	1.44	F11	2.29	F29	1.73	F115	2.47	F60	2.16
10	F99	1.38	F13	1.99	F100	1.66	F17	2.44	F23	2.11
11	F82	1.36	F3	1.93	F52	1.61	F99	2.36	F41	2.03
12	F17	1.31	F19	1.93	F2	1.60	F10	2.27	F62	1.69
13	F60	1.26	F17	1.91	F12	1.59	F70	2.16	F3	1.65
14	F3	1.24	F62	1.89	F79	1.56	F29	2.09	F82	1.58
15	F5	1.24	F27	1.85	F130	1.50	F73	2.03	F115	1.56

5.4 Lessons learned

Through this experiment, we tested four feature selection algorithms (the treatments) linked to the three feature selection types (i.e., filter, wrapper, and hybrid) to test their effect on the performance of the classification model. We used two variants (i.e., FCQ and RFCQ) on the mRMR algorithm as our filter method, applied recursive feature elimination as the wrapper method, and the best of both worlds through FeatureWiz (i.e., hybrid approach). For both filter methods, we tried three subset sizes (K=40, K=50, and K=60) to compare the effects of the size. We executed the experiment for two trace scenarios, being trace recommendation and trace maintenance.

For both mRMR algorithms, the effects of the different subset sizes were minor (see Section 5.2.2). This means that there was no clear winner in terms of precision, recall, or F-measure. However, a Friedman test on the results of the FCQ scheme did indicate a significant difference for trace recommendation in terms of precision. Post-hoc analyses showed that this difference was between subset sizes K=40 and K=60. The RFCQ scheme also indicated a significance for trace recommendation, this time for F_2, between K=50 and K=60. The tests did not indicate statistical differences for the trace maintenance scenario.

We then compared the performance of all treatments to each other and to our baseline (that utilised all 131 features). For **trace recommendation** (see Section 5.2.3), we saw very similar performance for F_2 scores, between a range of 2% (with an exception for RFECV). Also on a software project level, there was no configuration that always outperformed the others, although the baseline was outperformed by a reduced feature set 9 out of 10 times. Through statistical analyses, we learned that, occasionally, treatments are significantly different compared to the baseline in terms of precision and recall (positively for configurations regarding the mRMR algorithms). For **trace maintenance**, the results are closer (see Section 5.2.4). For this scenario, the baseline is always outperformed (sometimes by 0.79%, other times by 16.46%), especially by the mRMR (FCQ) algorithm. Surprisingly, the statistical tests did not find any significant difference between the baseline and the treatments across the examined metrics.

With feature importance, we wanted to learn more about the effect of certain features. As we saw in Section 5.3.1, analysing the feature importance for **trace recommendation** learned us that *process related* features are highly dominant. Across the treatments, we see an uniform selection of top 5 features, with F4 (i.e., time difference between issue resolved and commit created) being by far the most influential feature. The analysis on **trace maintenance** (see Section 5.3.2) showed a shift to *QQ-specificity* as most important feature family, though when we consider the average importance, *process related* features are still important. Also on a feature-level, the average importance is closer across the top 5 features, with mRMR (RFCQ) being the most similar to the baseline.

Chapter 6

Focus group

This chapter presents the results coming from the focus group. The steps as described in Section 2.4 are executed to come to these results. In Section 6.1, we describe the process of collecting the data. This also includes details on the included experts and steps that were taken to prepare the data for analysis. Next, we analyse the data and present the results that were derived from the data (see Section 6.2). In Section 6.3, we summarise the chapter by listing the most important findings.

6.1 Data collection

The process of collecting the data is separated into three parts. First, we describe the focus group (see Section 6.1.1). This includes details on the participants that were invited to be part of the group of experts. In Section 6.1.2, all the data that is collected during the process is specified. This section also includes details on data preparations prior to the analyses.

6.1.1 Focus group description

As mentioned in Section 2.4, the focus group consists of experts that all work within the field of low-code development. The focus group includes a variety of experts, coming from both Mendix and its parent company Siemens. Table 6.1 provides an overview of the experts that were part of the focus group. The focus group is conducted in two separate sessions. The composition of the experts to the sessions are not relevant, meaning the composition is solely based on the availability of the experts. The only restriction in place is that experts from the same team should always be part of the same session.

Table 6.1: Overview of experts and their backgrounds that were involved in the focus group

Expert	**Company**	**Team**	**Role**	**Session**
Expert 1	Mendix	I	Product manager	A
Expert 2	Mendix	I	Tech lead	A
Expert 3	Mendix	II	Principal engineer	A
Expert 4	Mendix	II	Product manager	A
Expert 5	Mendix	III	Product manager	B
Expert 6	Mendix	III	Senior engineer	B
Expert 7	Siemens	IV	Product manager	B
Expert 8	Siemens	IV	Architect	B

Both focus group sessions are conducted in an online setting. During each session, the experts are given a presentation that includes an introduction to the topic, what kind of research has been done by the researchers in the field of traceability, and what is expected from the focus group discussion. The rest of the session takes place on the online collaboration platform Miro[1]. For each session, a Miro board was prepared (see Appendix S). The Miro boards contained eight frames, with each frame having its own theme around the shared subject.

6.1.2 Data sources and preparations

As described in Section 2.4, data is collected in three ways, (1) being an audio and video recording, (2) input provided on the Miro board, and (3) input given through the survey. Each of them are discussed in more detail below, including the steps that were taken in order to prepare the data for its analysis. All three sources have been anonymised, where the experts' names are replaced with the numbering as displayed in Table 6.1.

[1]`http://miro.com`

Audio and video recording. Both sessions are recorded in terms of audio and video. All experts gave their consent for the session to be recorded. The recordings were only used to create a written representation of what has being said by transcribing the audio recording. The transcriptions of both sessions can be found in Appendix T1 (session A) and Appendix T2 (session B). The sessions are transcribed following the intelligent verbatim style [96]. This style keeps the original spoken sentences as much as possible, while allowing a filtering layer to remove stop words and grammatical changes.

Next, the transcriptions are coded to identify the main findings within the data. The coding process started with deductive coding, so that there was an initial starting point of interests. The process then allowed for additional codes to be added (i.e., inductive coding). Table 6.2 presents the main codes and corresponding concepts linked to them that were identified.

Table 6.2: Coding analysis and corresponding concepts of focus group. Note that it is possible for a reference to be categorised under more than one code.

Code	References	Concepts
Improvement	12	Opportunities and missing aspects compared to non low-code
Processes	26	Processes and practices explained
Roles	2	Identified roles within the process
Tracing	24	Trace creation and trace usage across the development process
Viewpoint	18	Opinions on processes, current or previous practices, and possibilities

Miro boards. All the input that is given by the experts through the Miro boards can be found in the online Appendix [33] (two files with .rtb extension). These files contain all the input given by the experts of both sessions and are exported by using the back-up function on the platform.

Survey. As part of the focus group session, the experts were asked to fill in a survey that contained eight statements that asked about their view (by Likert scale) on traceability. The results on this survey can be found in the online Appendix [33] as a spreadsheet. The sheet contains all answers given by the experts.

These results are processed prior to the analysis in two ways. First, one answer is adjusted based on the fact that the corresponding expert identified to misread one of the statements (also documented in Appendix T). Second, some negative-worded questions have been recoded in order to better fit other questions who were constructed by positive words.

6.2 Low-code and traceability

With the data prepared, we can take a deep dive to extract all the details. First, we go over the development process and current practices (see Section 6.2.1). In Section 6.2.2, we highlight some of the current practices and discuss some of the views on the value of traceability for their development. Finally, we take the experts view on how they feel from what the industry could benefit in the future (see Section 6.2.3).

6.2.1 Processes and practices

We discuss processes and current practices of the experts to get a picture on how traceability plays its part in their work. To start, we briefly go over the organisation within a team. Then, we cover the practices surrounding the creation of traces through commits. Thirdly, we describe aspects of traceability outside the scope of the established tracing of commits to user stories.

Team freedom versus standards

Numerous teams across the company are developing applications for both internal use as for external clients. Each team is, to some degree, free to choose how they operate the team. This freedom allow the teams to be in charge of their process, so that they can specialise the development to their needs as long as the same standards are all shared amongst the teams. Each team is responsible to follow certain guidelines that are set by the compliance department. This department is responsible for making sure the company is meeting its (ISO) certification requirements by maintaining and organising (external) audits.

Trace link establishment

One standard shared amongst the teams is that one should always include the corresponding Jira identifier when making a commit to link a revision to a user story. All experts are aware and identify this as “good practice” and to “always do it” (i.e., include the identifier). The freedom within teams does lead to different executions of the standard. Three practices were identified among the experts. Linking the commit to a user story is a requirement, but since the system does not make this a hard requirement, teams have to act when members are slacking off and make sure the standard is reinstated. Another team (Team II) took the opportunity to set up an obligation for a commit message in the form of a repository rule. This rule ensures that a commit message always includes a Jira identifier, if not, the commit would be rejected. Alternatively, a team (Team III) designed their process as such, that a commit automatically resulted in a build of a container. That container would then be, when released, ran through a pipeline. Releases that ran through these pipelines were then linked to user stories, so that one could trace the implementation of a story to a specific release build. With that, one could see the included artefacts within that release.

The process of creating trace links currently relies on the consistency of the developer who has to copy or typing over the identifier string and place it in the commit message. Since this is a manual step (while using two separate systems), typos can make their way into the process easily. To counter this risk, at some point in the release pipeline, the commit messages and Jira release scope are compared to see if all linked the stories linked in the commit messages are indeed part of that release scope. This step should be able to filter out most mistakes, assuming that a made typo does not refer to another user story that is included in that specific release as well. If that is the case, the error is likely to be found when one uses the trace, only to being unable to find the correct source document.

Tracing beyond commits

The identifier requirement as introduced above only, as intended, covers the creation of trace links between a commit and user stories. The experts were like-minded about the established traceability surrounding those specific traces, as “nothing gets done without a story” (Expert 4) and the process being “watertight” (Expert 3). However, tracing beyond commits and user stories becomes more difficult. They identified that there is no standardised way of tracing to or from documents other than Jira issues. For instance, documents that cover architectural decisions or documentation prior to putting user stories into Jira are not traced throughout development. This also means that, since there is no standardised process behind it, there is a lack of tool support. At this point, teams use custom code that they created themselves to accompany their needs in this aspect. This makes the process of tracing artefacts less transparent and manageable. The Jira identifier is also used for story branching. Expert 6 explained that, if the size of a story is larger, they have to split a story into separate branches. Then, they make sure each branch and commit message contains the identifier of the original user story.

Another type of traceability that is sometimes used in the low-code environment of Mendix is placement of comments in microflows[2]. To clarify, microflows allow you to visually express the application logic in a Mendix app to run a set of actions that can also act on certain conditions. For more complicated microflows, teams place comments inside these flows, that include information on the developer and a timestamp. One could also include Jira identifiers at this point, however this allows tracing from the microflow to the user story only. Placing these comments can be considered similar to placing comments in a full-code environment. Again, this practice is known to be performed by only one of the teams (Team III) that are involved in the focus group, making it another example of a teams freedom and lagging of a true structural process. These comments only get noticed when navigating inside the microflow as well, making it not transparent or visible from a management perspective.

[2]`https://docs.mendix.com/refguide/microflows/`

6.2.2 View on value

When the experts expressed their view on traceability (see Figure 6.1), they were unanimous on the importance of traceability (top plot). They all see the importance of traceability for the development process, with some difference on the level of agreement on the statement. One expert (Expert 6) finds it to be essential to properly fix issues if they occur. For another (Expert 4), a product manager in this case, it is more something that is needed to be done to be compliant in case of audits. On top of that, Expert 4 identified that it also contributes to be able to properly track the status of the development and communicate this to the customer.

Also, when asked on their perspective of the time and effort put into traceability (second plot), most experts were agreeing on it being worthwhile. That said, one expert (Expert 1), a product manager, was against this. As explained by the expert, it is not known how much time it actually takes. Especially for someone who does not take part in the actual trace establishment. This argument was followed up by another expert (Expert 3), who explained that if you follow the process, pursuing traceability does not take a lot of time. Only once you deviate from the process (e.g., intentionally or when mispronouncing an identifier), the thing gets annoying.

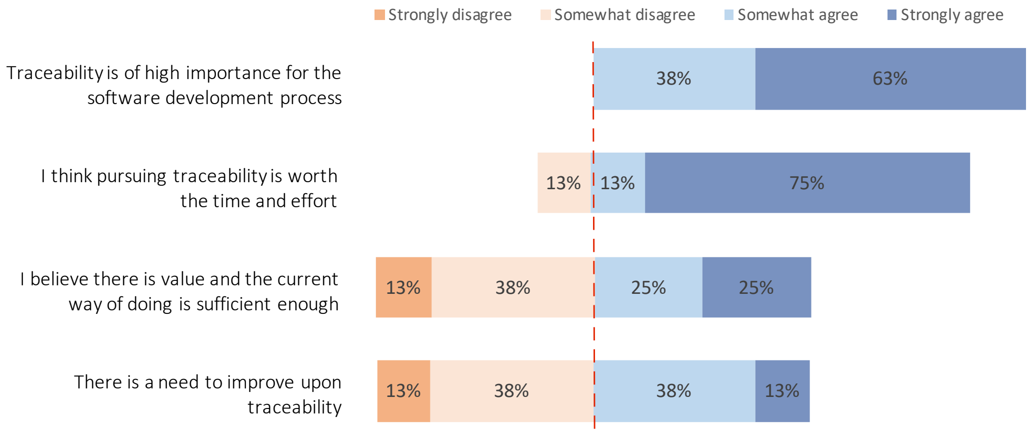


Figure 6.1: Responses on four statements on the experts view on traceability experiences in their profession.

From the first two statements, we learned that most experts think highly of having proper traceability. However, when they were asked about if they think the current way of tracing is sufficient enough, we got a more mixed picture. In Figure 6.1 (third plot), we see that the half of the experts state that the current way is not sufficient. When asked, one expert (Expert 3) was wondering if we can get more out of it and trace more artefacts besides commits and user stories. Another expert (Expert 8) feels that the current practice is less advanced compared to code-full development, where the development community has more sources regarding tracing (i.e., tool support and practical examples) at their disposal. On the other hand, the experts (Experts 1, 2, 3, and 6) who agreed to 'current practices being sufficient enough', explained this as the process delivering its value in the current form and putting more effort into this would be a waste of time, time that could be spend on the project itself. Another argument given states that their team is properly coping with the work and effort into traceability, but also feels that it is lagging in some areas.

Finally, the bottom plot in Figure 6.1 displays the opinion on the need for improvement in regard to traceability. The picture that we get on this is not surprising, considering the third statement that was discussed. Nevertheless, half of the experts identify the need for improvement. In Section 6.2.3, we discuss the aspects identified by the experts that they saw as the most interesting to enhance the still maturing industry of low-code development.

6.2.3 Maturing the industry

Compared to code-full software development, low-code is still in its early days. This means that, although learning from experiences in code-full development, the industry is still evolving and trying to fit their platform to more customers. As the platform is evolving, the process of developing applications already improved over the last couple of years. Three areas are discussed in this section, where each is found by the experts to enhance the traceability of software artefacts during development.

Tracing more

One area where the development process would benefit from enhancements is the extent to what is traced during development. As identified in Section 6.2.1, tracing user stories to commits is well in place. However, it might be beneficial for developer and product manager to be able to easily trace artefacts in documents prior to user story defining. One expert (Expert 2) found that, currently, no structured process exist that helps team members to trace artefacts between product ideation and user stories. Individual teams might have developed solutions to this, but the fact that this is not widely known in the company confirms the necessitation on more standardisation (will be discussed in an upcoming paragraph).

Besides tracing requirements to design artefacts, tracing requirements or design artefacts to test cases might be of great value as well. This type of traceability is known as *test traceability* [97, 98, 99]. As explained by Lormans & Van Deursen [98], test traceability should help to improve upon test coverage. One expert from the focus group also identified that a focus on test traceability should govern that changes made to the system are complete while also reducing the impact of defects. Making changes to the system also becomes safer, as trace links can be used to discover which tests need to be changed when revisions are made [99].

Changes made to a system are not always in terms of code or design elements, but also includes changes in terms of configuration. Configuration changes (enabling a country as an example given by Expert 8) are harder to track as this is not necessarily part of making a commit.

Automation and standardisation

Creating traceability in the current process includes some manual work (e.g., looking up the Jira identifier and placing it in a commit message), where a developer switches between Jira and their development environment (i.e., Mendix Studio Pro). Multiple experts (Experts 2 and 5) identified that (partially) automating this process helps to reduce mistakes and also creates a new standardised process across the teams, stepping away (or even include concepts) from custom created solutions. A recommendation system as covered in Section 3.3 and Figure 3.3b already provides a great improvement on the process, according to the experts. Including more automated processes also improves transparency on traceability practices and reduce the workload on the creation of traces.

Tool support and integration

When observing the results from the survey, only one expert expressed that the current tools that are available are not satisfying the needs. However, when we evaluate all input that is given during the session, we can identify urge for either better support of tools or better integration of existing tools developed by other teams or the industry. One expert (Expert 5) stated that using Jira issues requires the use of two independent systems, which always lag in terms of integration. Another expert states that it might be beneficial to have more out-of-the-box tools that work according to the same standardised processes. Then, Mendix' own developer portal does support the creation of user stories and selecting the respective user story while making a commit, out of the box. Based on the focus group sessions, this functionality is not actively used by the included experts (with one expert suggesting it works best when working on your own). This might indicate that the function is not as elaborate compared to using Jira.

Other functionality that is present in code-full environments is being able to compare your code changes side-by-side. Having two versions of a part of the system allows a developer to clarify what has indeed been changed during releases and requirement implementations. Multiple experts identified this as a missing piece in the current low-code experience, though also reporting on existing workarounds. Still, these workarounds do not support the idea of an out-of-the-box solution and might not be known as a way to go by everyone as well (i.e., less experienced developers attracted and/or targeted by low-code platforms).

Finally, we briefly discussed the two scenarios (i.e., trace recommendation and trace maintenance) that we experimented on in Chapter 5. Experts 2 expressed that, if the relevant Jira issue is recommended to the developer (especially if this is integrated into the development environment), this could be seen as a great win to the process. The view on the trace maintenance scenario is a bit different. As expert 3 mentions, a trace recommendation system will give more confidence compared to a system that is not transparently deciding on what things should be linked and what not.

6.3 Lessons learned

Through the focus group, we obtained a better understanding on nowadays practices, how teams within a low-code environment work with traceability and learn about their view on what currently is missing. The focus group involved eights experts, consisting of product managers, engineers and an architect. To collect data, two focus group sessions were held (see Section 6.1.1). From those sessions, we collected three hours of audio data, written input given on Miro boards, and data on Likert scale through a survey. All data is anonymised and the audio recordings has been transcribed.

The experts identified the value on both having freedom within development teams to organise the team as they like and having certain standards to make sure all teams follow similar good practices. This freedom and what this entails during the development of software becomes clear when different teams discuss how they act on the given standard to always include a Jira identifier string when making a commit. Within the group of eight experts, three ways of executing this activity are given, ranging from including the identifier without a check up to a complete comparison of all commit messages and user stories included in that release (see Section 6.2.1). A check on the identifier is able to filter out typos or unintentionally assigned stories, assuming the wrong identifier does not refer to a story that is included in that sprint. Besides tracing commits and user stories, other artefacts are traced to some extent as well. This includes for instance documents other than user stories or architectural decisions. However, this is not as easily traceable as for commits. This is also dependent on if a team found it worthwhile to create custom code to trace these artefacts themselves. Other elements within the platform are less traceable (e.g., microflows) and only allows for some comments to be there just as some additional information for manual tracing.

The experts were to some degree unanimous on their perspective on the importance of having proper traceability, though for some it is mainly about being compliant in case of audits rather than be of value (see Section 6.2.2). Half of the experts found that the current practice is not sufficient enough and improvements can be made (see Figure 6.1). When asked, the experts referred to tracing beyond commits and having less advanced support available compared to full-code alternatives.

Maturing the low-code industry, the experts found some areas that could be enhanced (see Section 6.2.3). First, they feel that the development process would benefit from more extensive tracing, beyond commits only. This might include test traceability, where there is a more elaborate link to the testing phase (e.g., through unit tests). Changes to system configurations are said to be hard to track, but these also might result in errors that need to be solved. Furthermore, a focus on automating processes and steering to more standardisation surrounding traceability would provide more structure and synergy across development teams, while also reduce the number of mistakes made when duplicating a Jira identifier string. Finally, the experts identified a large gap between low-code platforms and their full-code opponents. There exist more tools, examples and integration's for the latter.

Chapter 7

Discussion

In this chapter, we present the findings of our research in context of the research questions and research aim by interpreting and explaining the results we gathered in Chapters 3, 4, 5, and 6, following the methods described in Chapters 2 and **??**.

First, we briefly repeat our research questions and research aim in Section 7.1. Next, we present our key findings per utilised research method (see Section 7.2). In Section 7.3, we triangulate our (key) findings to their specific research question. Then, in Section 7.4, we go over the implications and the limitations of our research, including a section dedicated to the threats to validity. Thereafter, we present our recommendations for future work in this topic (see Section 7.5). In the end, we summarise the chapter in one concluding section (see Section 7.6).

7.1 Research questions and aim

In this section, we briefly repeat our research questions and the aim of our study as we formulated in Sections 1.2 and 1.3. Our main question is formulated as follows:

> MQ: What is the effectiveness of (semi-)automated requirements traceability in an MDD environment?

Three sub-questions were defined to support answering the main question:

> SQ-1: How do the industrial needs for software traceability in MDD environments differ to those in traditional software development?
>
> SQ-2: What can the existing state of the art techniques achieve to improve model performance aspects on a machine learning classifying algorithm?
>
> SQ-3: How to improve the state of the art in trace recovery to better address the needs of the industry?

For our research, we defined our research aim two-fold. We first want to have a clear and up-to-date identification of the problems that keep companies to fully commit to (semi-)automated traceability approaches. Secondly, we aim to show how state-of-the-art techniques can improve upon (semi-) automated approaches to traceability (i.e., upon a ML classification model).

7.2 Key findings

We present the key findings of each utilised research method below in a similar order as the chapters in this document.

Systematic mapping study

Software traceability (i.e., following the lifecycle of artefacts) has been a topic of research for many years. Artefacts can be of any kind for software traceability (e.g., requirements written down in natural text or source code). If two artefacts are connected, you can speak of a trace link between the two artefacts. Tracing artefacts is done either manual or through (semi-)automated approaches. Manual approaches have been identified as error-prone and complex, while (semi-)automated approaches are still lagging a wide adoption by the industry. MDD is a development method that is perceived as being very suitable for (semi-)automated approaches due to its own automated nature. Various approaches have been studied the last few years, with a ML approach to trace link recovery very recently. To improve upon this model, literature provides two options; (1) fine-tune the model's hyper-parameters and (2) systematically select features from a defined feature space to boost performance, improve the interpretability of the model, and reduce computational complexity.

Secondary analysis

Through the secondary analysis, we compared data on revision history from seven MDD and seven non-MDD software projects to better understand their differences and commonalities. When comparing the development process of (non-)MDD projects, we consistently recognised three patterns (e.g., increasing active participation towards the end of the projects time span) for non-MDD projects. MDD projects showed this to a less extent, with more fluctuating periods of active participation. When we compared the size of a revision (see Figure 4.3), we saw that revisions of MDD projects tend to be more often of larger size (though the results are still too mixed to make concrete conclusions). Also, when made an aggregated comparison on revision size categories between MDD and non-MDD (see Figure 4.4), both show comparable distributions. Comparing the type of revisions (see Figure 4.5) showed us that in MDD projects, a revision often includes more newly added artefacts (compared to modified or deleted artefacts). For non-MDD projects, most revisions are related to modifying existing artefacts. Finally, we analysed all commits and identified frequent coexisting term. This analysis supported our finding that MDD projects more often (relatively speaking) adds new functionality, compared to more commits dedicated to fixing things (for non-MDD projects). MDD projects also seem to show more object-oriented commits (e.g., 'add button') compared to their counterparts.

Experiment

By conducting a set of experiments, we tested four feature selection algorithms (across three types of feature selection) to assess their effect on the ML classification model for two trace scenarios (i.e., trace recommendation and trace maintenance). We analysed 400 model evaluations to see if we could detect an effect on its performance. We compared all algorithms (i.e., treatments) to our baseline, that used the complete feature space (i.e, 131 features). For the trace recommendation scenario (see Section 5.2.3), in terms of the F_2-measure, the baseline was outperformed 9 out of 10 times (although never by one algorithm on all projects). The mRMR algorithms were tested to be statistically significant compared to the baseline. For trace maintenance (see Section 5.2.4), the baseline is always outperformed, then again not on all projects by a single algorithm (sometimes by 0.79%, other times by 16.46%). Statistical tests did not find significance for a selection algorithm for the trace maintenance scenario. The next step was to analyse the feature importance across the treatments. For trace recommendation (see Section 5.3.1), the baseline and treatments showed a similar distribution across the six feature families, with process-related features being most dominant. Also on a feature-level, we see an uniform top-5, with F4 (i.e., time difference between issue resolved and commit created) always ranking highest. The feature importance for trace maintenance (see Section 5.3.2) showed a shift to QQ-specificity as most important feature family, though when we consider the average importance, process related features are still important. Also on a feature-level, the average importance is closer across the top 5 features, with mRMR (RFCQ) being the most similar to the baseline.

Focus group

The aim for the focus group was to better understand today's practices on traceability in an MDD environment. The focus group involved eights experts (product managers, engineers and architects) from Mendix and its parent company Siemens. The experts valued having both enough freedom as a development team while also caring about standards that ensure good practices among all the teams (see Section 6.2.1). This freedom may lead to different ways of executing a similar task, with some teams just performing the basics (with higher chances of making errors) while others took the opportunity to implement evaluation scripts that check the activity's execution (when speaking of creating traceability by including Jira identifiers in a commit log). The teams also trace other artefacts, besides user stories and architectural decisions, though these are not as easily traceable as commits (e.g., placing comments within a microflow as additional information on the implementation). Having proper traceability practices is valued across the different teams (see Section 6.2.2). For some it is a thing that is helpful for solving errors, while for others it is a thing to do to be compliant in case of an audit. The experts also expressed that, though commits and Jira issues are “water-tight”, tracing other artefacts is not sufficient enough to be as value as it could be. Compared to full code development, low-code does lack advanced tool support and practical examples of code problems. The low-code industry could benefit from various enhancements (see Section 6.2.3), such as including test traceability and make changes to system configurations better traceable. More automated process for standard tasks provides more structure and synergy between the different teams, while reducing the chance on possible errors.

7.3 Triangulating our findings

In this section, we discuss our findings and triangulate the gathered results from the distinct research methods (see Figure 2.2) to our three sub-questions. We cover each sub-question in ascending order in the upcoming sections, as presented in Section 7.1.

7.3.1 About industrial needs

In literature (see Section 3.1.2), we found various views on traceability practices in the industry. We saw contrary statements such as traceability being a “made up problem” and ”unnecessary evil” [62] on one side though “automated traceability would be worth its weight in gold” [67] on the other. A similar opposing view was found during our focus group (see Section 6.2.2) to some extent, suggesting that the view on traceability is quite depended on who is asked. Here we saw that some experts (Experts 3 and 8) were convinced that even more enhancement in traceability are necessary to create even more value, while Expert 4 (a product owner) does not want to spend more time on establishing traceability. Something that Cleland-Huang et al. [62] highlighted in their study as well was, when traceability is governed by standards, that this establishes a better understanding of the concept. In addition, the authors reported on two other studies [69, 70] that showed that when tracing is related to testing or regulatory compliance, practitioners are more likely to see value in traceability. Again, this is something that was understood from the experts involved in the focus group, where they see establishing traceability as “good practice” and “something we have to do to be compliant”. In that sense, we see similar perspectives across different paradigms (i.e., MDD vs. non-MDD) of software development.

Something that was mentioned by two experts (both product owners) independently, was that having proper traceability (e.g., to trace a projects status) is helpful in the communication towards customers. It is especially helpful whenever a problem in the product occurs and the customer has to be updated on the issue (e.g., time until fix). Torkar et al. [71] found similar experiences during a case study, where product development could be easily tracked leading to improved customer satisfaction. When describing the traceability practices of the studied companies, Torkar et al. found these practices to be static and requires manual input (though the company found their processes very efficient). The practices by the experts in our focus group also relied on manual tasks (e.g., including the Jira identifier in the commit message), but was found to be a part of the process that does not take a lot of time (Expert 3). Only once you go off-road, things get more messy (e.g., mistyping an identifier). This picture fits right in a statement by Spanoudakis and Zisman [13], where they link manual traceability to being error-prone and complex (e.g., once you go off-track).

The visual aspects of traceability has been studied [22, 61, 68] before. These studies identified that, carefully considering these visual aspects and making them easily accessible does support a better utilisation of such traces and contribute to a wider adoption. This is also identified by one of the experts (Expert 2) during our focus group, who stated: “you really have to see the benefits of doing this. Otherwise, I think there is a lot of resistance”. Another (Expert 5) also expressed that, currently, they have to work with two independent systems, which makes working with traces harder if there is no such integration in place, supporting the findings in literature of making them (i.e., traces) easily accessible.

When we look at differences between MDD and non-MDD (besides all that is stated in Section 7.2), non-MDD revisions seem to be more focused on project management and working on projects in parallel (see Section 4.2.3). When considering all analysed revisions, terms related to project management are always on top of the list (see Table M2), compared to MDD projects. In Table M1, we see revisions related to project management being reserved for Extra large revisions (XL-revisions represent approximately 8% of revisions in a project) only making them less frequent. During the focus group, Expert 6 explained that for their product development they make use of branching, though this really depended on the size of a user story (i.e., larger stories get split into smaller pieces).

7.3.2 Improvements of the ML model

Our mapping study provided us with various insights on how we could improve upon the existing ML classification model that was created as a result of a study by Rasiman [21]. Essentially, two groups of techniques were identified as potential candidates, being hyper parameter tuning and feature selection. The former is a technique that is often cited in research [21, 45, 82] and is by some is seen as part of the process in ML [77]. The technique aims on changing one or multiple of the classifiers' parameters (i.e., settings) to improve the models performance or computational intensity. The second technique, feature selection, can be used for similar objectives as parameter tuning, especially in the case of ML models that use a high number of (potential redundant) features. Feature selection is also an often cited technique [45, 82, 83, 84, 85] with additional potential benefits, such as reducing possible errors, computational time and chance to overfit the data. Work by Jiarpakdee et al. [83] provided us with three selection types (i.e., filter, wrapper, and hybrid), each with their own (dis)advantages.

These three types became the basis of our experiment, where we tested four selection algorithms across these three types. Two of them (i.e., mRMR (filter-based)) were created with the idea that individually good feature do not necessarily lead to the best performing set of features, and therefore the mRMR [46] algorithms also try to achieve minimal redundancy. This concept is implemented in our hybrid method (i.e., FeatureWiz) as well. When we compare the performance on the selected feature sets for both trace recommendation (see Table 5.5) and trace maintenance (see Table 5.8) we that the RFECV algorithm (which is focused only on selecting the highest scoring feature) is always outperformed by the other selected feature sets (that were selected taking redundant feature into account). When compared to the baseline in both trace scenarios, a reduced feature set is only in one occasion not able to surpass (by 0.34%) the baseline's performance. All other cases, reduced feature sets outperform the baseline for trace recommendation (up to 13.56%) and trace maintenance (up to 16.46%).

7.3.3 Improve trace recovery and link to the industry's needs

Traceability is not limited to tracing user stories and commits. The experts identified, besides tracing commits to user stories, traceability practices through branching and microflows (see Section 6.2.3). Other, currently non-traced, artefacts might be of interest as well, including code testing and configuration settings. Although in our mapping study we defined traceability as creating links between all sorts of artefacts, none of the included literature on MDD or low-code specified trace practices regarding testing. Only after it being mentioned by an expert we found additional literature [97, 98, 99] on the topic, suggesting that test traceability has yet to find its way into low-code. That said, tracing more artefacts might lead to more insight and a more transparent process if it is standardised across the company (see Section 6.2.3), however someones time is limited. Meaning, tracing more sounds good, but current practices (i.e., manual tasks) are not scalable, and as expressed by one of the experts from our focus group: "I wouldn't want to spend more effort on this in the current status". This suggests that transforming their process towards a semi-automated one, might be a necessary step. When we discussed (semi-)automated scenarios with the experts, one expert said to be a bit hesitant on a system that links traces on the background (i.e., automated traceability). When linking the artefacts manually, you (as a developer) are more aware of the link that is established, implying a semi-automated approach (e.g., suggesting presumable trace links) would be more meeting a developers' needs. Tracing more might also require a better integration in support tools. Having to use a multiple of systems, as a developer, might only lead to a more complex tracing process and easily makes things harder to use.

Our classification model and improvements through feature selection focused on the link between commit and user story. If we would like to trace more beyond commits, it might be necessary to re-train the model on these other artefacts as well (e.g., test cases). Still, in most cases, the common divisor would (probably) remains a Jira identifier (i.e., user story).

7.4 Implications and limitations

After having interpreted our results, we reflect on our research by expressing its contributions to the industrial and academic field and acknowledging its limitations. First, we discuss our study's implications where we present our academic and industrial contributions (see Section 7.4.1). Then, in Section 7.4.2, we address the limitations that we identified in our research. Finally, we assess the validity of our research in Section 7.4.3.

7.4.1 Implications

Through our literature review, we learned that traceability has been the subject of many studies in the past decade(s). Some of the literature that we reviewed focused on quantifying the status of industrial practices, while others concentrated on the role of IT and how it can support the path towards more automated approaches. It is in those areas where we noticed a gap in literature that lagged studies that combined the effort to study both modern industrial practices and experimented with state-of-the-art IT solutions.

With an increasing gain in popularity [16], low-code platforms, being a platform that aims to offer the user a complete solution for software development, could benefit any new technology to improve user experience. We saw that low-code software projects (developed through MDD) show distinctive characteristics when compared to non-MDD projects. The study by Alali, Kagdi & Maletic [43] provided a first step into characterising software revisions. We took the opportunity to compare additional non-MDD projects and to our set of MDD projects. Understanding these differences might be helpful to understand how these types of data can be utilised by the platforms (e.g., by identifying points of interest to enhance compared to traditional ways of software development).

The research by Rasiman [21] showed us what ML techniques can do for (semi-)automated traceability purposes in MDD. He constructed a feature space from various sources, resulting in a total of 131 features. To complement his research, we found it necessary to apply state-of-the-art techniques to experiment with a reduced feature space (by removing redundant or non-value adding features) and how that affects the models performance. Though the effect of feature selection techniques has been studied before [45, 82, 83, 84, 85], it is viewed as a necessary step towards a practical application in the industry.

Finally, the experts that participated in our focus group gave us the insights from an industry perspective on modern traceability practices in MDD that was missing from literature. The data showed us some similar details as we found in literature, like the use of separate (i.e., non-integrated) tools and static processes (see Section 3.1.2). However the focus group provided us with a list of interest with better quantifiable concepts from which the industry might benefit from, such as tracing more artefacts beyond commits with better integrated use of tools (to prevent a clutter of separate systems). The experts discussed various custom solutions to the field of traceability (e.g., repository rules) that, from a company perspective, other teams might benefit from as well.

7.4.2 Limitations

In this section, we address the limitations of our research and discuss the trade-offs that we made in our decision making.

To compare MDD revisions to non-MDD revisions, we considered seven software projects of both in our secondary analysis. These projects represent their respective paradigm of software development only to some extent. We can identify two limitations of our research in this. First, when we look at the time span of the considered software projects (see Tables G1 and G2), we see that the non-MDD projects cover more months of data compared to the MDD projects. This could have numerous reasons, like the researchers intentionally included open source projects that cover a large time period or maybe it is the case that MDD projects are just smaller in size. This is unknown at this point. Second, for our secondary analysis, we matched the number of non-MDD projects to the number of available MDD projects (though we had access to 33 non-MDD projects). We could have included more non-MDD software projects (which would improved the generalisability), though because of time we settled at seven projects.

For our experiment, we based our analysis on the performance results as presented in Appendices P and R. As explained in Section 5.1.2, these results were computed by running an evaluation script that computed the model's performance for our four treatments. The tables in the prior mentioned appendices display the aggregated results (i.e., the average of the 10 evaluation runs for each treatment). The evaluation parameter was set to 10 runs as this was similar to the original author [21]. However, when we randomly scan over the raw results (see experiment results in the online Appendix [33]), we see often fluctuating values (e.g., the recall on runs 5 and 10 on *Learn_1* for mRMR (FCQ) differ 12%) when we go over the 10 distinct evaluation runs. These fluctuations might influence the results (as presented in Appendices P and R). Especially when we consider how close some of our results are in Tables 5.5 and 5.8, such fluctuations could have had an impact on our analysis. A measure to counter this could have been to increase the number of evaluation rounds (e.g., 25 runs) to flatten out these fluctuations and have more reliable results. The choice was made to not go for this measure, as this would have required us to redo the experiment with roughly an 2.5 times increase in time to compute our results. Due to time limitations, this was not possible.

The focus group involved eight experts from Mendix and Siemens. During a meeting with our contact person within Mendix (prior to the focus group sessions), we explained our goal of the study and some thoughts on the participants that we are looking for to involve in our focus group (i.e., engineers and product owners). Based on that, our contact person asked around their development teams and our experts (see Table 6.1) were invited to the focus group after signing up voluntarily. However, during our focus group sessions, it was pointed out to us by the experts that we missed essential stakeholders from the company that would have been of great value for our research. These stakeholders include software development managers (SDMs) and employees from their compliance department. Missing out on valuable data could have been avoided by having either preliminary interviews to better identify the experts that we needed to recruit or by having additional interviews after our focus group. However, because of time constraints, it was not possible to set up interviews as an extension of our focus group.

In addition to the focus group, we initially planned to conduct a synthetic experiment to validate our findings from our study up to that point. The idea was to create an experimental environment where we would let the engineers from our focus group test our classification model with reduced feature set. This experiment would have been a proper first step to take the model outside its academic environment and test its performance in an industrial setting. This would also allowed us to examine different ways of visualising the data and provide high value insights in its practicality. However, this would have required too much time to develop an environment that would support our objective.

7.4.3 Threats to validity

We assess the validity of the research by using the classification scheme from both Yin [100] and Cook & Campbell [101] as recommended by Wohlin [32]. Though these scheme are designed to be used for case studies and experiments respectively, the combined aspects of validity all capture the threats that are believed to be applicable on this research. The scheme we use for this assessment concerns the threats to validity of the types construct, internal, external, reliability, and conclusion. The threats as discussed in this section, as well as possible counter measures to those threats, may include parts of the discussed validity evaluation for the experiment (see Section 2.3.2).

Construct validity reflects on the operational measures that are studied in the research and if these represent the objective of the study and its research questions [32]. From the start of the study, the research is designed to approach the research questions from various perspectives (i.e., research method triangulation). As such, we tried to mitigate the risk of mono-method bias as much as possible and this way collect multiple sources of evidence. Besides that, the used metrics and techniques used across the research methods were all taken or deducted from multiple sources found in literature (when possible), all of which are documented and referred to across this document. Still, the use of the chosen F-scores can be a point of discussion, as studied by Berry [102].

Internal validity can be threatened by influences to the variables of interest outside the knowledge of the researcher. Countering these threats must ensure that we can be certain that the possible casual relationship between a treatment and outcome is caused by the considered variable and is not influenced by a third factor. As addressed in Section 2.3.2, the experiment is primarily executed on a single computer to exclude variations in performance (in terms of the generated $F_{0.5}$ and F_2-measure) because of using different hardware and/or software. There is also a risk in terms of the data that is used for both the secondary analysis and the experiment. The involved data contains both open source data and data that is provided by an external company. This means that the researcher has no influence on the data collection process and cannot be completely certain if there might be a form of bias involved or not. The data that was provided by Mendix has been used in previous research [21, 38] and the attributes were carefully considered before use. These projects were selected by the our company contact and these projects might represent one way of working (i.e., other teams could be working differently). The use of open source data always includes a risk, as it might be difficult to estimate its trustworthiness. In the case of the SEOSS33 data [39], the authors did describe their method of collecting data thoroughly. The involved authors are believed to be trustworthy, based on their previous work [47] and studies that have used their work [21]. Nevertheless, it needs to be noted that the researcher himself was not involved in collecting the data and therefore cannot be completely certain about the validity of the data.

Some threats might have been introduced during the focus group sessions. Multiple participants were involved in each of the sessions, with always having two experts from the same development team. Obviously, the participants were aware of each others presence. This might have influenced the behaviour and/or input given by an individual compared to a situation where a participant was questioned in a one-to-one session. However, the choice was made to have a focus group for the reason to foster a discussion amongst the participants to extract high quality data and gather different points of view. During the sessions, the moderators also actively involved all the participants so that each one had the opportunity to share their expertise on most of the themes that were discussed. During one of the sessions, one expert expressed that he/she was not (too much) familiar with traceability. Though expressing his/her view on traceability in some occasions, overall the data from him/her was limited in its usability for this research.

External validity regards to threats that limit our ability to generalise to industrial practices. In this research, resources from one company (and to some extent its parent company Siemens) were utilised for the purpose of this study through a collaboration between the university and the company. This means that information in the form of data on software projects and expertise from employees were collected and used to derive academic knowledge from. The conclusions that were derived from project data on low-code platforms (as part of the secondary analysis and experiment) are solely based on the data from software projects coming from that one company. This threatens our ability to generalise this to the industry, as we cannot be sure that the data that was received from Mendix is representative to (all) other low-code development platforms that makes use of MDD. Besides the data, when considering traceability aspects in low-code through the focus group, this is again based on application development practices at Mendix. During the focus group, the participants were told they were free to describe their experiences with traceability in low-code that uses MDD beyond their practices at Mendix and Siemens to counter the threat.

The material (i.e., experimental environment, algorithms, and hardware) used in the experiment are all considered modern/state-of-the-art and are representative for industrial practice.

Threats to **reliability** validity refer to what extent the data and analysis of this research is dependent on the researcher. Or differently stated, if the research would be repeated by another researcher, would this lead to similar results? Throughout the study, all steps that were taken by the researcher were carefully documented and tools, custom code, and collected data are made publicly available. This all contributes to the research being highly repeatable and should counter the influence by the researcher that conducted the study. Note that, data relating to the MDD projects is excepted from the openly made resources due to confidentially reasons.

Conclusion validity is concerned with influences that threatens our ability to ensure that there is a statistical relationship between the treatment and outcome. As explained in Section 2.3.2, measures were taken (by conducting the experiment on 10 data sets and running 10 the evaluation 10 times) to improve the statistical power of the experiment.

7.5 Future work

We will now discuss future refinements that would complement our work and the field of traceability in (MDD) software development.

We based our research on the data on software projects that we received from our partner Mendix. This might limit the generalisability of our research, as we do not consider alternatives to low-code platforms (e.g., OutSystems[1]) that operate with MDD that might differently compared to Mendix. We feel that it is a logical step to extend our study to better understand how these different platforms operate and incorporate traceability practices.

In addition, the secondary analysis and the experiment we conducted are (partly) based on the seven software projects that we received from Mendix. In future research, we would like to expand our scope to (1) include more projects (developed through MDD) across a wider range of companies, and (2) include more non-MDD projects (not limited to the ones included in the SEOSS33 data set [39]). In our secondary analysis, we mainly focused on the characteristics of the commits and less on the data coming from their issue tracking system (i.e., Jira issues). Future research could focus more on these aspects, especially in combination with a wider scope in terms of software projects as mentioned before.

As mentioned in Section 7.4.2, we planned to study our classification models' performance outside its current academic environment. We believe that the insights that would be gathered with a large scale (synthetic) experiment would benefit the scientific community and our industrial partner. Recent work in deep learning and language models also might be interesting to study to a further extent and their applicability in software traceability.

7.6 Conclusion

This section presents the conclusions of our research. In this research, we studied the in traceability in Model-Driven development (MDD) from an academic and industrial perspective. In this section, we answer our research questions as we defined in Section 1.2. First, we present our answers to the sub-questions before answering our main question.

7.6.1 Sub-questions

Our research started with the question; *"How do the industrial needs for software traceability in MDD environments differ to those in traditional software development?"*. Through our systematic mapping study, secondary analysis, and the focus group, we found first of all many resemblances between traceability in MDD and non-MDD. Having proper traceability is for both seen as valuable for the development process. Both in literature and the experts from our focus group confirmed that the industry often relies on manual traceability practices that they identify as error-prone and complex. However, the experts from Mendix state that their process to trace commits to Jira issues is a 'water-tight' process that does not necessarily take a lot of time if rightly executed. We also found many differences in terms of the characteristics of a commit, where commits to MDD

[1]`https://www.outsystems.com`

projects are more object-oriented and often more focused on adding new functionality rather than performing fixes or exercise project management activities.

We explored ways to improve upon a state-of-the-art Machine Learning classification model for trace link recovery to answer our second sub-question; *"What can the existing state of the art techniques achieve to improve model performance aspects on a machine learning classifying algorithm?"*. We found two techniques to address this question, being parameter tuning and automated feature selection. The former has been studied by the original author of our ML model, which did not show a significant advantage over the default parameters. The latter, automated feature selection, showed promising results in our experiment, with improvements on some software projects up to 16% compared to our baseline results using the complete feature space, with the mRMR algorithms showing the greatest potential.

For our third sub-question; *"How to improve the state of the art in trace recovery to better address the needs of the industry?"*, we want to know in what way the industry can benefit from our approach to the problem through the ML model. Through the focus group, we found that more artefacts are being traced, though this is less advanced as the commits to user stories. We also found additional artefacts that the teams within Mendix could benefit from (e.g., testing and tracing prior to user story defining) if these. Tracing more requires more time invested to establish traceability, when sticking to manual traceability practices. This emphasizes even more the necessity to more towards (semi-)automated approaches to the practise. When we discussed a semi-automated and automated scenario with industry experts, they saw benefits from our semi-automated scenario (e.g., lower the chance of making mistakes). However, they were still hesitant about a complete automated approach, as this might lead to developers being less aware of the process.

7.6.2 Main question

With our sub-questions answered, we can use the insights gathered through those and provide an answer to our main question; *"What is the effectiveness of (semi-)automated requirements traceability in an MDD environment?"*.

We studied the use of (semi-)automated traceability from an academic and industrial perspective. We found, through our industrial partner Mendix (an industrial leading low-code platform), that traceability practices still mainly consist of manual activities. Their development teams establish traceability between commits and Jira issues by including identifiers in a commit message, which is something that we saw being done in non-MDD software projects as well. The process of tracing artefacts is seen, by Mendix developers, to be error-prone, though it is experienced as good practice and to always do it. We found similar practices being reported in literature, with only limited studies covering examples of automated traceability in an industrial setting. Experts from Mendix expressed that tracing more software artefacts (and making existing artefacts easier to trace) is seen as of great value for their development process and needed to enhance software development through MDD. The same experts (and in literature) said that more effort in manual tracing is not scalable.

We found multiple proposals in literature to enhance traceability (i.e., (semi-)automated traceability). These proposals included approaches, such as the comparison of meta models and the use of a ML classification model to recover trace links. We studied the latter in more detail, where we strengthened the model by applying automated feature selection and studied the effect. This often led to better performance (up to 16%) compared the to default model, making the model less computational intensive as well.

Traceability to be truly effective requires the tracing process to less depended on manual activities, or at least bring this to a minimum while also not allowing errors (i.e., faulty trace links) to be introduced. Through this study, we found that industrial practices, in its current form, are not as effective as they should be. (Semi-)automated approaches are available to the industry and they could make a difference to the developer and to their development process.

Bibliography

[1] R Ryan Nelson. It project management: Infamous failures, classic mistakes, and best practices. *MIS Quarterly executive*, 6(2), 2007.

[2] Leon A Kappelman, Robert McKeeman, and Lixuan Zhang. Early warning signs of it project failure: The dominant dozen. *Information systems management*, 23(4):31–36, 2006.

[3] Azham Hussain and Emmanuel OC Mkpojiogu. Requirements: Towards an understanding on why software projects fail. In *AIP Conference Proceedings*, volume 1761, page 020046. AIP Publishing LLC, 2016.

[4] Brenda Whittaker. What went wrong? unsuccessful information technology projects. *Information Management & Computer Security*, 1999.

[5] Axel Van Lamsweerde. Requirements engineering in the year 00: A research perspective. In *Proceedings of the 22nd international conference on Software engineering*, pages 5–19, 2000.

[6] Marcus Jakobsson. *Implementing traceability in agile software development.* Department of Computer Science, Lund University, 2009.

[7] Stale Walderhaug, Ulrik Johansen, Erlend Stav, and Jan Aagedal. Towards a generic solution for traceability in mdd. In *ECMDA Traceability Workshop (ECMDA-TW)*, pages 41–50, 2006.

[8] Linda A Macaulay. *Requirements engineering.* Springer Science & Business Media, 2012.

[9] Nikolaos Drivalos-Matragkas, Dimitrios S Kolovos, Richard F Paige, and Kiran J Fernandes. A state-based approach to traceability maintenance. In *Proceedings of the 6th ECMFA Traceability Workshop*, pages 23–30, 2010.

[10] Rashina Hoda, Norsaremah Salleh, and John Grundy. The rise and evolution of agile software development. *IEEE software*, 35(5):58–63, 2018.

[11] Pekka Abrahamsson, Outi Salo, Jussi Ronkainen, and Juhani Warsta. Agile software development methods: Review and analysis. *arXiv preprint arXiv:1709.08439*, 2017.

[12] Eran Rubin and Hillel Rubin. Supporting agile software development through active documentation. *Requirements Engineering*, 16(2):117–132, 2011.

[13] George Spanoudakis and Andrea Zisman. Software traceability: a roadmap. In *Handbook of software engineering and knowledge engineering: vol 3: recent advances*, pages 395–428. World Scientific, 2005.

[14] Hessa Abdulrahman A Alfraihi and Kevin Charles Lano. The integration of agile development and model driven development: A systematic literature review. *The 5th International Confrence on Model-Driven Engineeing and Software Development*, 2017.

[15] Brent Hailpern and Peri Tarr. Model-driven development: The good, the bad, and the ugly. *IBM systems journal*, 45(3):451–461, 2006.

[16] Ricardo Martins, Filipe Caldeira, Filipe Sa, Maryam Abbasi, and Pedro Martins. An overview on how to develop a low-code application using outsystems. In *2020 International Conference on Smart Technologies in Computing, Electrical and Electronics (ICSTCEE)*, pages 395–401. IEEE, 2020.

[17] Iván Santiago, Alvaro Jiménez, Juan Manuel Vara, Valeria De Castro, Verónica A Bollati, and Esperanza Marcos. Model-driven engineering as a new landscape for traceability management: A systematic literature review. *Information and Software Technology*, 54(12):1340–1356, 2012.

[18] Nicolas Anquetil, Uirá Kulesza, Ralf Mitschke, Ana Moreira, Jean-Claude Royer, Andreas Rummler, and André Sousa. A model-driven traceability framework for software product lines. *Software & Systems Modeling*, 9(4):427–451, 2010.

[19] Sven Wenzel. Unique identification of elements in evolving models: towards fine-grained traceability in model-driven engineering. 2010.

[20] Iván Santiago, Juan Manuel Vara, María Valeria de Castro, and Esperanza Marcos. Towards the effective use of traceability in model-driven engineering projects. In *International Conference on Conceptual Modeling*, pages 429–437. Springer, 2013.

[21] RS Rasiman. A machine learning approach for requirements traceability in model-driven development. Master's thesis, 2021.

[22] Christian Neumuller and Paul Grunbacher. Automating software traceability in very small companies: A case study and lessons learne. In *21st IEEE/ACM International Conference on Automated Software Engineering (ASE'06)*, pages 145–156. IEEE, 2006.

[23] Muhammad Atif Javed and Uwe Zdun. A systematic literature review of traceability approaches between software architecture and source code. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, pages 1–10, 2014.

[24] Gartner. Gartner forecasts worldwide low-code development technologies market to grow 23% in 2021. February 2021.

[25] Marcus Woo. The rise of no/low code software development—no experience needed? *Engineering (Beijing, China)*, 6(9):960, 2020.

[26] Roel J Wieringa. *Design science methodology for information systems and software engineering*. Springer, 2014.

[27] Davide Di Ruscio, Dimitris Kolovos, Juan de Lara, Alfonso Pierantonio, Massimo Tisi, and Manuel Wimmer. Low-code development and model-driven engineering: Two sides of the same coin? *Software and Systems Modeling*, 21(2):437–446, 2022.

[28] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. Model-driven software engineering in practice. *Synthesis lectures on software engineering*, 3(1):1–207, 2017.

[29] Nel Verhoeven. *Doing Research - The Hows and Whys of Applied Research*. 2007.

[30] Linda M Collins, Susan A Murphy, and Karen L Bierman. A conceptual framework for adaptive preventive interventions. *Prevention science*, 5(3):185–196, 2004.

[31] Barbara Kitchenham and Stuart Charters. Guidelines for performing systematic literature reviews in software engineering. 2007.

[32] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in software engineering*. Springer Science & Business Media, 2012.

[33] W. Van Oosten. Thesis Online Appendix: On the need and value of trace link recovery in Model-Driven development. `https://surfdrive.surf.nl/files/index.php/s/XMVPAjcgSUylMXJ`, 2023.

[34] Faheema Mahomed, De Wet Swanepoel, Robert H Eikelboom, and Maggi Soer. Validity of automated threshold audiometry: a systematic review and meta-analysis. *Ear and hearing*, 34(6):745–752, 2013.

[35] Riham Fliefel. *Novel strategies in the management of exposed necrotic bone and bone defects in oral and maxillofacial surgery*. PhD thesis, lmu, 2016.

[36] K Mahmood. Systematic (literature) review. `https://pt.slideshare.net/kmahmood2/data-extraction-in-systematic-literature-review`, 2021.

[37] Jeremy Dawson, Anna Rigby-Brown, Lee Adams, Richard Baker, Julia Fernando, Amanda Forrest, Anna Kirkwood, Richard Murray, Michael West, Paul Wike, et al. Developing and evaluating a tool to measure general practice productivity: a multimethod study. *Health Services and Delivery Research*, 7(13), 2019.

[38] Wouter Van Oosten, Randell Rasiman, Fabiano Dalpiaz, and Toine Hurkmans. On the effectiveness of automated tracing model changes to project issues [under review]. 2022.

[39] Michael Rath and Patrick Mäder. The seoss 33 dataset—requirements, bug reports, code history, and trace links for entire projects. *Data in brief*, 25:104005, 2019.

[40] European Medicine Agency. Workshop on data quality framework for medicines regulation - report. June 2022.

[41] Will Hillier. A guide to secondary data analysis. `https://careerfoundry.com/en/blog/data-analytics/secondary-data-analysis/`. Accessed: 09-09-2022.

[42] Hui G Cheng and Michael R Phillips. Secondary analysis of existing data: opportunities and implementation. *Shanghai archives of psychiatry*, 26(6):371, 2014.

[43] Abdulkareem Alali, Huzefa Kagdi, and Jonathan I Maletic. What's a typical commit? a characterization of open source software repositories. In *2008 16th IEEE international conference on program comprehension*, pages 182–191. IEEE, 2008.

[44] Victor R Basili and H Dieter Rombach. The tame project: Towards improvement-oriented software environments. *IEEE Transactions on software engineering*, 14(6):758–773, 1988.

[45] Rung-Ching Chen, Christine Dewi, Su-Wen Huang, and Rezzy Eko Caraka. Selecting critical features for data classification based on machine learning methods. *Journal of Big Data*, 7(1):1–26, 2020.

[46] Chris Ding and Hanchuan Peng. Minimum redundancy feature selection from microarray gene expression data. *Journal of bioinformatics and computational biology*, 3(02):185–205, 2005.

[47] Michael Rath, Jacob Rendall, Jin LC Guo, Jane Cleland-Huang, and Patrick Mäder. Traceability in the wild: automatically augmenting incomplete trace links. In *Proceedings of the 40th International Conference on Software Engineering*, pages 834–845, 2018.

[48] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine learning research*, 7:1–30, 2006.

[49] Martha Ann Carey and Jo-Ellen Asbury. *Focus group research*. Routledge, 2016.

[50] Tobias O. Nyumba, Kerrie Wilson, Christina J Derrick, and Nibedita Mukherjee. The use of focus group discussion methodology: Insights from two decades of application in conservation. *Methods in Ecology and evolution*, 9(1):20–32, 2018.

[51] Prudence Plummer-D'Amato. Focus group methodology part 1: Considerations for design. *International Journal of Therapy and Rehabilitation*, 15(2):69–73, 2008.

[52] Ismenia Galvao and Arda Goknil. Survey of traceability approaches in model-driven engineering. In *11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007)*, pages 313–313. IEEE, 2007.

[53] Balasubramaniam Ramesh, Curtis Stubbs, Timothy Powers, and Michael Edwards. Requirements traceability: Theory and practice. *Annals of software engineering*, 3(1):397–415, 1997.

[54] Stefan Winkler and Jens von Pilgrim. A survey of traceability in requirements engineering and model-driven development. *Software & Systems Modeling*, 9(4):529–565, 2010.

[55] Orlena CZ Gotel and CW Finkelstein. An analysis of the requirements traceability problem. In *Proceedings of ieee international conference on requirements engineering*, pages 94–101. IEEE, 1994.

[56] Francisco AC Pinheiro. Requirements traceability. In *Perspectives on software requirements*, pages 91–113. Springer, 2004.

[57] Senthil Karthikeyan Sundaram, Jane Huffman Hayes, Alex Dekhtyar, and E Ashlee Holbrook. Assessing traceability of software engineering artifacts. *Requirements engineering*, 15(3):313–335, 2010.

[58] Andrea De Lucia, Fausto Fasano, Rocco Oliveto, and Genoveffa Tortora. Recovering traceability links in software artifact management systems using information retrieval methods. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 16(4):13–es, 2007.

[59] Center of Excellence for Software and Systems Traceability (CoEST). Coest software traceability definition. `http://sarec.nd.edu/coest/index.html`, 2022.

[60] Jan-Martijn van der Werf. University of utrecht - introduction lecture (lecture 01) on software architecture. INFOMSWA, 2022.

[61] Jane Cleland-Huang, Carl K Chang, Gaurav Sethi, Kumar Javvaji, Haijian Hu, and Jinchun Xia. Automating speculative queries through event-based requirements traceability. In *Proceedings IEEE Joint International Conference on Requirements Engineering*, pages 289–296. IEEE, 2002.

[62] Jane Cleland-Huang, Orlena CZ Gotel, Jane Huffman Hayes, Patrick Mäder, and Andrea Zisman. Software traceability: trends and future directions. In *Future of software engineering proceedings*, pages 55–69. 2014.

[63] Giovanni Capobianco, Andrea De Lucia, Rocco Oliveto, Annibale Panichella, and Sebastiano Panichella. On the role of the nouns in ir-based traceability recovery. In *2009 IEEE 17th International Conference on Program Comprehension*, pages 148–157. IEEE, 2009.

[64] Xiaofan Chen and John Grundy. Improving automated documentation to code traceability by combining retrieval techniques. In *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*, pages 223–232. IEEE, 2011.

[65] Jonathan I Maletic, Ethan V Munson, Andrian Marcus, and Tien N Nguyen. Using a hypertext model for traceability link conformance analysis. In *Proc. of the Int. Workshop on Traceability in Emerging Forms of Software Engineering*, pages 47–54. Citeseer, 2003.

[66] Andrea Zisman, George Spanoudakis, Elena Pérez-Miñana, and Paul Krause. Towards a traceability approach for product families requirements. In *Proceedings of 3rd ICSE Workshop on Software Product Lines: Economics, Architectures, and Implications, Orlando, USA*. Citeseer, 2002.

[67] Merlin Dorfman. System and software requirements engineering. In *IEEE Computer Society Press Tutorial*. Citeseer, 1990.

[68] Thorsten Merten, Daniela Jüppner, and Alexander Delater. Improved representation of traceability links in requirements engineering knowledge using sunburst and netmap visualizations. In *2011 4th International Workshop on Managing Requirements Knowledge*, pages 17–21. IEEE, 2011.

[69] Padmalata Nistala and Priyanka Kumari. Establishing content traceability for software applications: An approach based on structuring and tracking of configuration elements. In *2013 7th International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE)*, pages 68–71. IEEE, 2013.

[70] Michael C Panis. Successful deployment of requirements traceability in a commercial engineering organization... really. In *2010 18th IEEE International Requirements Engineering Conference*, pages 303–307. IEEE, 2010.

[71] Richard Torkar, Tony Gorschek, Robert Feldt, Mikael Svahnberg, Uzair Akbar Raja, and Kashif Kamran. Requirements traceability: a systematic review and industry case study. *International Journal of Software Engineering and Knowledge Engineering*, 22(03):385–433, 2012.

[72] Gilbert Regan, Fergal McCaffery, Kevin McDaid, and Derek Flood. The barriers to traceability and their potential solutions: Towards a reference framework. In *2012 38th Euromicro Conference on Software Engineering and Advanced Applications*, pages 319–322. IEEE, 2012.

[73] Mendix inc. Mendix studio tour domain model. `https://www.mendix.com`, 2022.

[74] Frédéric Jouault. Loosely coupled traceability for atl. In *Proceedings of the European Conference on Model Driven Architecture (ECMDA) workshop on traceability, Nuremberg, Germany*, volume 91, page 2. Citeseer, 2005.

[75] Bahzad Charbuty and Adnan Abdulazeez. Classification based on decision tree algorithm for machine learning. *Journal of Applied Science and Technology Trends*, 2(01):20–28, 2021.

[76] Christopher D Manning. *Introduction to information retrieval.* Syngress Publishing,, 2008.

[77] Sotiris B Kotsiantis, Ioannis Zaharakis, P Pintelas, et al. Supervised machine learning: A review of classification techniques. *Emerging artificial intelligence applications in computer engineering*, 160(1):3–24, 2007.

[78] Jasmina Dj Novaković, Alempije Veljović, Siniša S Ilić, Željko Papić, and Tomović Milica. Evaluation of classification models in machine learning. *Theory and Applications of Mathematics & Computer Science*, 7(1):39–46, 2017.

[79] Yonghee Shin, Jane Huffman Hayes, and Jane Cleland-Huang. Guidelines for benchmarking automated software traceability techniques. In *2015 IEEE/ACM 8th International Symposium on Software and Systems Traceability*, pages 61–67. IEEE, 2015.

[80] Tom Fawcett. Roc graphs: Notes and practical considerations for researchers. *Machine learning*, 31(1):1–38, 2004.

[81] Jakub Czakon. F1 score vs roc auc vs accuracy vs pr auc: Which evaluation metric should you choose? *neptuneblog*, 2021.

[82] Chris Mills, Javier Escobar-Avila, and Sonia Haiduc. Automatic traceability maintenance via machine learning classification. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 369–380. IEEE, 2018.

[83] Jirayus Jiarpakdee, Chakkrit Tantithamthavorn, and Christoph Treude. The impact of automated feature selection techniques on the interpretation of defect models. *Empirical Software Engineering*, 25(5):3590–3638, 2020.

[84] Yi Yang, Zhigang Ma, Alexander G Hauptmann, and Nicu Sebe. Feature selection for multimedia analysis by sharing information among multiple tasks. *IEEE Transactions on Multimedia*, 15(3):661–669, 2012.

[85] Hanchuan Peng, Fuhui Long, and Chris Ding. Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on pattern analysis and machine intelligence*, 27(8):1226–1238, 2005.

[86] Gautham Krishnadas and Aristides Kiprakis. A machine learning pipeline for demand response capacity scheduling. *Energies*, 13(7):1848, 2020.

[87] Peter Sugimura and Florian Hartl. Building a reproducible machine learning pipeline. *arXiv preprint arXiv:1810.04570*, 2018.

[88] Randal S Olson and Jason H Moore. Tpot: A tree-based pipeline optimization tool for automating machine learning. In *Workshop on automatic machine learning*, pages 66–74. PMLR, 2016.

[89] LT Zeya. Evaluating classification models: Why accuracy is not good enough. `https://medium.com/mlearning-ai/evaluating-classification-models-why-accuracy-is-not-enough-abf3d9c93a69`. Accessed: 26-09-2022.

[90] Lile P Hattori and Michele Lanza. On the nature of commits. In *2008 23rd IEEE/ACM International Conference on Automated Software Engineering-Workshops*, pages 63–71. IEEE, 2008.

[91] Zhenyu Zhao, Radhika Anand, and Mallory Wang. Maximum relevance and minimum redundancy feature selection methods for a marketing machine learning platform. In *2019 IEEE international conference on data science and advanced analytics (DSAA)*, pages 442–452. IEEE, 2019.

[92] Adi Zaenul Mustaqim, Sumarni Adi, Yoga Pristyanto, and Yuli Astuti. The effect of recursive feature elimination with cross-validation (rfecv) feature selection algorithm toward classifier performance on credit card fraud detection. In *2021 International Conference on Artificial Intelligence and Computer Science Technology (ICAICST)*, pages 270–275. IEEE, 2021.

[93] Luiz Victor Ferrato Melo de Carvalho. *Machine Learning in Poultry Companies' Data. Applications and Methodologies.* PhD thesis, North Carolina State University, 2022.

[94] Huawen Liu, Jigui Sun, Lei Liu, and Huijie Zhang. Feature selection with dynamic mutual information. *Pattern Recognition*, 42(7):1330–1339, 2009.

[95] Abu-Rmileh Amjad. The multiple faces of 'feature importance' in xgboost. `https://towardsdatascience.com/be-careful-when-interpreting-your-features-importance-in-xgboost-6e16132588e7`, 2019.

[96] Summa Linguae Technologies. Types of transcription explained: Verbatim vs. intelligent vs. edited transcription. `https://summalinguae.com/data/verbatim-vs-intelligent-vs-edited-transcription/`, 2021.

[97] Abdallah Qusef. Test-to-code traceability: Why and how? In *2013 IEEE Jordan Conference on Applied Electrical Engineering and Computing Technologies (AEECT)*, pages 1–8. IEEE, 2013.

[98] Marco Lormans and Arie Van Deursen. Reconstructing requirements coverage views from design and test using traceability recovery via lsi. In *Proceedings of the 3rd international workshop on Traceability in emerging forms of software engineering*, pages 37–42, 2005.

[99] Robert White, Jens Krinke, and Raymond Tan. Establishing multilevel test-to-code traceability links. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, pages 861–872, 2020.

[100] Robert K Yin. *Case study research: Design and methods*, volume 5. sage, 2009.

[101] Thomas D Cook, Donald Thomas Campbell, and Arles Day. *Quasi-experimentation: Design & analysis issues for field settings*, volume 351. Houghton Mifflin Boston, 1979.

[102] Daniel M Berry. Empirical evaluation of tools for hairy requirements engineering tasks. *Empirical Software Engineering*, 26(6):111, 2021.

[103] Norman G Vinson and Janice Singer. A practical guide to ethical research involving humans. In *Guide to Advanced Empirical Software Engineering*, pages 229–256. Springer, 2008.

[104] Utrecht University The Institute for Information and Computing Sciences. Ethics and privacy informed consent examples. `https://www.uu.nl/en/research/institute-of-information-and-computing-sciences/ethics-and-privacy`, 2022.

[105] Neta Aizenbud-Reshef, Brian T Nolan, Julia Rubin, and Yael Shaham-Gafni. Model traceability. *IBM Systems Journal*, 45(3):515–526, 2006.

[106] Hongyu Kuang, Jia Nie, Hao Hu, Patrick Rempel, Jian Lü, Alexander Egyed, and Patrick Mäder. Analyzing closeness of code dependencies for improving ir-based traceability recovery. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 68–78. IEEE, 2017.

[107] Chris Mills, Javier Escobar-Avila, Aditya Bhattacharya, Grigoriy Kondyukov, Shayok Chakraborty, and Sonia Haiduc. Tracing with less data: active learning for classification-based traceability link recovery. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 103–113. IEEE, 2019.

Appendices

A Minimum attribute list

Table A: Minimum attribute list. We use green for attributes present in both data sets and orange for attributes that are present in only one of them. The most right column (i.e., Attribute name) is the list of attributes that must be present in any data set to be of use, together with the collective name that will be used in this study.

Mendix	**SEOSS 33**	**Attribute name**
Issue Tracking System		
Author	Author	Author
Created (date)	Created (date)	Created (date)
Description	Description	Description
Issue key	Issue key	Issue key
Resolved (date)	Resolved (date)	Resolved (date)
Summary	Summary	Summary
Updated (date)	Updated (date)	Updated (date)
Comment	Comment	Comment
	Status	
	Resolution	
	Priority	
Version Control System		
Author	Author	Author
Date	Commit date	Date
Log	Message	Message
Revision number	Commit hash	Commit id
Metadata		

B Focus group discussion

B1 Interview protocol

Session details
Moderator 1: Wouter van Oosten
Moderator 2: Fabiano Dalpiaz
Location: online
Date: 05 December 2022 / 06 December 2022

1. Introduction
Start the session with a presentation about the submitted paper and the results that were found in that study. Next, provide background information highlighting the subject of the Masters' thesis and what the objectives are that we aim to accomplish through this focus group.

2. Expert background
Next, we start with an introduction round of the participants so that we can get a picture of each others backgrounds.

- How would you define or explain traceability to one another? (establish common ground)

3. Traceability in practice
We want to establish a general overview of traceability practices during the development of software applications.

- Are you familiar with traceability and in what way are you using or creating it?
- How are trace links created during daily development and is that something that is actively pursued or encouraged?
- Are traces often used to trace back to certain requirements (or is it something just good to have in case you need it)?

4. View on traceability
We would like to know your thoughts about traceability. What are your experiences (the good and the bad).

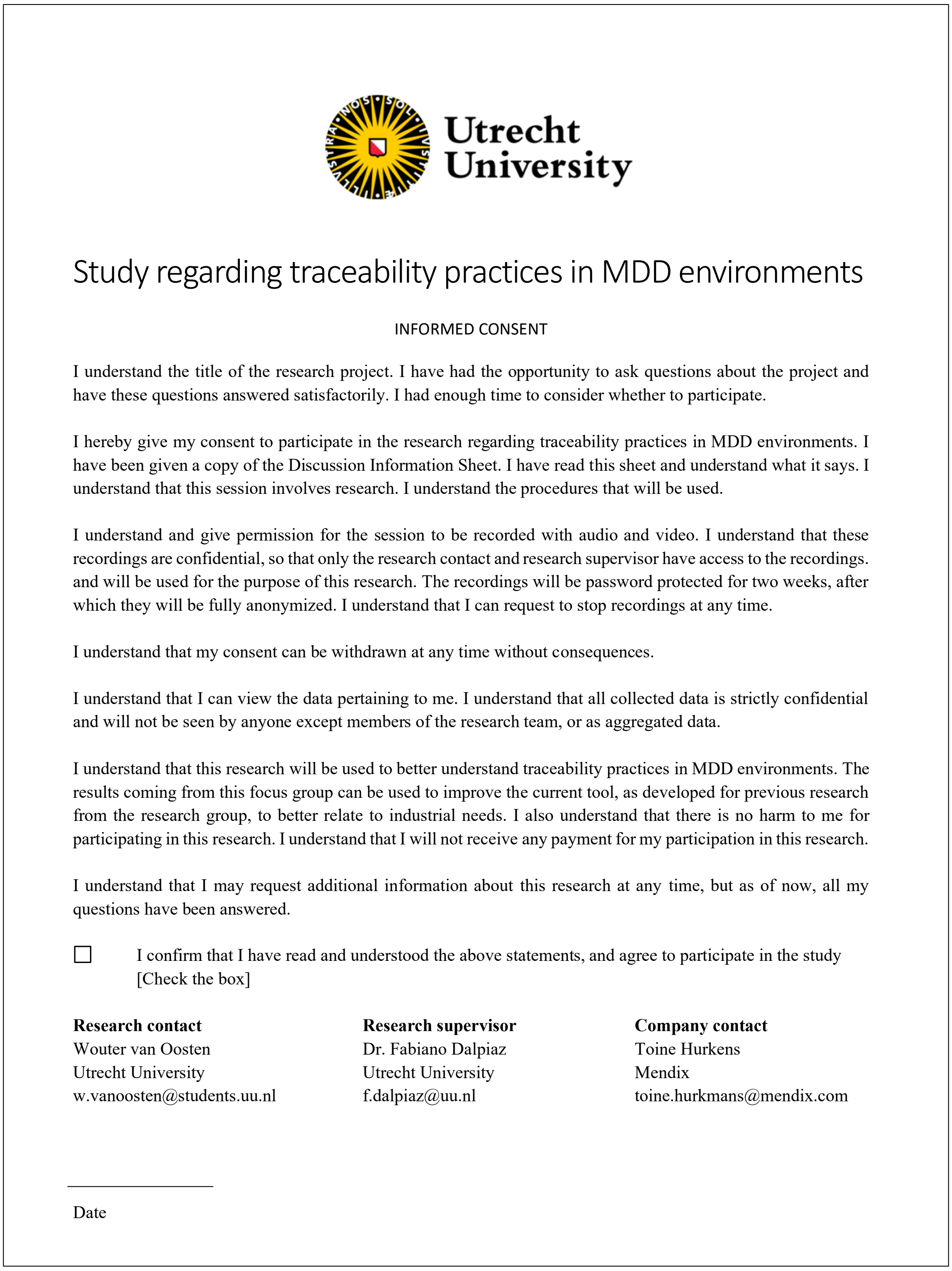
- Traceability is sometimes described as an unnecessary evil. What do you think about this?
- What kind of issues are you experiencing regarding traceability (during development or more generally speaking)?
- Are the activities to establish traceability considered to be time consuming?

5. Making traceability more valuable
We like to get an idea in what way traceability can better support developers.

- What are your thoughts on the two traceability scenarios we studied in our research (briefly explain the two scenarios).
- Could you see what benefits the improvements in those aspects would provide (for example for bug fixes) or are we missing something?
- Could you think of other scenarios that would provide value to a developer (the way of presenting trace links, through visualisations or providing lists)

6. Room for suggestions by participants
Room for additional topics relevant to this research as suggestions by participants.

B2 Informed consent form



Utrecht University

Study regarding traceability practices in MDD environments

INFORMED CONSENT

I understand the title of the research project. I have had the opportunity to ask questions about the project and have these questions answered satisfactorily. I had enough time to consider whether to participate.

I hereby give my consent to participate in the research regarding traceability practices in MDD environments. I have been given a copy of the Discussion Information Sheet. I have read this sheet and understand what it says. I understand that this session involves research. I understand the procedures that will be used.

I understand and give permission for the session to be recorded with audio and video. I understand that these recordings are confidential, so that only the research contact and research supervisor have access to the recordings. and will be used for the purpose of this research. The recordings will be password protected for two weeks, after which they will be fully anonymized. I understand that I can request to stop recordings at any time.

I understand that my consent can be withdrawn at any time without consequences.

I understand that I can view the data pertaining to me. I understand that all collected data is strictly confidential and will not be seen by anyone except members of the research team, or as aggregated data.

I understand that this research will be used to better understand traceability practices in MDD environments. The results coming from this focus group can be used to improve the current tool, as developed for previous research from the research group, to better relate to industrial needs. I also understand that there is no harm to me for participating in this research. I understand that I will not receive any payment for my participation in this research.

I understand that I may request additional information about this research at any time, but as of now, all my questions have been answered.

☐ I confirm that I have read and understood the above statements, and agree to participate in the study [Check the box]

Research contact	**Research supervisor**	**Company contact**
Wouter van Oosten	Dr. Fabiano Dalpiaz	Toine Hurkens
Utrecht University	Utrecht University	Mendix
w.vanoosten@students.uu.nl	f.dalpiaz@uu.nl	toine.hurkmans@mendix.com

Date

Figure B2: Informed consent form. Adapted from [103, 104]

C Summary of found studies in the literature review

Table C: Found studies in literature review

Ref. ID	Author	Title	Year	Publication	Country
LRS01	Sundaram et al. [57]	Assessing traceability of software engineering artifacts	2010	Requirements engineering, Springer	United Kingdom
LRS02	Aizenhub-Reshef et al. [105]	Model traceability	2006	IBM Systems Journal, IBM	United States
LRS03	Galvao & Goknil [52]	Survey of Traceability Approaches in Model-Driven Engineering	2007	IEEE	The Netherlands
LRS04	Cleland-Huang et al. [62]	Software traceability: Trends and future directions	2014	Association for Computing Machinery	United States
LRS05	Ramesh et al. [53]	Requirements traceability: Theory and practice	1997	Unknown	United States
LRS06	Winkler & Von Pilgrim [54]	A survey of traceability in requirements engineering and model-driven development	2010	Software & Systems Modeling	Germany
LRS07	Santiago et al. [17]	Model-Driven Engineering as a new landscape for traceability management: A systematic literature review	2012	Information and Software Technology	Spain
LRS08	Wenzel [19]	Unique Identification of Elements in Evolving Models: Towards Fine-Grained Traceability in Model-Driven Engineering	2010	University of Siegen	Germany
LRS09	Chen & Grundy [64]	Improving Automated Documentation to Code Traceability by Combining Retrieval Techniques	2011	Association for Computing Machinery	United States
LRS10	De Lucia et al. [58]	Recovering Traceability Links in Software Artifact Management Systems using Information Retrieval Methods	2007	Association for Computing Machinery	United States
LRS11	Neumuller & Grunbacher [22]	Automating Software Traceability in Very Small Companies: A Case Study and Lessons Learned	2006	IEEE	United States
LRS12	Torkar et al. [71]	Requirements traceability: a systematic review and industry case study	2012	Int. Journal of Software- and Knowledge Eng."	Singapore
LRS13	Kuang et al. [106]	Analyzing Closeness of Code Dependencies for Imporoving IR-based Traceability Recovery	2017	IEEE	Austria
LRS14	Spanoudakis & Zisman	Software Traceability: A Roadmap	2005	World Scientific	United Kingdom
LRS15	Rath et al.	Traceability in the Wild: Automatically Augmenting Incomplete Trace Links	2018	International Conference on Software Engineering	Sweden
LRS16	Merten, Juppner & Delater	Improved Representation of Traceability Links in Requirements Engineering Knowledge using Sunburst and Netmap Visualisations	2011	IEEE	Italy
LRS17	Cleland-Huang et al.	Automating Speculative Queries through Event-based Requirements Traceability	2002	IEEE	Germany
LRS18	Gotel & Finkelstein	An Analysis of the Requirements Traceability Problem	1994	IEEE	United States
LRS19	Pinheiro	Requirements traceability	2004	Springer	United States
LRS20	Dorfman & Theyer	System and software requirements engineering	1990	IEEE	United States
LRS21	Maletic et al.	Using a Hypertext Model for Traceability Link Conformance Analysis	2003	Computer Sience	United States
LRS22	Zisman et al.	Towards a Traceability Approach for Product Families Requirements	2002	ICSE	United Kingdom
LRS23	Capobianco et al.	On the role of the nouns in IR-based traceability recovery	2009	IEEE	Italy
LRS24	Brambilla, Cabot & Wimmer	Model-driven software engineering in practice	2017	Morgan & Claypool Publishers	United States
LRS25	Novakovic et al. [78]	Evaluation of Classification Models in Machine Learning	2017	Theory Appl. Math. Comput. Sci	Romania
LRS26	Chen et al. [45]	Selecting critical features for data classification based on machine learning methods	2020	Journal of Big Data	Taiwan
LRS27	Kotsiantis [77]	Supervised Machine Learning: A Review of Classification Techniques	2007	Informaticanbsp;	Greece
LRS28	Mills et al. [107]	Tracing with Less Data: Active Learning for Classification-Based Traceability Link Recovery	2019	IEEE ICSME	United States
LRS29	Mills, Escobar-Avila & Haiduc [82]	Automatic Traceability Maintenance via Machine Learning Classification	2018	IEEE ICSME	United States
LRS30	Rasiman [21]	A Machine Learning Approach for Requirements Traceability in Model-Driven Development	2021	Utrecht University	The Netherlands
LRS31	Yang et al. [84]	Feature Selection for Multimedia Analysis by Sharing Information amoung Multiple Tasks	2013	IEEE	United States
LRS32	Manning [76]	Introduction to information retrieval	2008	Syngress Publishing	United States
LRS33	Peng, Long & Ding [85]	Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy	2005	IEEE	United States
LRS34	Jiarpakdee et al. [83]	The Impact of Automated Feature Selection Techniques on the Interpretation of Defect Models	2020	Empirical Software Engineering	Australia
LRS35	Charbuty & Abdulazeez [75]	Classification Based on Decision Tree Algorithm for Machine Learning	2021	Journal of Applied Science and Technology Trends	Iraq
LRS36	Shin, Hayes & Cleland-Huang [79]	Guidelines for Benchmarking Automated Software Traceability Techniques	2015	Int. Symposium on Software and Systems Traceability	United States
LRS37	Ding & Peng [46]	Minimum redundancy feature selection from microarray gene expression data	2005	Journal of Bioinformatics and Computational Biology	United States
LRS38	Krishnadas & Kiprakis [86]	A Machine Learning Pipeline for Demand Response Capacity Scheduling	2020	Energies	United Kingdom
LRS39	Olson & Moore [88]	TPOT: A Tree-based Pipeline Optimization Tool for Automating Machine Learning	2016	JMLR: Workshop and Conference Proceedings	United States
LRS40	Sugimura & Hartl [87]	Building a Reproducible Machine Learning Pipeline	2018	arXiv	Unkown

D Test-retest study selection

Table D: Test-retest procedure results

Ref. ID	Status	Additional comment
LRS01	Keep	Study provides details on traceability and includes various techniques for this purpose.
LRS02	Keep	Study provides a detailed analysis on traceability in both non-MDD and MDD software development. Though being older than 15 years, the paper is cited by others included in the review [17, 19, 52, 54] and is therefore assumed to be relevant.
LRS06	Keep	Paper describes common terminology and helps to better conceptualise traceability in MDD with the use of visualisations.
LRS07	Keep	Study identified MDD as new landscape with opportunities for traceability in software development, which is relevant and inline with the selection criteria.
LRS08	Keep	Thesis on traceability in MDD with information on MDD in general as well.
LRS10	Keep	Study provides various insights in traceability recovery, IR approaches to traceability and implementation of traceability tools.
LRS11	Keep	Study provides a case study where automated software traceability has been implemented in a software company.
LRS12	Keep	Study provides insight in industrial practices in requirements traceability.
LRS16	Keep	Study provides insight in how visualisation can be part of a wider addoption of traceability.
LRS19	Keep	Article is often cited [21, 54] for its the definition of requirements traceability.
LRS26	Keep	Book that covers every aspect of MDD to get a good understanding of it.
LRS27	Keep	Paper explains aspects of supervised learning and goes in depth about algorithms (decision trees) that are studied in this research.
LRS29	Keep	Touches briefly upon feature selection techniques.
LRS31	Keep	Study compared six different feature selection methods to their own proposed algorithm.
LRS34	Keep	Study goes in depth in three types of feature selection and does an elaborate comparison on 11 feature selection techniques.
LRS39	Keep	The authors developed a tool to optimise ML pipelines, though the article itself does not provide that many details. Still, it shares some of the possibilities in this field.

E Literature quality assessment

Table E1: Description of literature quality assessment

Item	Description	Grade
1	**Title**	0 = inaccurate/not concise 1 = accurate and concise
2	**Abstract** Summary of introduction, methods, results and discussion	0 = clearly inaccurate 1 = possibly accurate 2 = clearly accurate
3	**Introduction** Background information and relevance	0 = clearly insufficient 1 = possibly sufficient 2 = clearly sufficient
4	**Introduction** Objective and/or research question(s)	0 = not clear 1 = clear
5	**Methods** Clear description and relevant to objective and research question(s)	0 = clearly insufficient 1 = possibly sufficient 2 = clearly sufficient
6	**Methods** Multiple research methods[2]	0 = no 1 = yes
7	**Methods** Statistical method details	0 = clearly inadequate 1 = possibly adequate 2 = clearly adequate
8	**Results** Summary of data characteristics	0 = clearly insufficient 1 = possibly sufficient 2 = clearly sufficient
9	**Results** Analysis of the data	0 = clearly insufficient 1 = possibly sufficient 2 = clearly sufficient
10	**Results** Outcome of analysis	0 = clearly inadequate 1 = possibly adequate 2 = clearly adequate
11	**Discussion** Interpretation/scientific implications	0 = clearly inadequate 1 = possibly adequate 2 = clearly adequate
12	**Discussion** Generalisability	0 = clearly insufficient 1 = possibly sufficient 2 = clearly sufficient

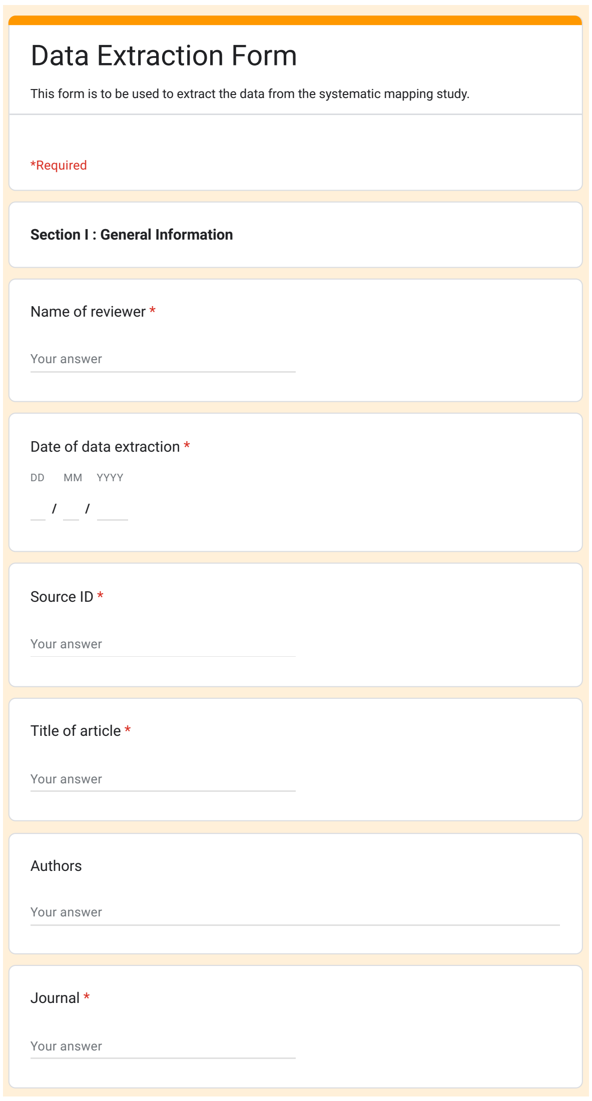
[2]With an exception made for literature studies that often limit themselves to literature only.

Table E2: Literature quality assessment

	Items														
Ref. ID	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**T(21)**	**Score**	**Quality Coeff.**
LRS01	1	1	2	1	1	1	2	2	1	2	1	1	**16**	**0.76**	**Average**
LRS02	0	1	0	0	0	1	0	0	0	0	1	1	**4**	**0.19**	**Poor**
LRS03	1	1	1	1	1	1	0	1	1	1	1	1	**11**	**0.52**	**Average**
LRS04	1	1	2	1	1	1	0	1	2	2	2	2	**16**	**0.76**	**Excellent**
LRS05	1	1	1	1	1	1	0	1	2	2	1	1	**13**	**0.62**	**Average**
LRS06	1	2	2	1	1	1	0	1	2	2	2	2	**17**	**0.81**	**Excellent**
LRS07	1	2	2	1	2	1	0	2	2	2	2	2	**19**	**0.90**	**Excellent**
LRS08	1	2	2	1	1	1	0	2	2	2	1	1	**16**	**0.76**	**Average**
LRS09	1	2	2	1	2	1	0	2	2	2	1	1	**17**	**0.81**	**Excellent**
LRS10	1	2	2	1	1	1	1	2	2	2	2	1	**18**	**0.86**	**Excellent**
LRS11	1	2	2	1	1	1	0	1	1	1	1	1	**13**	**0.62**	**Average**
LRS12	1	2	2	1	2	1	0	1	2	2	2	1	**17**	**0.81**	**Excellent**
LRS13	1	2	2	1	2	1	2	2	2	2	1	1	**19**	**0.90**	**Excellent**
LRS14	1	2	2	1	1	1	0	1	2	1	2	1	**15**	**0.71**	**Average**
LRS15	1	2	2	1	2	1	2	2	2	2	2	2	**21**	**1.00**	**Excellent**
LRS16	1	2	2	1	1	1	0	0	1	1	1	1	**12**	**0.57**	**Average**
LRS17	1	1	2	1	2	1	0	1	1	1	1	1	**13**	**0.62**	**Average**
LRS18	1	2	1	0	1	1	1	1	2	2	1	2	**15**	**0.71**	**Average**
LRS19	1	1	2	0	0	1	0	0	1	1	2	2	**11**	**0.52**	**Average**
LRS20[3]															
LRS21	1	2	2	1	2	0	0	1	2	1	1	1	**14**	**0.67**	**Average**
LRS22	1	0	1	1	2	1	0	1	1	1	1	1	**11**	**0.52**	**Average**
LRS23	1	2	2	1	2	1	2	2	2	2	2	1	**20**	**0.95**	**Excellent**
LRS24[2]															
LRS25	1	2	2	1	1	1	0	2	1	2	2	2	**17**	**0.81**	**Excellent**
LRS26	1	2	2	1	0	1	0	2	2	2	2	2	**17**	**0.81**	**Excellent**
LRS27	1	1	1	1	1	1	0	1	2	2	2	2	**15**	**0.71**	**Average**
LRS28	1	1	2	1	2	1	2	2	2	2	2	1	**19**	**0.90**	**Excellent**
LRS29	1	2	2	1	2	1	2	2	2	2	2	2	**21**	**1.00**	**Excellent**
LRS30	1	2	2	1	2	1	2	2	2	2	2	1	**20**	**0.95**	**Excellent**
LRS31	1	1	2	1	0	1	1	1	2	2	2	1	**15**	**0.71**	**Average**
LRS32[2]															
LRS33	1	2	2	1	2	1	1	2	2	2	2	1	**19**	**0.90**	**Excellent**
LRS34	1	2	2	1	2	1	2	2	2	2	2	2	**21**	**1.00**	**Excellent**
LRS35	1	2	1	1	1	1	0	1	2	1	2	1	**14**	**0.67**	**Average**
LRS36	1	2	2	1	2	1	1	2	2	2	2	1	**19**	**0.90**	**Excellent**
LRS37	1	1	2	1	1	1	0	2	2	2	2	2	**17**	**0.81**	**Excellent**
LRS38	1	2	2	1	2	1	0	1	2	2	1	1	**16**	**0.76**	**Average**
LRS39	1	1	1	1	1	2	1	2	2	2	2	1	**16**	**0.76**	**Average**
LRS40	1	2	1	1	1	1	0	0	1	2	2	1	**13**	**0.62**	**Average**
Cat. score	**36**	**60**	**64**	**34**	**49**	**36**	**23**	**50**	**63**	**63**	**60**	**49**			
Max cat. score	**41**	**82**	**82**	**41**	**82**	**41**	**82**	**82**	**82**	**82**	**82**	**82**			

[3]Source refers to a scholarly book, whereas the quality criteria aims to assess scientific studies only. The source is accepted as the book is used to clarify mostly general aspects of its respective field of study.

F Data Extraction Form


Data Extraction Form

This form is to be used to extract the data from the systematic mapping study.

*Required

Section I : General Information

Name of reviewer *

Your answer

Date of data extraction *

DD MM YYYY

__ / __ / ____

Source ID *

Your answer

Title of article *

Your answer

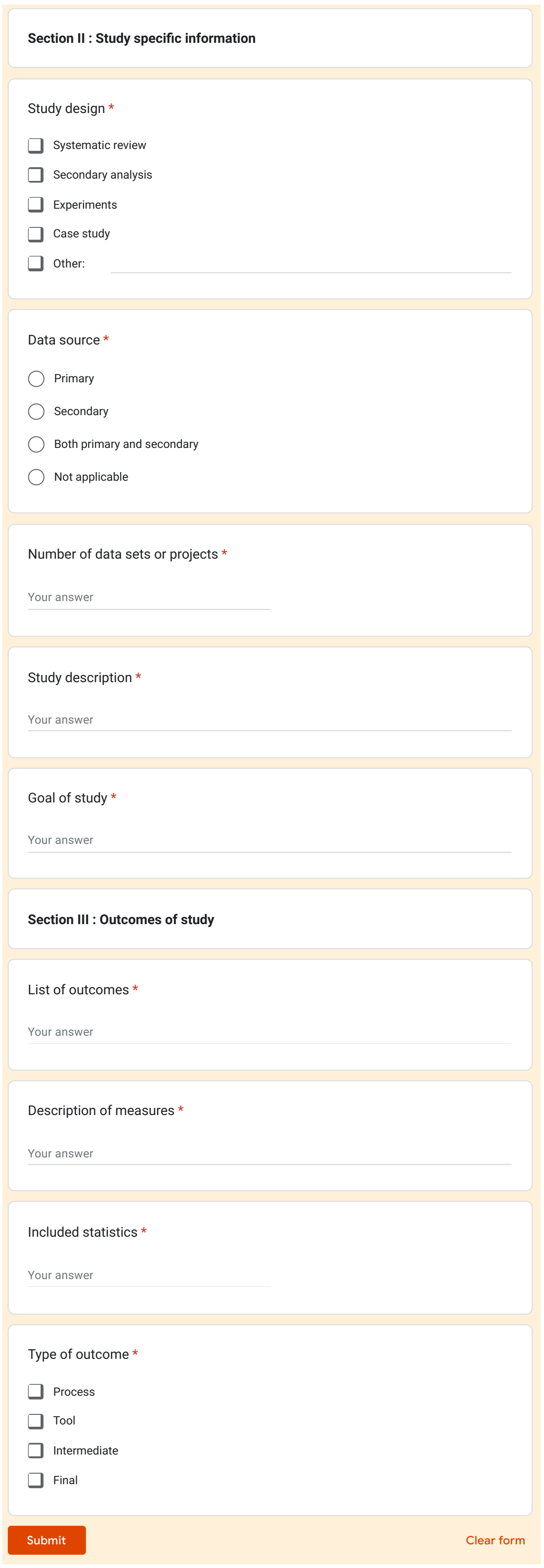
Authors

Your answer

Journal *

Your answer

Figure F1: Data Extraction Form section I



Section II : Study specific information

Study design *

- ☐ Systematic review
- ☐ Secondary analysis
- ☐ Experiments
- ☐ Case study
- ☐ Other: ____

Data source *

- ○ Primary
- ○ Secondary
- ○ Both primary and secondary
- ○ Not applicable

Number of data sets or projects *

Your answer

Study description *

Your answer

Goal of study *

Your answer

Section III : Outcomes of study

List of outcomes *

Your answer

Description of measures *

Your answer

Included statistics *

Your answer

Type of outcome *

- ☐ Process
- ☐ Tool
- ☐ Intermediate
- ☐ Final

Submit Clear form

Figure F2: Data Extraction Form section II & III

G Source outlines for secondary analysis

G1 Mendix data sets outline

Table G1: Trace details for all 10 software projects of the acquired Mendix data set. For data sets *Control* and *Learn*, partitions were created to be used for the experiment in Chapter 5. For the secondary analysis (Chapter 4), only the complete set that is considered. Table adapted from [38].

Project	Time period	Issues	Revisions	Revision traces to			
				Untraced		Traced	
	Months			Count	%	Count	%
Company	11	29	626	153	24.44	473	75.56
Control	17	326	2,895	905	31.26	1990	68.74
Control_1	8	111	1,629	485	29.77	1144	70.23
Control_2	9	116	768	275	35.81	493	64.19
Control_3	5	99	498	145	29.12	353	70.88
Data	13	58	818	236	28.85	582	71.15
Learn	10	446	1542	316	20.49	1226	79.51
Learn_1	9	278	1,239	211	17.03	1028	82.97
Learn_2	3	168	303	105	34.65	198	65.35
Portfolio	9	97	1,056	254	24.05	802	75.95
Service	3	173	2,930	1,435	48.98	1495	51.03
Store	34	634	713	508	71.25	205	28.75
Average	*14*	*176*	*1,058*	*381*	*34.40*	*677*	*65.50*

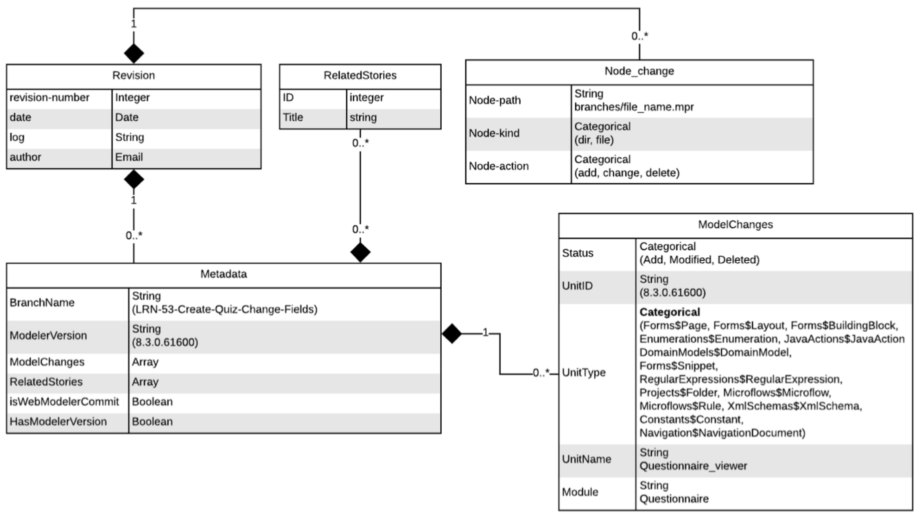


Figure G1: Structure of the Mendix data sets [21, p. 39]

G2 SEOS33 data sets outline

Table G2: Trace details for all 33 software projects of the acquired SEOSS33 data set. Table adapted from [39].

Project	Time period	Issues	Revisions	Revision traces to			
				Untraced		Traced	
	Months			Count	%	Count	%
Archiva	162	1,929	8,006	7,496	93.63	510	6.37
Axis2	164	5,796	31,114	30,630	98.44	484	1.56
Cassandra	106	13,965	23,592	20,229	85.75	3,363	14.25
Derby	160	6,969	8,156	4,627	56.73	3,529	43.27
Drools	146	5,103	11,117	10,484	94.31	633	5.69
Erray	99	1,060	7,645	7,605	99.48	40	0.52
Flink	43	8,100	12,419	10,524	84.74	1,895	15.26
Groovy	173	8,137	12,378	11,391	92.03	987	7.97
Hadoop	150	39,086	27,776	4,150	14.94	23,626	85.06
Hbase	131	19,247	14,331	6,637	46.31	7,694	53.69
Hibernate	172	11,971	8,173	5,063	61.95	3,110	38.05
Hive	113	18,025	11,179	2,069	18.51	9,110	81.49
Hornetq	158	3,286	11,121	10,389	93.42	732	6.58
Infinispan	106	8,422	10,654	8,304	77.94	2,350	22.06
Izpack	108	1,337	5,489	5,416	98.67	73	1.33
Jbehave	162	1,234	3,282	3,278	99.88	4	0.12
Jboss-T,-M,	145	2,887	2,204	1,390	63.07	814	36.93
Jbpm	159	10,397	4,919	2,708	55.05	2,211	44.95
Kafka	78	6,219	4,426	2,727	61.61	1,699	38.39
Keycloak	54	5,523	10,106	8,698	86.07	1,408	13.93
Log4J	117	2,114	9,792	9,304	95.02	488	4.98
Lucene	197	17,329	28,995	24,072	83.02	4,923	16.98
Maven	183	5,073	10,315	8,344	80.89	1,971	19.11
Pig	123	5,234	3,134	1,655	52.81	1,479	47.19
Railo	101	3,326	3,990	3,945	98.87	45	1.13
Resteasy	119	1,649	3,684	3,404	92.40	280	7.60
Seam2	147	5,031	11,309	10,356	91.57	953	8.43
Spark	93	22,205	20,829	12,759	61.26	8,070	38.74
Switchyard	86	3,010	2,928	2,046	69.88	882	30.12
Teeid	196	4,899	7,266	6,543	90.05	723	9.95
Weld	115	2,518	8,086	7,421	91.78	665	8.22
Wildfly	203	24,566	36,711	25,678	69.95	11,033	30.05
Zookeeper	116	2,907	1,600	634	39.63	966	60.38
Average	*133*	*8,441*	*11,416*	*8,484*	*75.75*	*2,932*	*24.25*

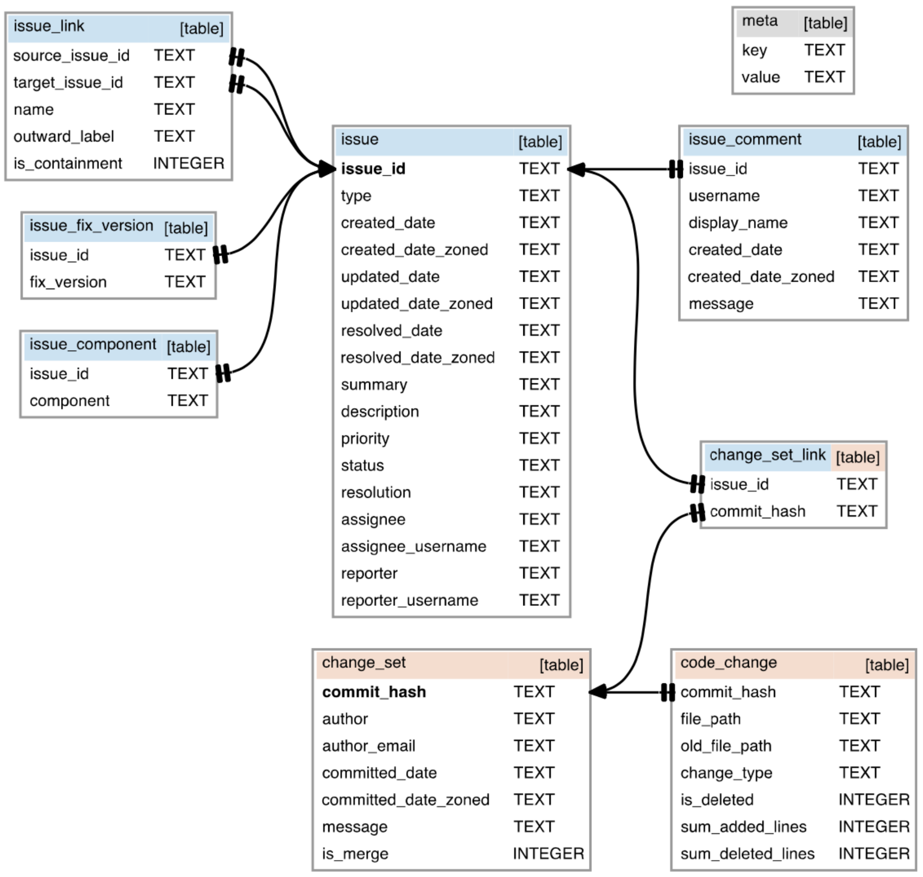


Figure G2: Database schemes of the SEOS33 data sets [39, p. 9].

H Distributions of revisions and JIRA issues

H1 Mendix distributions of revisions and JIRA issues

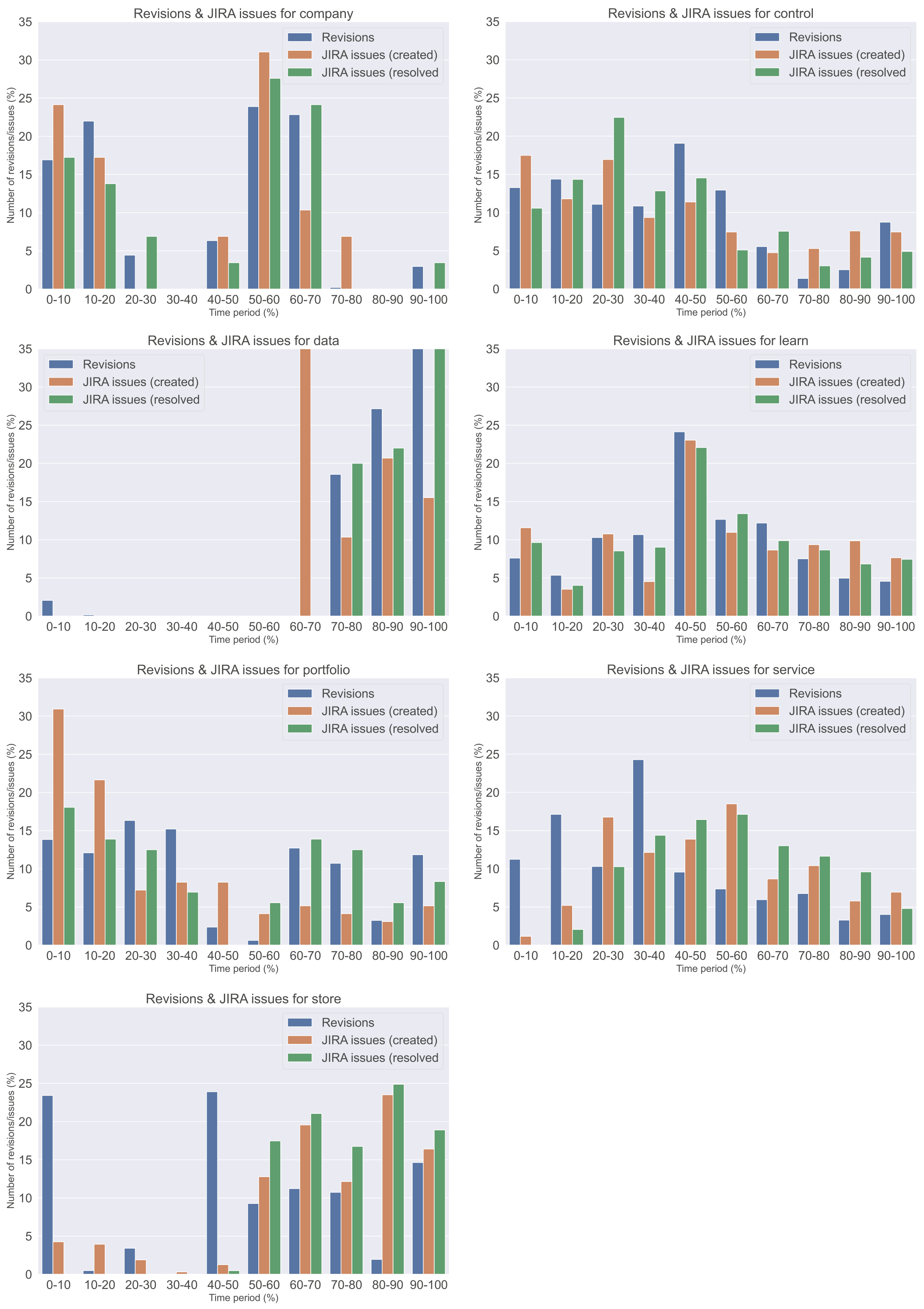


Figure H1: Data distribution visualisation of all Mendix software projects. The horizontal axis represent the projects time period as percentage in steps of 10%. The relative number of revisions and JIRA issues are displayed on the vertical axis. Note that, for project *Data*, the data expands beyond the graphs vertical axis. This is done intentionally for readability purposes. The values outside the range are {Revisions: 48.5%, JIRA created: 51.7%, JIRA resolved: 54.0%}.

H2 SEOSS33 distributions of revisions and JIRA issues

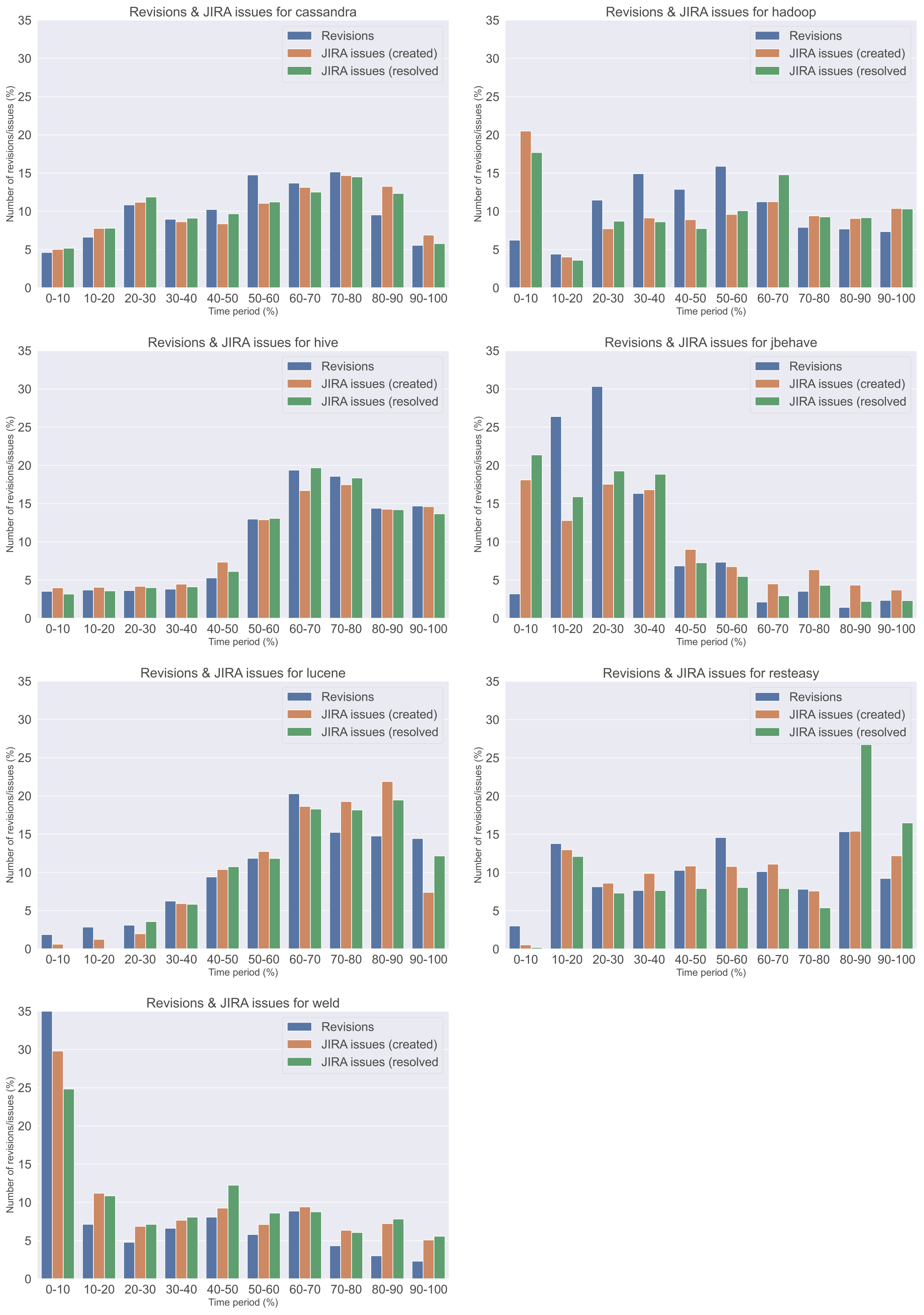


Figure H2: Data distribution visualisation of all SEOS33 software projects. The horizontal axis represent the projects time period as percentage in steps of 10%. The relative number of revisions and JIRA issues are displayed on the vertical axis. Note that, for project *Weld*, the data expands beyond the graphs vertical axis. This is done intentionally for readability purposes. The values outside the range are {Revisions: 49.0%}.

I Frequency on commit size histograms

I1 Frequency on commit sizes of MDD projects

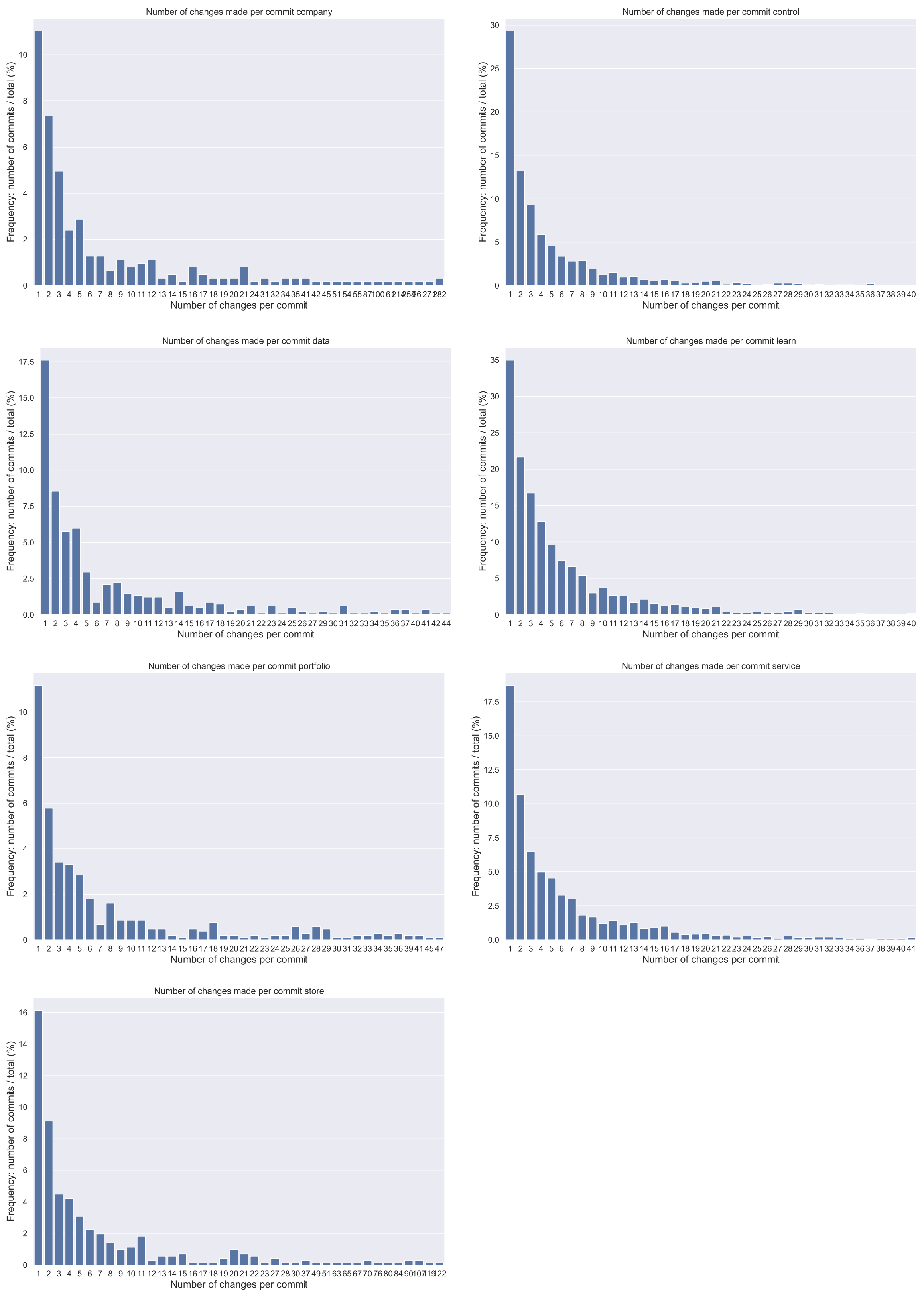


Figure I1: Histograms displaying the number of changes that are made per commit on MDD projects. The horizontal axis show the number of changes, with the frequency of that occurrence on the vertical axis.

I2 Frequency on commit sizes of non-MDD projects

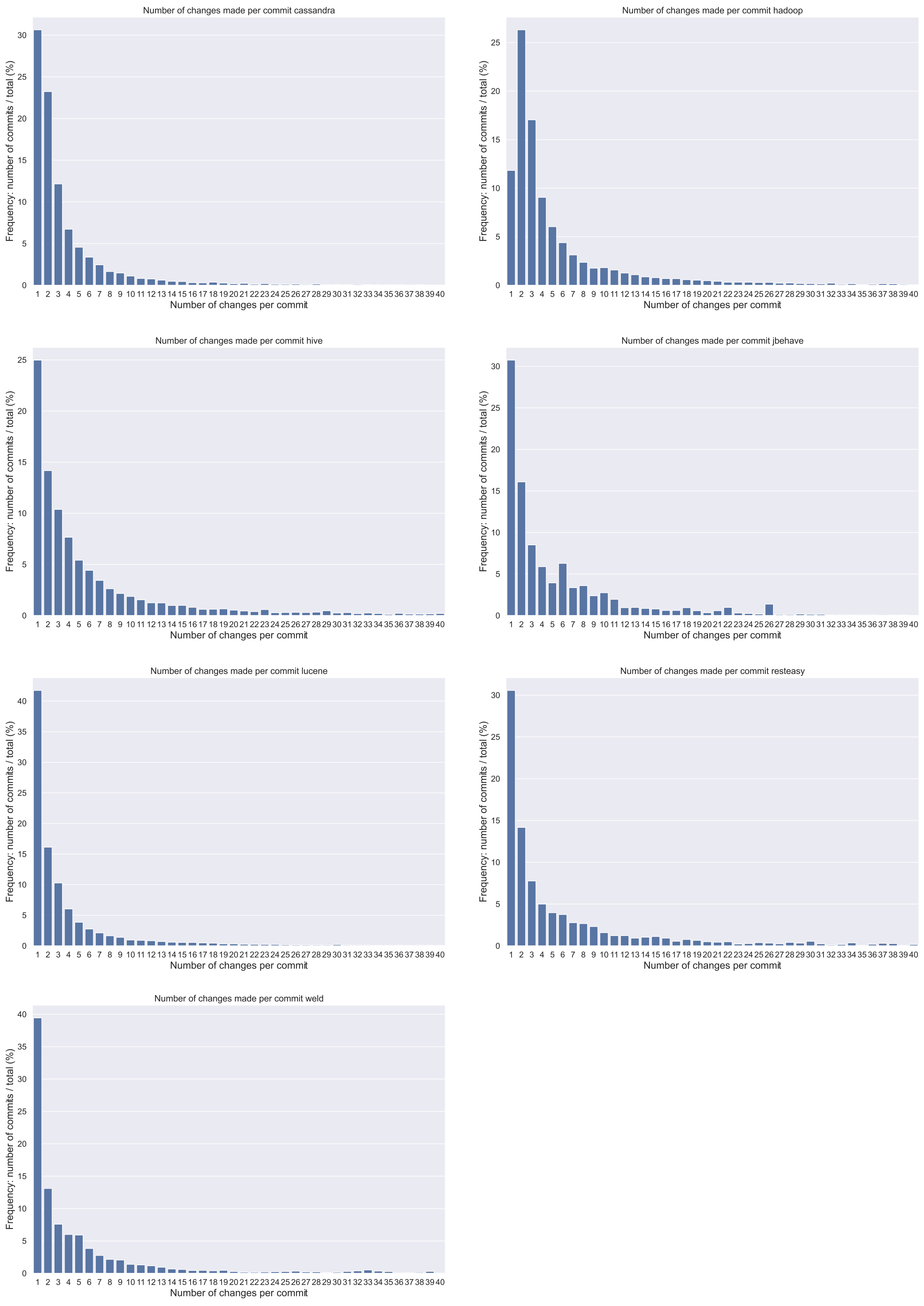


Figure I2: Histograms displaying the number of changes that are made per commit on non-MDD projects. The horizontal axis show the number of changes, with the frequency of that occurrence on the vertical axis.

J Types of change development

J1 Frequency on commit sizes of MDD projects

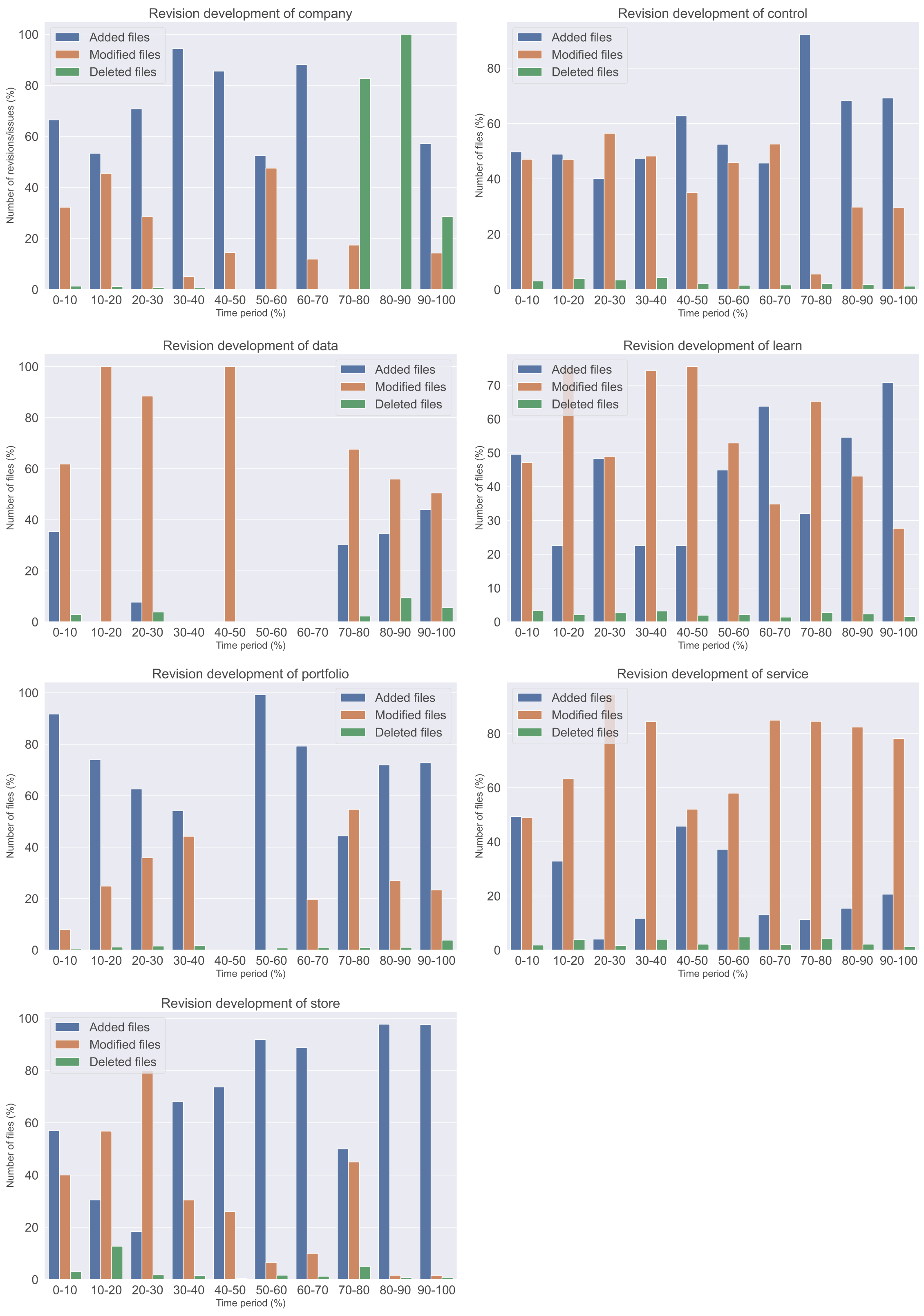


Figure J1: Distribution between files being added, modified, or deleted to all MDD software projects. The horizontal axis represent the projects time period as percentage in steps of 10%. The relative number of added, modified, and removed files are displayed on the vertical axis (within each interval).

J2 Frequency on commit sizes of non-MDD projects

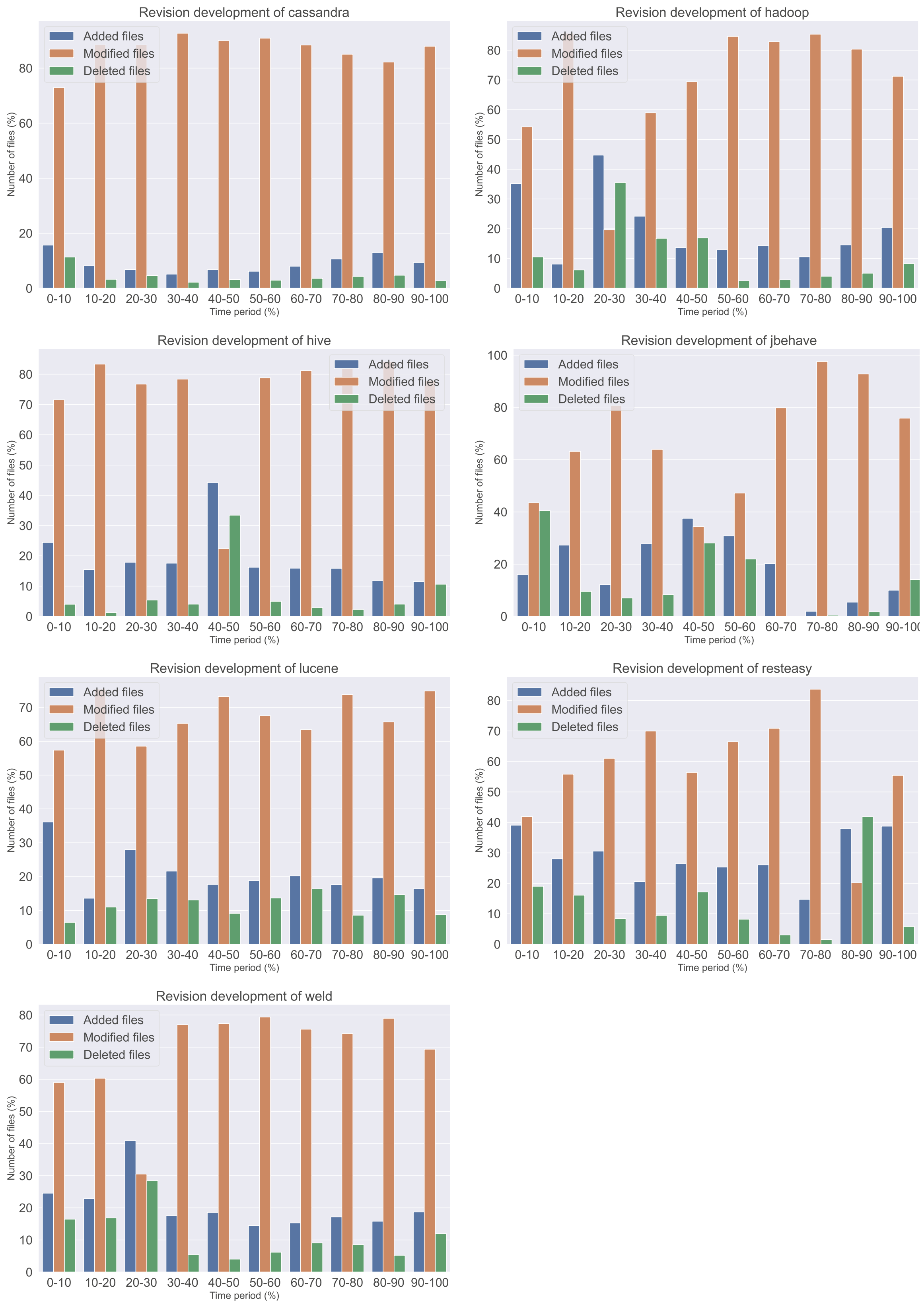


Figure J2: Distribution between files being added, modified, or deleted to all non-MDD software projects. The horizontal axis represent the projects time period as percentage in steps of 10%. The relative number of added, modified, and removed files are displayed on the vertical axis (within each interval).

K Commits for (non-)MDD projects categorised

Table K: Commits categorised by size for both non-MDD projects (top row) and MDD projects (bottom row). Table format adapted from [43]. The values are produced with the use of a Jupyter notebook, which can be found in the online Appendix [33].

Quartiles	Cassandra			Hadoop			Hive			Jbehave			Lucene			Resteasy			Weld		
Q_0 (Min)		1			1			1			1			1			1			1	
Q_1		1			2			1			1			1			1			1	
Q_2 (Median)		2			3			4			3			2			3			2	
Q_3		4			6			9			7			5			10			6	
Q_4 (Max)		662			9400			8980			725			5570			8339			1726	
IQR		2			3			5			4			3			7			4	
Boxplots	**Range**	**Freq.**	**Ratio**	**Range**	**Freq.**	**Ratio**	**Range**	**Freq.**	**Ratio**	**Range**	**Freq.**	**Ratio**	**Range**	**Freq.**	**Ratio**	**Range**	**Freq.**	**Ratio**	**Range**	**Freq.**	**Ratio**
Extra Small	1-1	7228	32%	1-2	10592	38%	1-1	2792	25%	1-1	1009	31%	1-1	12105	42%	1-1	1126	31%	1-1	3189	40%
Small	2-4	9933	44%	3-6	10138	37%	2-9	5617	50%	2-7	1445	45%	2-5	10522	37%	2-10	1619	45%	2-6	2945	37%
Medium	5-7	2449	11%	7-10	2513	9%	10-16	969	9%	8-13	413	13%	6-9	2290	8%	11-20	327	9%	7-12	874	11%
Large	8-10	996	4%	11-15	1545	6%	17-24	458	4%	14-19	144	4%	10-14	1152	4%	21-31	143	4%	13-18	280	4%
Extra Large	11 - 662	1826	8%	16 - 9400	2848	10%	25 - 8980	1328	12%	20-725	223	7%	15-5570	2624	9%	32-8339	418	12%	10-1726	696	9%

Quartiles	Company			Control			Data			Learn			Portfolio			Service			Store		
Q_0 (Min)		1			1			1			1			1			1			1	
Q_1		2			1			1			2			2			1			1	
Q_2 (Median)		3			3			4			4			4			3			3	
Q_3		10			7			12			8			13			8			9	
Q_4 (Max)		401			518			397			612			454			562			530	
IQR		7			4			8			4			9			5			6	
Boxplots	**Range**	**Freq.**	**Ratio**	**Range**	**Freq.**	**Ratio**	**Range**	**Freq.**	**Ratio**	**Range**	**Freq.**	**Ratio**	**Range**	**Freq.**	**Ratio**	**Range**	**Freq.**	**Ratio**	**Range**	**Freq.**	**Ratio**
Extra Small	1 - 2	115	41%	1 - 1	848	33%	1 - 1	144	26%	1 - 2	873	38%	1 - 2	179	25%	1 - 1	548	26%	1 - 1	115	28%
Small	3 - 10	96	35%	2 - 7	1133	44%	2 - 12	275	50%	3 - 8	902	39%	3 - 13	181	51%	2 - 8	1019	49%	2 - 9	196	48%
Medium	11 - 20	33	12%	8 - 13	277	11%	13 - 24	56	10%	9 - 14	243	10%	14 - 26	38	8%	9 - 15	244	12%	10 - 18	39	9%
Large	21 - 31	8	3%	14 - 19	86	3%	25 - 36	23	4%	15 - 20	109	5%	27 - 40	30	6%	16 - 23	106	5%	19 - 27	23	6%
Extra Large	32 - 401	26	9%	20 - 518	243	9%	37 - 397	51	9%	21 - 612	189	8%	41 - 454	47	10%	24 - 562	168	8%	28 - 530	37	9%

L Top 50 most frequent terms in commits

Table L: Top 50 most frequent terms present in commit messages for both MDD and non-MDD projects, all seven projects are covered in each respective column. Table format adapted from [43]. The values are produced with the use of a Jupyter notebook, which can be found in the online Appendix [33].

	MDD		**Non-MDD**	
	Term	**Average rank**	**Term**	**Average rank**
1	add	42.87%	fix	14.02%
2	chang	32.45%	add	11.92%
3	page	22.73%	merg	11.91%
4	updat	21.44%	test	9.18%
5	merg	19.19%	branch	8.74%
6	remov	15.18%	use	6.45%
7	microflow	12.87%	trunk	5.72%
8	fix	12.12%	remov	5.28%
9	creat	9.65%	updat	4.24%
10	modul	9.40%	chang	3.86%
11	new	7.75%	file	3.64%
12	button	7.64%	fail	3.39%
13	user	7.12%	support	3.12%
14	use	6.71%	make	2.74%
15	rev	6.66%	log	2.53%
16	delet	6.53%	move	2.49%
17	review	6.21%	method	2.07%
18	entiti	6.00%	commit	1.99%
19	overview	5.85%	new	1.94%
20	access	5.60%	improv	1.93%
21	test	5.55%	version	1.86%
22	check	5.55%	code	1.79%
23	attribut	5.53%	javadoc	1.78%
24	branch	5.51%	error	1.75%
25	name	5.44%	api	1.74%
26	compani	5.39%	class	1.69%
27	develop	5.21%	configur	1.69%
28	small	5.21%	revert	1.68%
29	logic	5.19%	allow	1.67%
30	project	4.94%	name	1.65%
31	action	4.78%	set	1.63%
32	add	4.69%	miss	1.60%
33	order	4.69%	default	1.57%
34	admin	4.69%	check	1.49%
35	class	4.67%	bug	1.47%
36	set	4.65%	queri	1.46%
37	style	4.62%	releas	1.45%
38	valid	4.44%	creat	1.44%
39	edit	4.37%	work	1.44%
40	member	4.26%	except	1.41%
41	mf	4.06%	conflict	1.41%
42	messag	4.06%	document	1.35%
43	view	3.92%	build	1.33%
44	made	3.87%	handl	1.33%
45	text	3.69%	failur	1.32%
46	imag	3.69%	implement	1.31%
47	object	3.67%	messag	1.29%
48	show	3.65%	run	1.27%
49	modifi	3.47%	upgrad	1.26%
50	associ	3.44%	field	1.25%

M Frequent term sets from commit messages

M1 MDD frequent term sets

Table M1: Aggregated frequent term sets and their support from the seven MDD software projects. Each column presents the most often coexisting terms in a commit message for that respective commit size. Table format adapted from [43]. The values are produced with the use of a Jupyter notebook, which can be found in the online Appendix [33].

Extra Small		Small		Medium		Large		Extra Large	
Terms	**Support**	**Terms**	**Support**	**Terms**	**Support**	**Terms**	**Support**	**Terms**	**Support**
add, page	2.63%	add, microflow	3.75%	chang, review	4.33%	add, modul	3.46%	merg, rev, branch	5.00%
chang, page	2.49%	page, updat	3.65%	merg, develop	3.07%	page, updat	2.77%	add, creat	2.12%
add, button	2.39%	chang, review	2.95%	add, creat	3.07%	add, new	2.77%	creat, modul	1.92%
merg, rev	2.34%	add, remov	2.91%	add, user	3.07%	add, attribut	2.77%	merg, branch, develop	1.92%
add, chang	2.05%	add, creat	2.60%	add, attribut	2.65%	updat, microflow	2.42%	chang, review	1.54%
add, check	1.71%	merg, develop	2.46%	add, use	2.51%	microflow, user	2.42%	add, chang	1.54%
page, button	1.66%	add, updat	2.39%	microflow, creat	2.37%	page, addd	2.08%	modul, access	1.35%
updat, small	1.56%	add, button	2.31%	updat, microflow	2.37%	add, page, overview	2.08%	chang, merg	1.35%
add, class	1.56%	add, new	2.28%	add, check	2.37%	page, modul	2.08%	modul, entiti	1.35%
add, remov	1.41%	chang, microflow	2.28%	remov, microflow	2.23%	add, delet	2.08%	chang, branch	1.35%
add, user	1.36%	add, check	2.21%	page, creat	2.09%	page, button	2.08%	add, updat, modul	1.35%
chang, small	1.32%	page, remov	2.21%	merg, rev, branch	2.09%	add, set	2.08%	creat, entiti	1.35%
add, new	1.32%	add, attribut	2.17%	updat, remov	2.09%	add, action	2.08%	add, entiti	1.15%
add, fix	1.27%	remov, microflow	2.07%	page, overview	1.96%	creat, modul	2.08%	modul, set	1.15%
updat, test	1.27%	add, valid	2.03%	add, associ	1.96%	page, edit	1.73%	add, overview	1.15%
add, compani	1.27%	add, order	2.00%	add, valid	1.96%	page, new	1.73%	page, entiti	1.15%
add, microflow	1.27%	chang, remov	1.93%	updat, use	1.82%	entiti, access	1.73%	chang, updat	1.15%
page, member	1.22%	page, microflow	1.82%	microflow, new	1.82%	page, set	1.73%	page, microflow	1.15%
page, fix	1.22%	add, entiti	1.79%	page, member	1.82%	updat, modul	1.73%	add, logic	1.15%
page, remov	1.22%	add, logic	1.79%	add, access	1.82%	add, creat, entiti	1.73%	creat, object	1.15%
add, modul	1.17%	updat, microflow	1.75%	chang, microflow	1.82%	remov, entiti	1.73%	modul, user	1.15%
merg, branch	1.17%	add, delet	1.68%	add, overview	1.68%	chang, page, microflow	1.73%	microflow, creat	0.96%
fix, small	1.17%	microflow, creat	1.68%	new, entiti	1.68%	microflow, associ	1.73%	creat, logic	0.96%
page, project	1.12%	chang, updat	1.68%	updat, modul	1.68%	chang, develop	1.73%	modul, addd	0.96%

M2 Non-MDD frequent term sets

Table M2: Aggregated frequent term sets and their support from the seven non-MDD software projects. Each column presents the most often coexisting terms in a commit message for that respective commit size. Table format adapted from [43]. The values are produced with the use of a Jupyter notebook, which can be found in the online Appendix [33].

Extra Small		Small		Medium		Large		Extra Large	
Terms	**Support**	**Terms**	**Support**	**Terms**	**Support**	**Terms**	**Support**	**Terms**	**Support**
merg, branch, trunk	3.38%	merg, branch, trunk	5.03%	merg, branch, trunk	5.01%	merg, branch, trunk	4.47%	merg, branch, trunk	5.54%
add, test	1.34%	merg, branch, conflict	1.61%	add, support	1.63%	add, support	1.63%	add, support	1.75%
fix, javadoc	1.18%	fix, use	0.72%	merg, branch, conflict	1.53%	merg, branch, conflict	1.48%	merg, branch, conflict	1.37%
test, use	0.73%	add, support	0.71%	commit, revert	0.73%	merg, trunk, conflict	0.85%	test, use	0.97%
fix, build	0.58%	test, use	0.59%	branch, trunk, conflict	0.65%	commit, revert	0.83%	merg, trunk, conflict	0.96%
fix, chang	0.57%	test, fail	0.51%	merg, trunk, conflict	0.65%	branch, trunk, conflict	0.83%	fix, test	0.93%
fix, use	0.57%	merg, trunk, conflict	0.46%	test, use	0.63%	add, api	0.68%	commit, revert	0.85%
fix, error	0.51%	commit, revert	0.46%	add, new	0.62%	add, new	0.66%	branch, trunk, conflict	0.80%
fix, merg	0.51%	branch, trunk, conflict	0.46%	add, method	0.56%	use, new	0.64%	add, test	0.77%
fix, test, bug	0.50%	fix, error	0.46%	fix, use	0.56%	fix, bug	0.64%	move, class	0.64%
remov, code	0.49%	fix, file	0.44%	add, use	0.52%	use, api	0.55%	use, chang	0.62%
error, messag	0.49%	add, miss	0.43%	remov, method	0.44%	test, use	0.55%	fix, merg	0.60%
add, chang	0.49%	updat, version	0.42%	add, api	0.44%	fix, use	0.55%	fix, bug	0.60%
test, run	0.48%	fix, merg	0.40%	add, file	0.43%	test, improv	0.55%	test, move	0.59%
chang, move	0.48%	add, method	0.40%	test, fail	0.42%	use, chang	0.53%	use, api	0.58%
updat, chang	0.47%	error, messag	0.40%	fix, file	0.41%	test, fail	0.53%	add, use	0.55%
commit, revert	0.46%	fix, fail	0.39%	use, new	0.41%	add, use	0.49%	use, new	0.54%
merg, branch, conflict	0.45%	fix, javadoc	0.38%	use, make	0.37%	test, file	0.49%	test, remov	0.53%
test, remov	0.43%	add, file	0.38%	use, api	0.36%	fix, error	0.47%	add, api	0.52%
fix, file	0.42%	fix, handl	0.37%	add, configur	0.36%	add, method	0.47%	fix, use	0.52%
fix, commit	0.41%	test, run	0.34%	use, chang	0.36%	add, file	0.47%	add, new	0.51%
log, messag	0.40%	fix, queri	0.33%	use, file	0.35%	chang, new	0.45%	remov, class	0.50%
updat, releas	0.40%	fix, work	0.33%	use, updat	0.35%	use, remov	0.42%	use, remov	0.50%
fix, test, failur	0.40%	fix, chang	0.30%	fix, merg	0.34%	use, updat	0.40%	remov, method	0.48%

N Experimental method

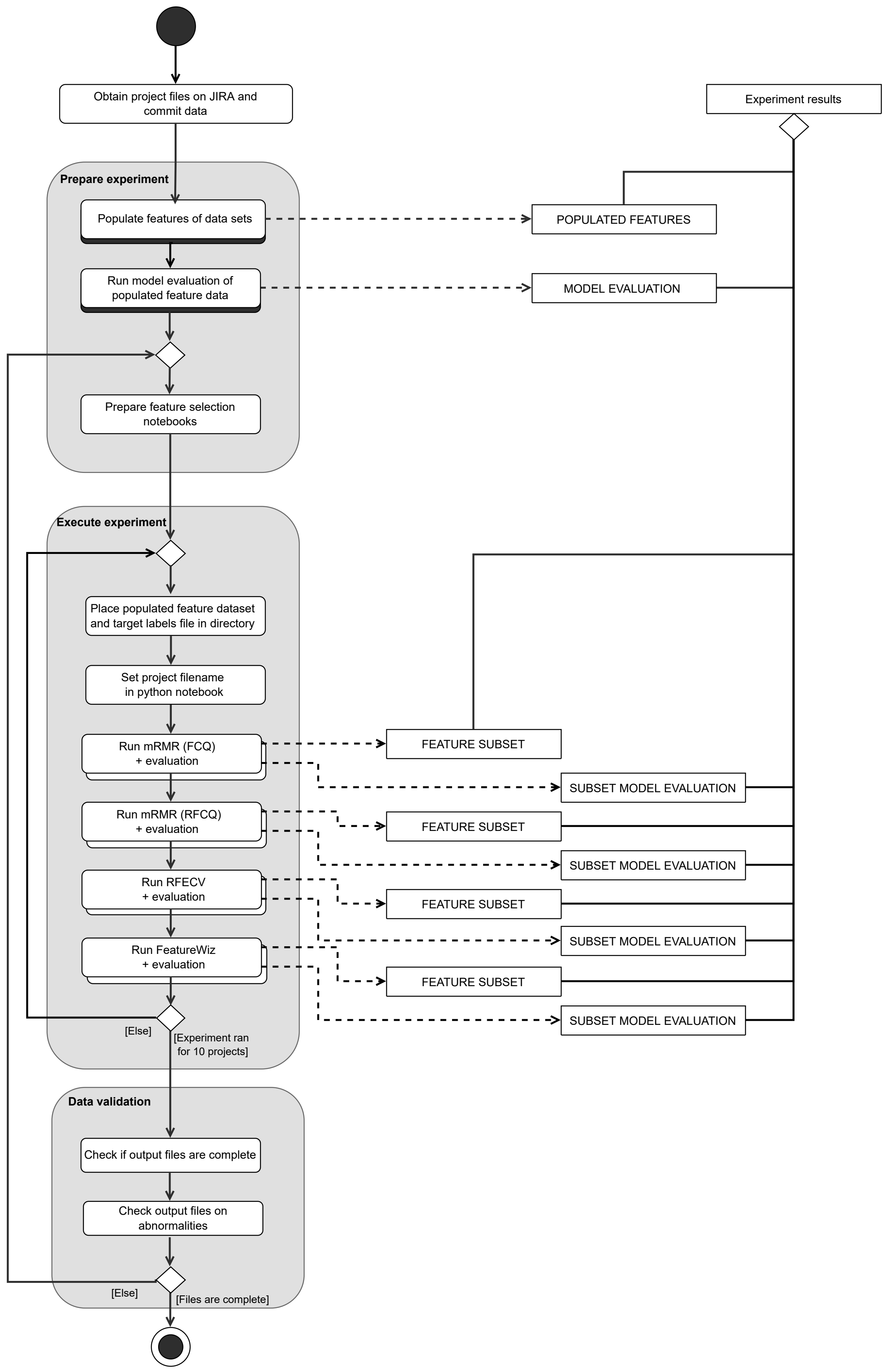

Obtain project files on JIRA and commit data
Prepare experiment
Populate features of data sets
Run model evaluation of populated feature data
Prepare feature selection notebooks
Execute experiment
Place populated feature dataset and target labels file in directory
Set project filename in python notebook
Run mRMR (FCQ) + evaluation
Run mRMR (RFCQ) + evaluation
Run RFECV + evaluation
Run FeatureWiz + evaluation
[Else]
[Experiment ran for 10 projects]
Data validation
Check if output files are complete
Check output files on abnormalities
[Else]
[Files are complete]
Experiment results
POPULATED FEATURES
MODEL EVALUATION
FEATURE SUBSET
SUBSET MODEL EVALUATION
FEATURE SUBSET
SUBSET MODEL EVALUATION
FEATURE SUBSET
SUBSET MODEL EVALUATION
FEATURE SUBSET
SUBSET MODEL EVALUATION

Figure N: Process-Deliverable Diagram of experiment operation

O Baseline experiment

O1 Baseline performance results

Table O1: Baseline results for all ten software projects for trace recommendation (LightGBM algorithm and over/under rebalancing applied) and trace maintenance scenario (XGBoost algorithm and no rebalancing strategy applied). All 131 features are used to generate the baseline results.

Project K = 131	Trace recommendation						Trace maintenance					
	Precision		Recall		F_2		Precision		Recall		$F_{0.5}$	
	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ
Company	54.78	4.50	66.67	6.64	63.69	3.88	74.35	7.35	48.89	2.43	67.20	4.87
Control_1	37.10	2.31	55.48	4.41	50.41	2.23	79.53	6.27	41.29	4.06	66.91	4.42
Control_2	44.89	6.18	59.75	11.27	55.81	6.45	70.76	7.45	48.25	7.64	64.43	6.58
Control_3	51.77	5.61	71.54	5.54	66.39	5.48	75.99	7.02	56.00	6.03	70.83	6.30
Data	62.68	3.63	76.18	5.03	72.94	3.20	84.18	3.74	61.73	5.34	78.39	3.58
Learn_1	46.18	3.42	60.67	3.31	57.01	2.99	80.65	3.95	44.45	3.62	69.19	2.28
Learn_2	46.68	8.83	48.97	8.57	48.27	8.18	59.14	11.36	31.03	11.26	49.20	11.42
Portfolio	40.69	2.41	66.17	5.60	58.74	2.52	74.15	6.94	46.23	3.12	66.07	5.05
Service	54.79	4.72	73.57	5.44	68.69	4.18	78.77	6.43	46.67	5.49	69.02	4.82
Store	58.22	7.95	70.59	9.20	67.43	7.47	82.10	10.35	57.65	12.34	74.62	6.27
Macro-Avg	*49.78*	*8.04*	*64.96*	*8.63*	*60.94*	*8.18*	*75.96*	*7.18*	*48.22*	*8.80*	*67.59*	*7.69*

O2 Baseline feature importance results

Table O2: Baseline feature importance results for trace recommendation (LightGBM algorithm and over/under rebalancing applied) and trace maintenance scenario (XGBoost algorithm and no rebalancing strategy applied). All 131 features are used to generate the baseline results. The feature importance is presented as percentage and categorised per feature family. The table presents the number of selected features for each feature family, aggregated the sum, average, standard deviation, and the maximum feature importance.

Feature Family	$\overline{N}$	Trace Recommendation				Trace Maintenance			
		$\overline{\sum}$	$\overline{\overline{X}}$	$\overline{\sigma}$	Max	$\overline{\sum}$	$\overline{\overline{X}}$	$\overline{\sigma}$	Max
Process related	4	64.55	16.14	1.61	68.73	6.58	1.65	0.56	4.36
Documen Statistics	7	1.12	0.16	0.07	1.41	6.80	0.97	0.34	3.32
Information Retrieval	18	6.89	0.38	0.25	5.71	21.10	1.17	0.25	8.14
QQ-Specificity	72	20.32	0.28	0.09	7.82	49.73	0.69	0.06	7.49
QQ-Similarity	18	4.20	0.23	0.11	2.39	7.67	0.43	0.16	2.96
QQ-Term Relatedness	12	2.93	0.24	0.11	2.51	8.12	0.68	0.15	7.28

P Experiment aggregated results

P1 Performance results by mRMR (FCQ)

Table P1: Mean and standard deviation for precision, recall, and F$_2$-measure (trace recommendation) and F$_{0.5}$- (trace maintenance scenario) across all ten datasets. For the F-measure column's, the maximum value is included as well. The subsets are created by using the **mRMR** algorithm with **FCQ**, with sizes K=40, K=50, K=60. The colored cells highlight which value of K leads to the highest metric for each of the projects in each scenario. This comparison is shown in green for the trace recommendation scenario, in orange for the trace maintenance scenario.

Project K = 40	Trace recommendation							Trace maintenance						
	Precision		Recall		F_2			Precision		Recall		$F_{0.5}$		
	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	Max	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	Max
Company	43.18	3.22	73.06	5.45	64.06	3.87	67.57	75.01	4.95	53.47	5.00	69.28	3.93	76.92
Control_1	31.13	1.41	66.29	2.71	54.07	2.18	57.41	76.80	4.33	44.76	1.91	67.15	3.23	70.39
Control_2	42.41	3.39	66.25	5.43	59.39	3.67	65.32	70.14	8.49	43.75	6.04	62.41	7.03	72.92
Control_3	41.42	3.13	81.69	5.91	68.28	4.14	73.64	69.96	5.41	57.23	8.33	66.79	5.32	73.26
Data	53.14	4.85	79.64	3.21	72.33	3.27	77.97	82.52	4.21	61.91	4.74	77.27	3.32	82.65
Learn_1	34.80	1.38	69.02	3.77	57.65	2.59	61.62	83.48	2.87	47.80	2.91	72.57	2.30	76.17
Learn_2	40.61	6.42	57.59	11.84	52.65	8.40	66.67	51.80	9.23	35.52	8.14	47.23	8.36	60.00
Portfolio	27.14	1.20	76.23	2.89	55.96	1.97	59.35	79.34	2.58	46.82	2.30	69.61	1.65	71.96
Service	42.61	2.51	74.52	3.81	64.75	2.76	69.33	80.42	5.12	46.90	7.19	70.17	5.64	78.31
Store	53.83	7.00	71.76	14.62	66.86	11.78	81.52	80.59	9.37	55.88	6.93	73.74	7.25	90.16
Macro-Avg	*41.03*	*3.45*	*71.61*	*5.96*	*61.60*	*4.46*	*68.04*	*75.00*	*5.65*	*49.40*	*5.35*	*67.62*	*4.80*	*75.27*

Project K = 50	Trace recommendation							Trace maintenance						
	Precision		Recall		F_2			Precision		Recall		$F_{0.5}$		
	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	Max	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	Max
Company	42.45	3.38	70.14	4.21	61.94	3.11	66.34	75.97	6.47	53.89	4.48	70.04	4.57	77.38
Control_1	31.46	1.50	62.58	3.81	52.21	2.60	57.03	76.88	5.94	42.98	4.75	66.21	4.58	74.22
Control_2	40.92	4.06	64.50	7.05	57.60	4.63	67.80	69.72	7.22	50.75	6.46	64.73	6.24	75.00
Control_3	45.46	3.04	81.23	4.63	70.08	3.01	75.06	71.86	3.98	55.23	1.84	67.73	2.85	72.29
Data	54.32	3.01	80.27	3.69	73.20	2.68	77.11	84.07	3.90	62.73	5.83	78.62	3.84	82.94
Learn_1	35.68	2.39	66.77	4.94	56.85	3.98	63.80	80.77	4.71	47.01	4.85	70.45	3.92	79.14
Learn_2	45.57	4.01	54.48	11.92	52.03	9.26	64.81	56.35	7.95	39.31	9.64	51.28	7.01	61.80
Portfolio	26.83	1.35	78.57	3.11	56.66	1.91	60.54	74.99	5.19	48.25	2.86	67.40	3.45	72.87
Service	46.47	2.55	68.10	5.46	62.19	3.83	66.95	81.46	6.27	43.57	5.56	69.05	4.66	78.63
Store	49.34	8.05	67.06	9.28	62.19	7.28	74.47	84.01	8.68	65.88	9.92	79.33	7.91	89.04
Macro-Avg	*41.85*	*3.33*	*69.37*	*5.81*	*60.50*	*4.23*	*67.39*	*75.61*	*6.03*	*50.96*	*5.62*	*68.48*	*4.91*	*76.33*

Project K = 60	Trace recommendation							Trace maintenance						
	Precision		Recall		F_2			Precision		Recall		$F_{0.5}$		
	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	Max	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	Max
Company	44.57	3.16	65.83	5.94	59.99	4.49	68.18	72.14	5.74	53.89	5.66	67.24	3.41	71.69
Control_1	32.05	1.51	63.06	4.57	52.79	2.94	57.12	77.18	4.96	45.81	4.31	67.78	4.21	73.86
Control_2	48.79	7.16	66.50	6.37	61.89	6.29	74.07	71.06	9.01	50.25	11.63	65.02	8.56	77.21
Control_3	45.66	3.50	75.54	4.56	66.70	3.46	72.60	72.02	7.40	59.85	6.04	68.99	5.74	75.51
Data	54.10	2.97	81.00	3.39	73.62	2.55	76.53	85.60	2.57	63.45	4.98	79.89	1.61	82.09
Learn_1	36.51	2.22	66.59	3.68	57.16	3.13	61.04	81.20	3.90	44.39	1.49	69.62	2.64	73.08
Learn_2	39.63	7.32	49.31	10.79	46.76	9.12	59.75	58.67	11.75	36.90	7.97	52.29	10.05	64.71
Portfolio	28.00	2.24	77.08	3.88	57.03	3.34	63.48	76.11	4.22	49.03	3.68	68.48	3.68	74.55
Service	50.57	4.38	75.83	3.55	68.89	3.65	75.39	78.36	3.99	53.57	5.20	71.54	3.02	77.05
Store	60.96	8.84	77.65	7.23	73.45	6.99	84.27	73.98	8.96	54.12	14.36	68.28	9.66	82.19
Macro-Avg	*44.08*	*4.33*	*69.84*	*5.40*	*61.83*	*4.59*	*69.24*	*74.63*	*6.25*	*51.12*	*6.53*	*67.91*	*5.26*	*75.19*

P2 Performance results by mRMR (RFCQ)

Table P2: Mean and standard deviation for precision, recall, and F_2-measure (trace recommendation) and $F_{0.5}$- (trace maintenance scenario) across all ten datasets. For the F-measure column's, the maximum value is included as well. The subsets are created by using the **mRMR** algorithm with **RFCQ** scheme, with sizes K=40, K=50, K=60. The colored cells highlight which value of K leads to the highest metric for each of the projects in each scenario. This comparison is shown in green for the trace recommendation scenario, in orange for the trace maintenance scenario.

Project	Trace recommendation							Trace maintenance						
K = 40	Precision		Recall		F_2			Precision		Recall		$F_{0.5}$		
	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	Max	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	Max
Company	50.92	3.22	66.53	6.53	62.54	4.85	68.48	75.08	4.15	52.36	7.66	68.87	4.53	74.22
Control_1	30.09	2.01	64.03	2.35	52.17	1.53	54.96	76.91	5.61	39.19	3.86	64.40	4.56	69.15
Control_2	42.20	5.62	64.75	6.92	58.34	5.99	68.58	74.13	9.84	46.25	8.35	65.56	7.86	79.55
Control_3	47.92	3.97	77.85	6.49	69.12	5.20	77.66	70.09	4.78	54.77	4.36	66.31	4.19	71.43
Data	40.10	2.62	73.45	3.85	62.94	3.22	67.13	78.80	5.50	46.55	3.31	69.10	3.93	76.59
Learn_1	34.86	2.53	62.62	4.45	53.89	2.82	57.90	80.55	3.60	40.98	4.40	67.39	3.87	73.82
Learn_2	43.21	10.41	54.14	15.08	51.46	13.74	74.68	62.02	10.06	37.24	9.99	54.37	9.83	64.36
Portfolio	37.52	1.51	68.12	3.75	58.54	2.73	63.33	75.10	4.43	46.56	3.37	66.81	3.47	71.84
Service	44.61	2.98	74.88	5.00	65.85	3.73	72.33	80.59	5.85	48.69	5.22	71.09	4.64	75.76
Store	57.68	6.11	75.29	11.02	70.70	8.63	83.33	76.49	14.45	52.35	13.14	69.23	12.46	84.91
Macro-Avg	*42.91*	*4.10*	*68.17*	*6.55*	*60.56*	*5.24*	*68.84*	*74.98*	*6.83*	*46.49*	*6.37*	*66.31*	*5.93*	*74.16*

Project	Trace recommendation							Trace maintenance						
K = 50	Precision		Recall		F_2			Precision		Recall		$F_{0.5}$		
	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	Max	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	Max
Company	50.65	3.76	70.69	5.34	65.46	4.44	75.00	77.15	4.70	53.19	3.82	70.61	2.84	74.60
Control_1	35.36	2.61	59.68	3.90	52.40	3.02	57.83	77.51	5.16	39.68	3.27	64.96	3.42	69.74
Control_2	45.30	4.92	66.00	5.30	60.24	3.67	64.44	70.46	10.56	42.00	5.11	61.86	8.18	72.58
Control_3	52.05	2.98	74.46	3.18	68.54	3.03	72.44	73.37	5.56	58.31	5.25	69.61	4.31	74.01
Data	58.41	2.74	79.27	3.71	73.93	2.68	79.12	82.16	2.67	61.73	5.53	76.99	3.13	84.04
Learn_1	35.66	2.22	65.98	4.47	56.34	3.35	63.28	79.48	3.44	43.72	3.81	68.17	2.53	71.81
Learn_2	43.04	8.28	47.59	11.92	46.50	10.97	59.21	67.76	10.22	36.55	8.93	57.30	9.71	72.16
Portfolio	36.90	2.79	67.21	5.10	57.67	3.94	65.02	72.43	3.67	43.31	3.09	63.78	2.99	68.83
Service	42.79	3.93	72.50	5.76	63.57	4.57	67.85	78.59	6.18	44.40	2.97	67.99	4.22	73.08
Store	61.81	10.68	74.71	7.87	71.25	6.78	86.21	91.75	10.15	58.82	8.77	82.04	8.27	92.31
Macro-Avg	*46.20*	*4.49*	*67.81*	*5.65*	*61.59*	*4.65*	*69.04*	*77.07*	*6.23*	*48.17*	*5.06*	*68.33*	*4.96*	*75.31*

Project	Trace recommendation							Trace maintenance						
K = 60	Precision		Recall		F_2			Precision		Recall		$F_{0.5}$		
	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	Max	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	Max
Company	49.41	3.99	66.67	7.99	62.20	6.45	70.33	72.45	3.66	51.39	6.07	66.86	4.22	75.00
Control_1	33.03	2.46	56.13	5.95	49.23	4.76	57.34	79.46	6.19	40.65	3.34	66.67	5.00	72.16
Control_2	43.06	6.33	60.25	10.30	55.46	8.16	67.13	73.77	5.22	49.25	8.34	66.71	4.99	75.00
Control_3	50.47	4.23	74.92	2.18	68.21	2.29	70.82	69.98	6.98	58.00	5.89	67.05	5.76	75.09
Data	58.69	2.48	78.55	4.72	73.49	3.28	76.99	84.56	3.92	61.45	2.78	78.59	2.99	84.10
Learn_1	36.78	2.14	62.74	2.75	54.94	2.11	57.19	76.78	3.76	41.95	4.33	65.74	3.74	71.81
Learn_2	41.75	5.60	47.59	11.36	45.95	8.95	64.81	54.80	9.15	30.69	8.67	47.00	9.22	61.90
Portfolio	36.87	3.00	66.17	4.12	57.05	3.51	63.06	73.21	4.23	43.90	2.72	64.50	3.00	68.44
Service	44.27	2.66	73.45	5.81	64.80	4.07	74.69	77.78	6.00	47.02	5.00	68.54	4.23	73.94
Store	62.89	12.85	68.24	18.01	66.47	16.21	94.44	85.75	7.99	61.18	13.92	78.83	8.92	90.91
Macro-Avg	*45.72*	*4.57*	*65.47*	*7.32*	*59.78*	*5.98*	*69.68*	*74.85*	*5.71*	*48.55*	*6.11*	*67.05*	*5.21*	*74.84*

P3 Performance results by RFECV

Table P3: Mean and standard deviation for precision, recall, and F_2-measure (trace recommendation) and $F_{0.5}$-measure (trace maintenance scenario) across all ten datasets using the **RFECV** feature selection algorithm. For the F-measures, the maximum value is presented as well.

Project		Trace recommendation							Trace maintenance						
		Precision		Recall		F_2			Precision		Recall		$F_{0.5}$		
	K	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	Max	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	Max
Company	36	47.57	4.54	66.94	3.75	61.83	3.55	66.75	76.34	6.46	52.50	5.23	69.90	5.69	77.21
Control_1	6	16.77	1.11	75.89	3.08	44.49	2.20	49.46	73.09	3.76	38.31	2.44	61.75	2.30	65.22
Control_2	28	44.31	5.96	67.50	6.45	61.00	6.09	70.14	74.50	8.15	46.00	5.80	65.88	5.08	79.17
Control_3	21	45.55	1.87	75.54	5.00	66.66	3.06	71.80	71.91	7.49	54.31	5.98	67.40	6.30	80.91
Data	11	48.21	2.07	80.36	3.98	70.86	2.75	74.33	82.67	4.26	63.27	4.05	77.83	3.59	82.60
Learn_1	8	14.42	0.68	78.05	3.56	41.45	1.76	44.27	80.58	5.73	39.70	4.25	66.68	4.61	71.57
Learn_2	21	33.79	5.16	50.00	11.05	45.38	8.93	61.73	60.96	10.60	36.21	7.67	53.25	8.75	67.42
Portfolio	3	18.88	0.77	90.06	2.41	51.33	1.51	54.31	71.87	5.52	36.69	3.90	60.24	4.90	68.57
Service	6	24.30	0.75	81.55	3.90	55.39	1.65	57.45	74.49	4.54	46.07	4.83	66.14	3.65	71.88
Store	5	28.01	6.68	75.29	11.70	55.46	8.15	63.06	79.78	11.30	62.35	13.36	74.63	8.55	86.96
Macro-Avg	*15*	*32.18*	*2.96*	*74.12*	*5.49*	*55.38*	*3.97*	*61.33*	*74.62*	*6.78*	*47.54*	*5.75*	*66.37*	*5.34*	*75.15*

P4 Performance results by FeatureWiz

Table P4: Mean and standard deviation for precision. recall. and F_2-measure (trace recommendation) and $F_{0.5}$-measure (trace maintenance scenario) across all ten datasets using the **FeatureWiz** feature selection algorithm. For the F-measures, the maximum value is presented as well.

Project		Trace recommendation							Trace maintenance						
		Precision		Recall		F_2			Precision		Recall		$F_{0.5}$		
	K	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	Max	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	Max
Company	36	50.52	4.96	66.53	4.56	62.39	3.33	67.53	72.37	5.54	50.97	3.41	66.74	4.70	75.76
Control_1	37	32.77	4.42	59.35	7.63	51.01	6.39	59.49	76.47	4.23	42.26	4.12	65.70	3.75	71.93
Control_2	30	44.86	7.15	67.75	7.95	61.32	7.35	75.89	69.37	4.64	50.25	7.68	64.26	4.53	72.92
Control_3	31	54.98	4.03	75.85	4.23	70.45	3.80	74.07	75.99	5.95	58.00	6.03	71.39	5.01	78.31
Data	40	59.85	4.53	76.55	4.55	72.41	3.72	77.78	82.33	3.95	57.91	3.54	75.86	3.15	79.60
Learn_1	32	43.88	2.45	63.84	3.60	58.49	2.99	61.60	79.83	2.54	45.00	1.46	69.11	1.91	71.56
Learn_2	33	45.59	4.76	57.93	7.59	54.81	6.30	63.33	59.37	10.82	37.59	5.96	52.61	7.08	67.42
Portfolio	31	42.38	2.55	63.51	4.39	57.67	3.07	62.21	76.84	4.03	46.88	3.47	68.05	3.29	73.89
Service	40	49.13	3.49	68.45	3.19	63.40	2.65	67.98	80.89	4.51	52.50	4.68	72.83	3.23	80.25
Store	37	53.97	6.11	71.76	11.02	66.87	7.74	79.79	85.06	13.19	55.29	11.83	75.32	8.77	86.96
Macro-Avg	*35*	*47.79*	*4.44*	*67.15*	*5.87*	*61.88*	*4.73*	*68.97*	*75.85*	*5.94*	*49.67*	*5.22*	*68.19*	*4.54*	*75.86*

Q Box and Whisker plots experiment

Q1 Box and Whisker plots for trace recommendation

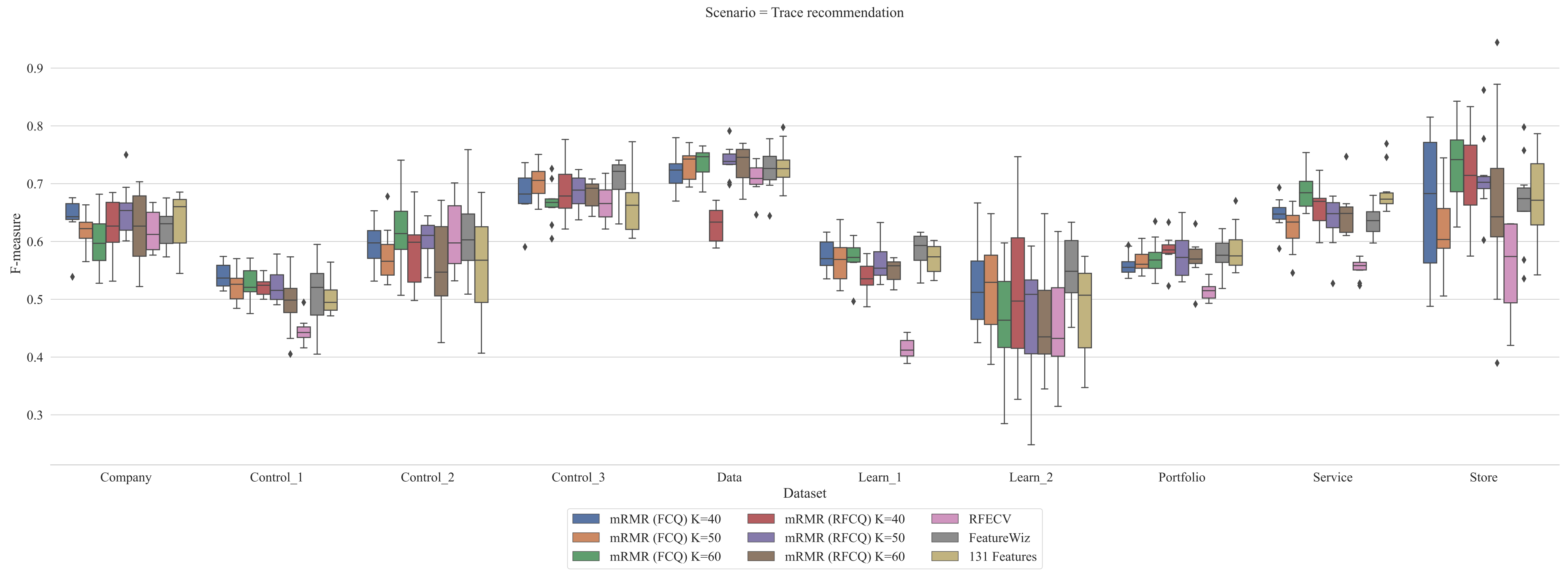


Figure Q1: Box and Whisker plots for the trace recommendation scenario. The X-axis represents the software projects (n=10), with the y-axis displaying the F_2-measure. Each plot is generated from 10 individual classifier evaluation rounds.

Q2 Box and Whisker plots for trace maintenance

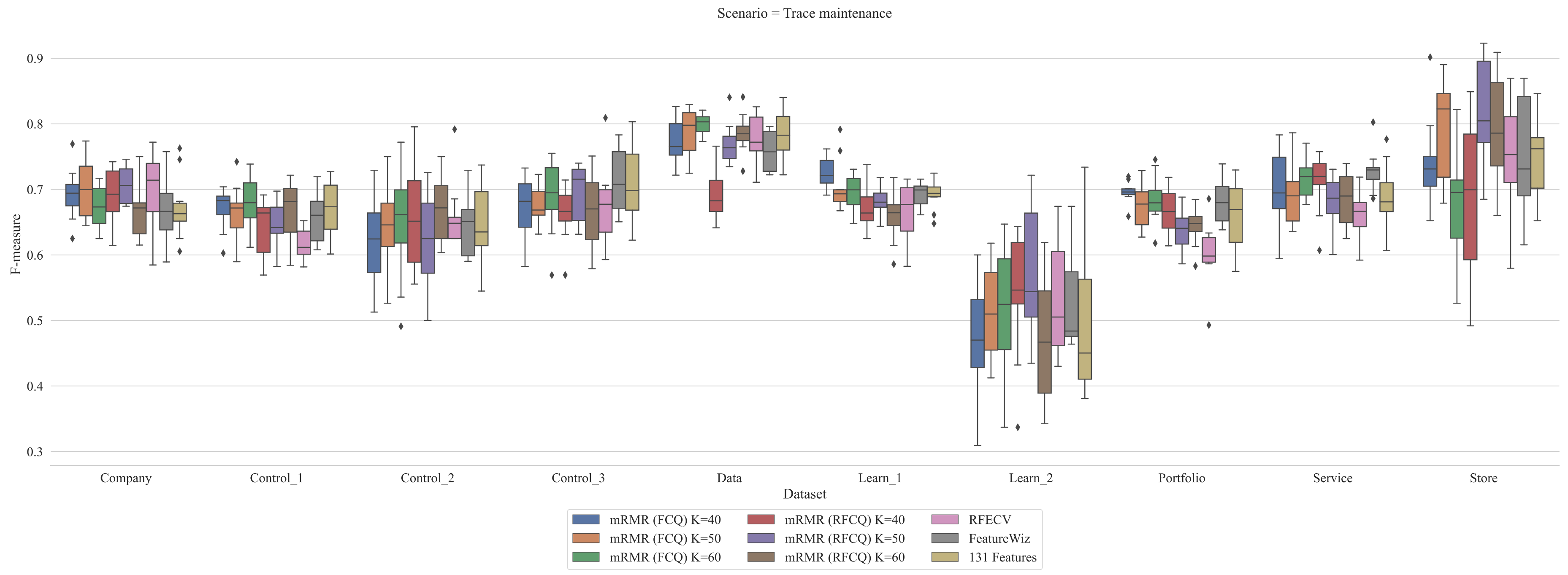


Figure Q2: Box and Whisker plots for trace maintenance scenario. The X-axis represents the software projects (n=10), with the y-axis displaying the $F_{0.5}$-measure. Each plot is generated from 10 individual classifier evaluation rounds.

Q3 Box and Whisker plots by mRMR (FCQ)

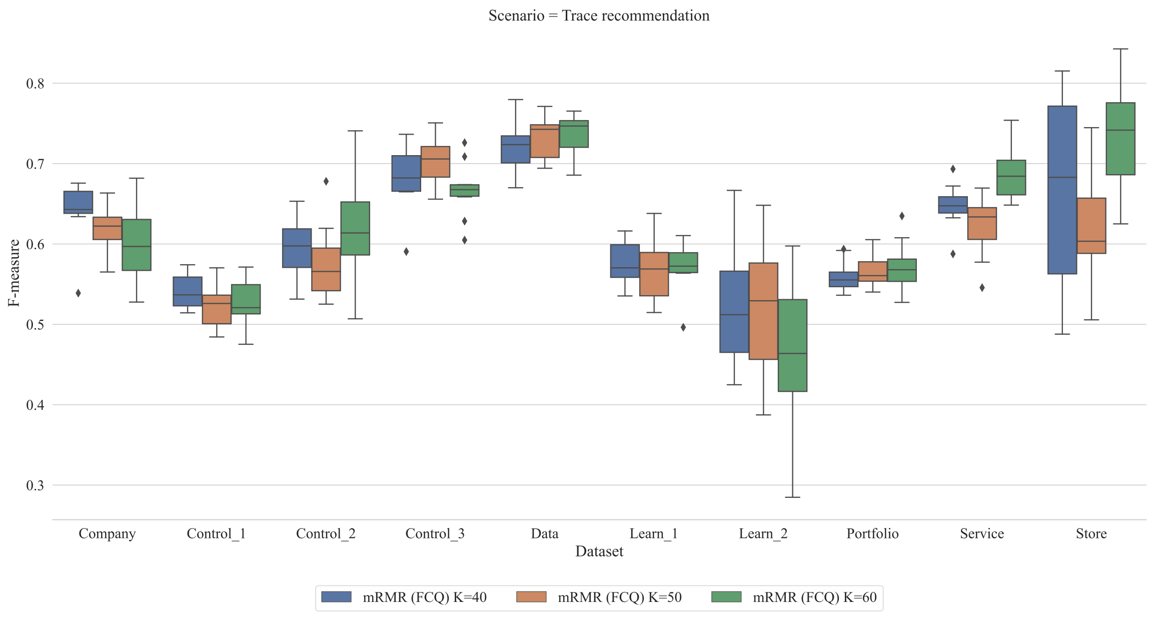


Figure Q3.1: Box and Whisker plots for trace recommendation scenario using the **mRMR** algorithm with **FCQ** scheme. The X-axis represents the software projects (n=10), with the y-axis displaying the F_2-measure. Each plot is generated from 10 individual classifier evaluation rounds.

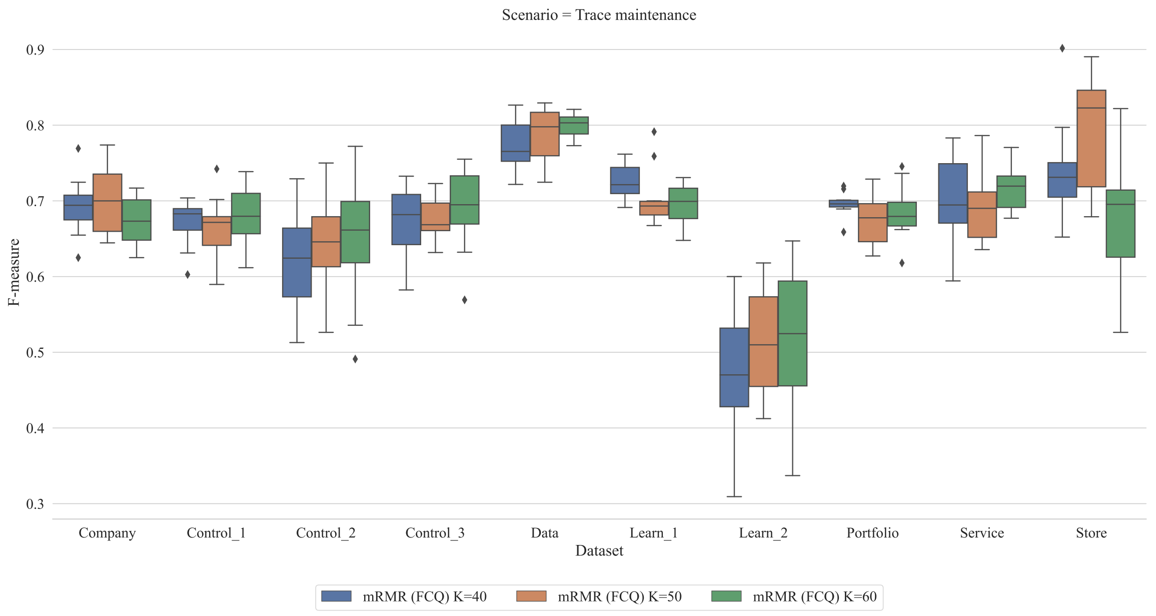


Figure Q3.2: Box and Whisker plots for trace maintenance scenario using the **mRMR** algorithm with **FCQ** scheme. The X-axis represents the software projects (n=10), with the y-axis displaying the $F_{0.5}$-measure. Each plot is generated from 10 individual classifier evaluation rounds.

Q4 Box and Whisker plots by mRMR (RFCQ)

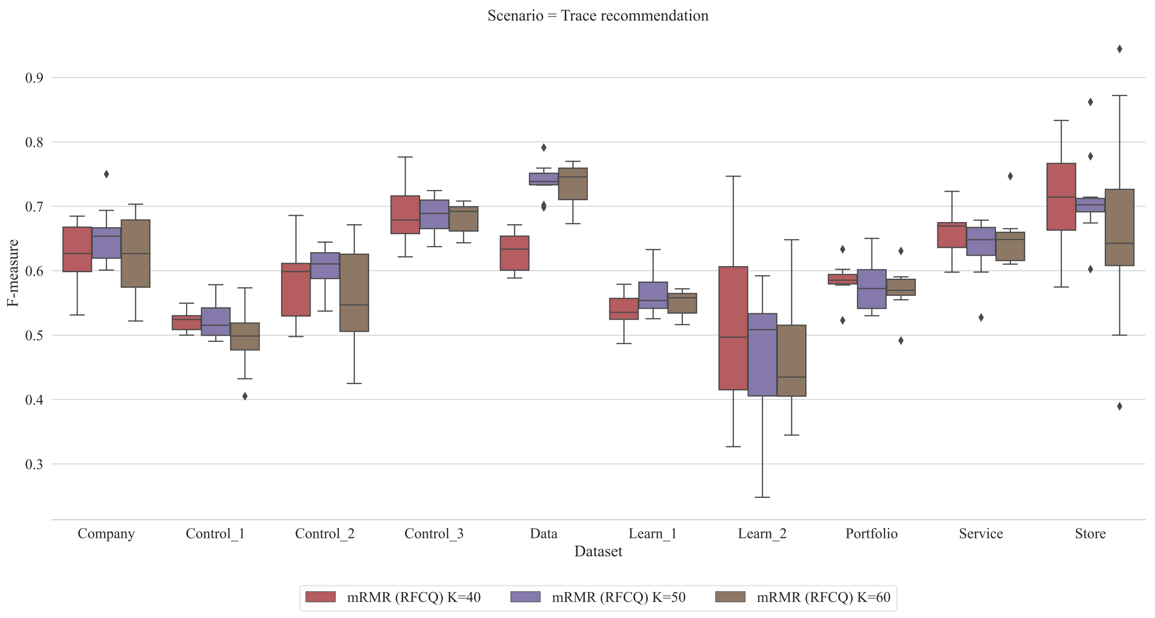


Figure Q4.1: Box and Whisker plots for trace recommendation scenario using the **mRMR** algorithm with **RFCQ** scheme. The X-axis represents the software projects (n=10), with the y-axis displaying the F_2-measure. Each plot is generated from 10 individual classifier evaluation rounds.

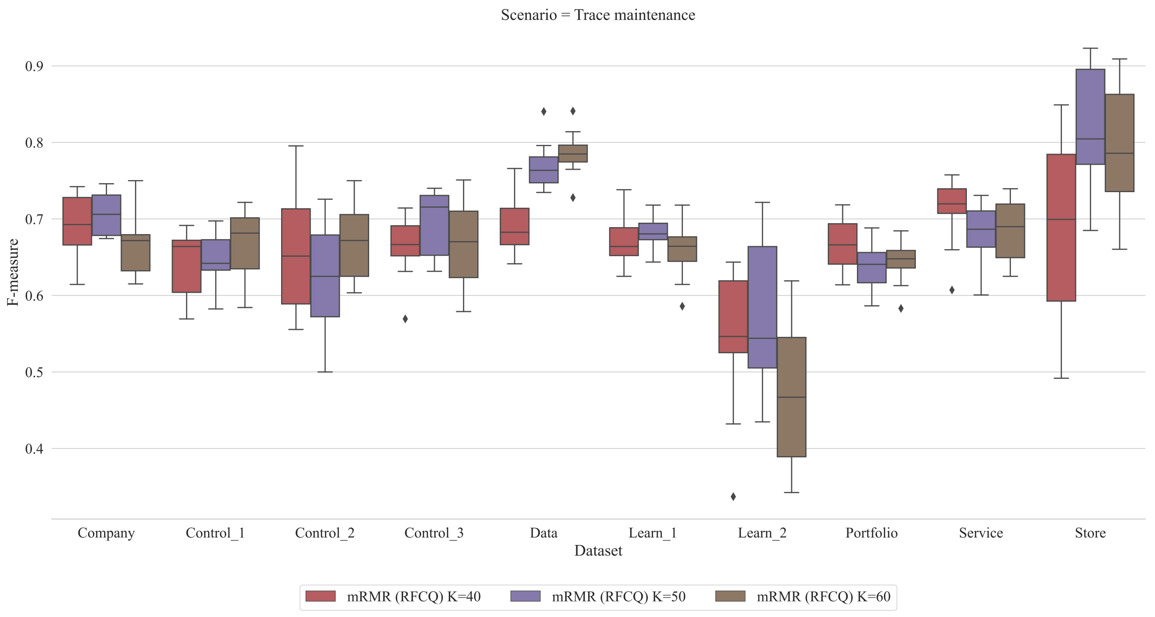


Figure Q4.2: Box and Whisker plots for trace maintenance scenario using the **mRMR** algorithm with **RFCQ** scheme. The X-axis represents the software projects (n=10), with the y-axis displaying the $F_{0.5}$-measure. Each plot is generated from 10 individual classifier evaluation rounds.

R Experiment feature importance results

R1 Feature Importance by mRMR (FCQ)

Table R1: Aggregated feature importance results for the **mRMR** feature selection algorithm using the **FCQ** scheme, for K=40, K=50, and K=60. The feature importance is presented as percentage and categorised per feature family. The table presents the number of selected features for each feature family, aggregated the sum, average, standard deviation, and the maximum feature importance.

			Trace Recommendation				Trace Maintenance			
Feature Family	K	$\overline{N}$	$\overline{\sum}$	$\overline{\overline{X}}$	$\overline{\sigma}$	Max	$\overline{\sum}$	$\overline{\overline{X}}$	$\overline{\sigma}$	Max
Process Related	40	4	64.40	18.48	3.96	67.16	11.52	3.22	1.15	9.11
	50	4	67.13	19.01	3.36	66.55	9.91	2.74	0.88	7.43
	60	4	66.32	18.11	2.77	65.66	9.16	2.43	0.87	6.56
	Avg	*4*	*65.95*	*18.54*	*3.37*	*66.46*	*10.20*	*2.80*	*0.97*	*7.70*
Document Statistics	40	4	2.52	0.61	0.45	2.50	11.30	2.71	0.52	5.91
	50	5	2.35	0.54	0.45	2.31	10.20	2.27	0.51	5.26
	60	6	1.86	0.38	0.33	2.39	8.79	1.86	0.51	5.45
	Avg	*5*	*2.24*	*0.51*	*0.41*	*2.40*	*10.10*	*2.28*	*0.52*	*5.54*
Information Retrieval	40	13	15.56	1.26	0.67	32.17	32.59	2.41	0.51	10.11
	50	14	13.81	0.99	0.39	15.57	30.61	2.05	0.41	9.89
	60	16	13.44	0.86	0.32	17.97	29.01	1.79	0.41	10.92
	Avg	*14*	*14.27*	*1.04*	*0.46*	*21.90*	*30.74*	*2.09*	*0.44*	*10.31*
Query Quality (Specificity)	40	14	13.99	1.02	0.57	7.79	34.73	2.43	0.34	7.99
	50	20	13.03	0.66	0.19	4.40	38.28	1.90	0.23	7.99
	60	26	13.71	0.54	0.20	4.83	41.33	1.61	0.19	9.91
	Avg	*20*	*13.58*	*0.74*	*0.32*	*5.67*	*38.11*	*1.98*	*0.25*	*8.63*
Query Quality (Similarity)	40	3	1.92	0.76	0.73	5.55	4.35	1.71	0.90	3.88
	50	5	2.13	0.52	0.35	3.76	4.96	1.11	0.59	3.75
	60	7	2.43	0.39	0.25	3.72	5.97	0.89	0.44	3.28
	Avg	*5*	*2.16*	*0.55*	*0.45*	*4.34*	*5.09*	*1.24*	*0.65*	*3.64*
Query Quality (Term relatedness)	40	2	1.62	0.78	0.47	3.07	5.50	3.15	2.38	14.34
	50	3	1.55	0.46	0.30	2.07	6.05	1.84	1.07	13.16
	60	4	2.24	0.66	0.71	7.21	5.75	1.52	0.86	13.01
	Avg	*3*	*1.80*	*0.63*	*0.49*	*4.12*	*5.77*	*2.17*	*1.44*	*13.50*

R2 Feature Importance by mRMR (RFCQ)

Table R2: Aggregated feature importance results for the **mRMR** feature selection algorithm using the **RFCQ** scheme, for K=40, K=50, and K=60. The feature importance is presented as percentage and categorised per feature family. The table presents the number of selected features for each feature family, aggregated the sum, average, standard deviation, and the maximum feature importance.

			Trace Recommendation				Trace Maintenance			
Feature Family	K	$\overline{N}$	$\overline{\sum}$	$\overline{\overline{X}}$	$\overline{\sigma}$	Max	$\overline{\sum}$	$\overline{\overline{X}}$	$\overline{\sigma}$	Max
Process Related	40	4	65.29	22.12	14.79	65.90	14.01	4.28	1.62	9.48
	50	4	66.01	16.50	2.22	64.74	13.93	3.481	1.25	8.52
	60	4	65.68	16.89	2.18	64.89	11.49	2.92	1.05	7.38
	Avg	*4*	*65.66*	*18.50*	*6.40*	*65.18*	*13.14*	*3.56*	*1.31*	*8.46*
Document Statistics	40	2	1.81	1.02	0.61	3.24	6.31	3.52	1.51	6.39
	50	3	1.82	0.77	0.50	2.81	6.64	2.61	0.75	4.64
	60	4	2.29	0.63	0.41	5.49	8.39	2.23	0.47	5.23
	Avg	*3*	*1.97*	*0.81*	*0.51*	*3.85*	*7.11*	*2.79*	*0.91*	*5.42*
Information Retrieval	40	5	8.70	1.92	1.35	14.99	15.74	3.07	0.64	9.35
	50	7	9.04	1.61	1.88	20.26	18.19	2.46	0.59	9.38
	60	8	7.70	0.95	0.67	21.56	18.17	2.09	0.35	8.77
	Avg	*7*	*8.48*	*1.49*	*1.30*	*18.94*	*17.37*	*2.54*	*0.53*	*9.17*
Query Quality (Specificity)	40	20	17.54	0.88	0.30	7.49	44.50	2.17	0.23	7.30
	50	27	17.37	0.66	0.20	4.99	46.93	1.75	0.23	8.47
	60	31	16.59	0.54	0.15	3.88	46.31	1.50	0.16	6.30
	Avg	*26*	*17.17*	*0.69*	*0.22*	*5.45*	*45.91*	*1.81*	*0.21*	*7.36*
Query Quality (Similarity)	40	5	2.86	0.53	0.27	2.30	9.02	1.69	0.52	4.31
	50	5	2.56	0.52	0.36	5.98	6.11	1.35	0.39	3.99
	60	7	3.55	0.50	0.32	5.18	6.71	1.00	0.27	4.41
	Avg	*5*	*2.99*	*0.51*	*0.32*	*4.49*	*7.28*	*1.35*	*0.39*	*4.24*
Query Quality (Term-relatedness)	40	4	3.81	0.91	0.55	6.93	10.42	2.57	1.52	16.51
	50	5	3.20	0.64	0.39	3.89	8.19	1.56	0.30	5.08
	60	7	4.19	0.63	0.48	5.30	8.93	1.37	0.15	2.99
	Avg	*5*	*3.73*	*0.72*	*0.47*	*5.37*	*9.18*	*1.83*	*0.66*	*8.19*

R3 Feature Importance by RFECV

Table R3: Aggregated feature importance results for the **RFECV** feature selection algorithm. The feature importance is presented as percentage and categorised per feature family. The table presents the number of selected features for each feature family, aggregated the sum, average, standard deviation, and the maximum feature importance.

		Trace Recommendation				Trace Maintenance			
Feature Family	$\overline{N}$	$\overline{\sum}$	$\overline{\overline{X}}$	$\overline{\sigma}$	Max	$\overline{\sum}$	$\overline{\overline{X}}$	$\overline{\sigma}$	Max
Process Related	3	74.72	23.64	5.13	76.13	42.07	13.41	9.17	56.44
Document Statistics	1	0.44	0.74	0.46	1.88	4.37	7.28	0.56	9.29
Information Retrieval	4	9.13	4.13	4.20	13.14	23.35	9.08	6.38	21.71
QQ (Specificity)	6	11.77	3.90	3.14	10.02	24.08	10.02	10.91	36.76
QQ (Similarity)	1	3.28	3.78	3.60	10.14	5.28	6.84	11.71	18.27
QQ (Term-relatedness)	0	0.67	2.23	2.51	5.12	0.86	2.85	0.91	3.88

R4 Feature Importance by FeatureWiz

Table R4: Aggregated feature importance results for the **FeatureWiz** feature selection algorithm. The feature importance is presented as percentage and categorised per feature family. The table presents the number of selected features for each feature family, aggregated the sum, average, standard deviation, and the maximum feature importance.

		Trace Recommendation				Trace Maintenance			
Feature Family	$\overline{N}$	$\overline{\sum}$	$\overline{\overline{X}}$	$\overline{\sigma}$	Max	$\overline{\sum}$	$\overline{\overline{X}}$	$\overline{\sigma}$	Max
Process Related	3	65.36	28.27	15.29	68.74	12.62	4.95	1.49	8.68
Document Statistics	2	1.32	0.53	0.34	2.83	6.85	3.31	1.64	7.44
Information Retrieval	4	9.78	2.41	1.13	10.68	14.34	3.84	1.29	8.27
QQ (Specificity)	18	15.49	0.87	0.24	6.21	45.15	2.53	0.38	7.42
QQ (Similarity)	4	4.73	1.26	0.75	6.22	10.21	2.66	0.45	7.84
QQ (Term-relatedness)	4	3.33	1.06	1.10	6.59	10.83	2.90	1.01	13.08

S Empty Miro board structure

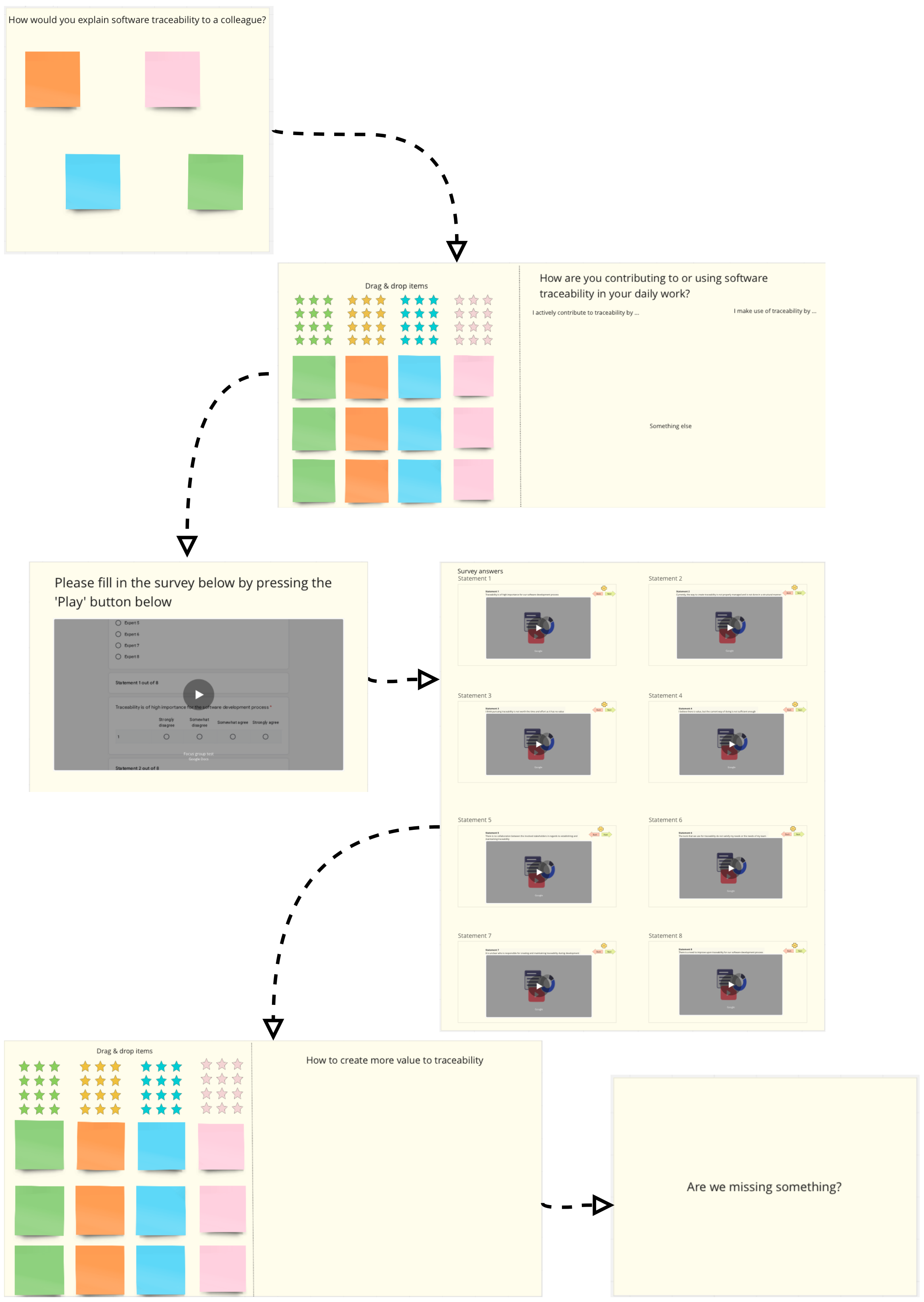

How would you explain software traceability to a colleague?
Drag & drop items
How are you contributing to or using software traceability in your daily work?
I actively contribute to traceability by ...
I make use of traceability by ...
Something else
Please fill in the survey below by pressing the 'Play' button below
Statement 1 out of 8
Traceability is of high importance for the software development process
Strongly disagree
Somewhat disagree
Somewhat agree
Strongly agree
Statement 2 out of 8
Survey answers
Statement 1
Statement 2
Statement 3
Statement 4
Statement 5
Statement 6
Statement 7
Statement 8
Drag & drop items
How to create more value to traceability
Are we missing something?

Figure S: Empty structure of the Miro board that was used for the focus group. Note that this visualisation does not include the opening and closing frames (this is present in the online Appendix [33]).

T Focus group transcriptions

T1 Focus group session A

Session details
Attendance: Moderator 1, Moderator 2, Expert 1, Expert 2, Expert 3, and Expert 4
Date and time: 05/12/2022 at 09:00
Where: Online

Note on the transcript:
The transcript does not include the introduction presentation and the first (welcome) frame on the Miro board. All participants of the focus group are anonymised.

Start of transcript
[Frame: defining traceability]

Moderator 1: Here, I would like you to write down how you would describe software traceability to a colleague.

Moderator 2: Just to add, this is not an exam but we want you to write down your perspective on traceability to have a shared starting point.

[Expert 3 was dropped out of the session for some reason]

Moderator 1: Meanwhile, for the others, it is good to note that the definition is more or less quite similar. Which is good that everybody is on the same page. So, the definition by Expert 2, we are connecting artifacts of project management to entries done into the software version management system. The definition by Expert 4, making sure that we are able to attribute all the work that gets done in our product back to the reason why it is needed to get done.

Expert 2: Is this exercise also to make sure that we are on the same page?

Moderator 1: Yes, that is the main point of this exercise.

Moderator 2: If I may intervene Moderator 1, as we said there is resonance in the definitions. Maybe the invitation for everyone is to think also beyond what Moderator 1 has presented earlier. Because today we want to get a broader perspective of traceability. What Moderator 1 presented was surely one way to think of it, but for instance what Expert 4 has written down is a broader perspective. It is not only about tracing commits to Jira issues, but any time you link certain artefacts it is traceability. We are interested in hearing from you whether the scenario that Expert 4 presented is interesting or is important but also about other scenarios you may encounter in your daily life.

[Expert 3 again joined the session]

[Frame: contributing to traceability]

Moderator 1: So, I think we are ready to move to the next frame. Here, I want you to identify how you are actively contributing or using software traces in your daily work. On the left, you have again your sticky notes and colored stars. You can use these to indicate when you are agreeing with something on a sticky note of someone else. We have a few headliners. The first one is about that you are actively contributing to traceability. The second one is on how you make use of traceability. Something else, maybe something is not part of the first two.

Moderator 2: Moderator 1, do they only have to think about their roles or also about roles of other people from their teams.

Moderator 1: It is fine to also think about people from their team. Essentially, everything that is part of the development process and their experiences with that.

Once everybody is done, we can discuss answers on the sticky notes. I already see a few interesting ones. Also, if you have any questions, feel free to throw them in the group.

For the something else section, maybe you can mention any software or platforms that you use to create or manage traces.

A question to Expert 3. I see these two sticky notes. Are those linked to each other?

Expert 3: In a way what happens, we add the Jira issue identifier to the commit message. When you try to push it to the GitLab repo, the repository will refuse any commit message that does not have an identifier in the message. If so, you have to go back and add the identifier.

Moderator 1: Okay. And is that repo rule in place since recently or already since a few years?

Expert 3: I think we have had it for one or two years now. Because previously, most Mendix applications, by default, they run SVN server. Somewhere about midway last year we migrated to GitLab instance for RD. This gives us some flexibility, because we can have our own pipeline, write on our repository.

Moderator 1: Sounds like a good rule to have in place, where you are forced to include an identifier in a commit message.

Expert 3: Yes, in the older days we had these audits like is mentioned in the rightest column. We had to present evidence that we had a commit was linked correctly to a user story that a person has the ability to approve like our SDMs. That the release scope was correctly in place. We had to really look into our SVN history and compare to our release thing manually. That was not ideal, as it was error prone and labor intensive. In the meantime, while we built an app that allowed us to do this in Mendix logic. So, pulled it from an SVN teams' server and paired it to a release scope. But these days it is part of our pipeline.

Moderator 1: Okay, nice. And Expert 1, I see on one sticky note that you mention: That's a part of refinement sessions where user stories are refined. Can you explain a bit about this? Are you then tracing back to these user stories to see where refinement is needed?

Expert 1: So, what normally happens. Within our teams, when we start a new Epic, we assign an Epic owner. That owner will write the user stories that I will review as well. And during refinement sessions, that epic owner will share these user stories with the rest of the team where they gonna write out the complete story to make sure everyone understands it. They also estimate the work. Normally, I don't have an active role there. I more act as a listener and they can ask questions at that time. But at least, there we make sure that the user stories that they will refine are sort of approved by me as well. To make sure that all stories that end up a sprint are always PM approved. And of course, sometimes that doesn't happen. But we try to achieve that.

Moderator 2: If I may add a question to everyone then. You are all actively contributing to traceability, but much less make use of traceability. Maybe the question is, in your team who is making use of traceability?

Expert 3: To me there are kind of two main dimensions. On one hand, the requirement of standard operating procedures and all the auditable stuff that we have to make sure we can prove that everything is in place. We don't want to lose that certification, which is a measure that we have proper practices in place. The other area is, as a developer, I want to understand why something has changed, months after the fact. For instance, why has this disappeared.

Moderator 1: And Expert 4, you mentioned that you keep track of the status of stories that are in implementation or are done. Are you personally involved in tracking that status?

Expert 4: It is more like, I want to be informed about where we are. Because when we are done, probably we want to start a new process or communicate this towards customers, so that it is available to them. That is why I wrote that down. Otherwise, the traceability like Expert 3 just mentioned is very helpful, but that is not something I actively participate in. Like tracing back why something changed or something broke. But I definitely see the value there. It is more like a to do, something we have to do to be compliant indeed. It is not really adding that much value in my day-to-day process if I may be honest. It is just good practice, but it is not something that I think about a lot.

Expert 3: I think it might be different in organization that have anonymous people or have a lot of people contributing to a repository. In smaller teams, I think it is rarer that you actually need it because you know each other, you usually know what is going on. You only use it once you need to go back a bit further, once it is out of your memory. In our team we have like 10 or 11 people in it.

[Expert 2 was dropped out of the session for some reason]

Expert 1: I think in our process we have a sort of key role, that is the software development manager. What happens before we can even release something. The team needs to ask for a release approval. A release approval consists of a list of stories that are included in the release. For instance, what happens if something goes wrong, sort of a rollback situation, a rollback plan. That needs to be approved. If that is approved, this is also then logged and the team is able to release. What we have since a couple of months, that we can only release on a Thursday morning.

Expert 4: Does it ever happen that someone, like an SDM, blocks a release because of anything on that list?

Expert 1: Yes, that happens. When I did that in the past, at that time the rules were a bit stricter. Because I had to check all the user stories and see if the stories where set to Done in Jira or at least had a label of QA okay. And sometimes that wasn't the case, and then I had to refuse the release. But then within 5 minutes, they changed the stories to make sure it was reviewed. Often, they did test it, but they forgot to add the label.

[Expert 2 again joined the session]

Moderator 1: Is there anyone who want to add or finish something that they were working on? Otherwise, I think we have some great input here and we can move onto the next frame.

Expert 1: Yes, Moderator 1. I have a question to Expert 2 and Expert 3. Is an ADR a form of traceability?

Expert 2: I'm not sure if everybody knows what an ADR is. So, what we do is Architectural Design Records. We need to use that if we have a decision to make from an architectural perspective. We write that down and we have at least three directions, all with their pros and cons. Out of that, we choose the own everybody agrees on. We also use that as a way of traceability, when in 6 months' time. Why did we make this decision? So, this is traceability on a way different level, but a very important one. Especially when you have turn around people or new people joining.

Expert 3: Yes, it is not the same type of traceability as with commits and issues. But it is another form of traceability.

Moderator 1: So, I think we are ready to move towards the next frame.

[Frame: Rate the statements]

Moderator 1: When you press the 'play button' you are met with a Google Form where you are asked to rate a number of statements on your view on traceability. Once everybody has done that we will discuss the answers on those statements.

[Frame: Survey answer discussion]

Moderator 1: Expert 1, could you explain your answer on the statement 'the way to create traceability is not properly managed and is not done in a structural manner', since your answer deviates from the others?

Expert 1: The thing is, I have no clue. I don't know how stories are linked to Jira issues and if this even happens. So, I agree to this, because if something is properly managed then it should be transparent and every team does it in the same way.

Moderator 1: Do the others share the point that Expert 1 is making?

Expert 2: I think this could be the case, on the other hand we have software development managers who are responsible for this part, so I expect they have everything place. I don't have a complete overview, because we are so big and have so many teams but on the other hand I don't really care. As long as it is fine on our end, right.

Expert 1: There is a point that indeed, maybe it is the responsibility of the SDM. And if the SDM does not complain to us or ask for changes in the current process, maybe it is fine.

Expert 3: Speaking for my team, I think we have everything pretty well in place. I almost went for strongly disagree, but for documents other than the Jira issues, it is a bit harder to find these traces usually because these do not have a standardized way.

Moderator 1: Okay, and then for the Jira issues and commits?

Expert 3: Yeah, that is pretty much water tight.

Expert 4: I think we are quite religious about that. Nothing gets done without a story.

Moderator 2: Maybe a clarification, how are SDM's actually structured around the company?

Expert 1: Every team has an SDM and an SDM could have one or more teams.

Expert 3: Usually, two or three teams if I recall

Moderator 1: Okay, for our third statement we can see again some differences. The statement is about 'pursuing traceability is not worth the time and effort as it has no value'. Again Expert 1, somewhat agrees to this while others are more on the disagree side.

Expert 1: Yeah, for this one I do not know how much time it takes. And if Expert 3 says a lot of things are automated.

Expert 3: If you follow the process, it does not take a lot of time. It is more, once you go off-road, then things get a bit more annoyed. Or if you mispronounced an identifier.

Moderator 1: Does it happen often that something needs to get fixed or that you get of the trail?

Expert 3: No, the most common thing that happens is that you find someone has made a typo in the issue commit, which is difficult to find by the system. Especially when the typo refers to a different story, you kind have to search for what is this story again. Usually, as part of the release story in Jira we have a not with the correct Jira link there.

Moderator 1: And are you then automatically get notified when the issue has a wrong identifier, or when somebody is looking into that, only then it gets notified?

Expert 3: There is one step in the pipeline where we pull those GitLab commit messages and the Jira release scope and we compare those two. As long as the typo changes to an identifier that is not part of the release scope, at that point the script will note that there is a commit referring story that is not part of the release scope. Of course, when you accidently miss one number and therefore refers to a different story that is part of the release scope, as stories can be quite close to each other, you might overlook it.

Moderator 1: So, for the next statement, which was about ‘I believe there is value, but the current way of doing is not sufficient enough’. Maybe we can start with Expert 3, who agrees on this statement. You believe that the current way of doing so is not sufficient?

Expert 3: I am wondering if we can get more out of it, when we are looking at different documents outside the scope of Jira issues and the commits. You would really want to trace from Jira issues, to the commit, to the design document, essentially through the entire development process. There we could do more I think. Strictly speaking about Jira issues and the commits, I don’t see any issues.

Moderator 1: It could be a more complete picture if you could include more documents, so to say?

Expert 3: Yes.

Moderator 1: Expert 4, you indicate that you strongly disagree to this statement. So, in your opinion the current way of doing so is sufficient enough to support the development?

Expert 4: I think our current doing is sufficient. I wouldn’t want to spend more effort on this in the current status.

Moderator 1: And why so, is that related on your view on the value of it?

Expert 4: I think we are delivering the value so any more effort would be a waste of time in my opinion. That time we can better spend on the project, rather than bureaucratic things.

Moderator 1: That’s clear, thanks. For our fifth statement, which is about that there is no collaboration between the involved stakeholders in regards to establishing and maintaining traceability. We see most of them disagreeing.

Expert 1: I think I haven’t read ‘no’ in this statement. So, I would also disagree with this. I think there is a collaboration. The only question is, who are the stakeholders? I have no clue. If we are not part of it, how should we know?

Expert 3: I think we are all stakeholders in the process, because we all take part it. You have the SDM, who is the formal person to be responsible to make sure this is all in place. Then you could go all the way up in command. Besides the SDM, the compliance department might be a stakeholder as they in charge of making sure all these audits and things are maintained correctly. But for all these things, they are not actively involved.

Moderator 1: So, mostly the people from the team are the ones that are actively involved in the traceability process?

Expert 3: In my experience, in terms of compliance, the document states what should be done and it’s up to the team to make sure that it happens.

Expert 2: I think when we started with this, it was all high urgency and high priority, but now since we have everything in place they do not look as much at is as they used too.

Expert 3: Yes, I think after a number of cycles of audits we build enough trust that they did not bother to guide us as specific anymore.

Moderator 1: So, in the beginning there was more attention to it to have it properly maintained?

Expert 2: Yes, from the external auditor's perspective. From internally, we still keep up the good game, because we don't want to let it slip and then pay the burden later, right.

Moderator 1: The sixth statement was about the tools that are used for traceability and them not satisfying the needs. I think from everything we discussed until now, the tool that was introduced one or two years ago, the repo rule, which satisfies the needs for traceability as you are forced to include the identifier.

Expert 3: That is something we did for my team specifically. I don't know if this is done for other teams as well.

Expert 2: I'm not that close to the operation anymore. But we do not have the same tool in that sense as you do, but we check that a commit is always linked to a story. So, I do think everything is in place. There will always be room for improvement. So, looking at the machine learning solution, I think it is better to solve it at the entry point instead of using the machine learning approach to solve everything that is being missed. Then the investment will be bigger.

Expert 3: But it could be part of the scenario where you are presented with issues that might be related.

Moderator 1: You might be more involved if you are indeed actively making the links instead of everything being linked on the background.

Moderator 2: Definitely the first scenario that is interesting, that we picked. Where you will be given recommendations and you pick one. The second one is what Expert 2 was talking about, which is riskier I would say.

Moderator 1: Okay, for the last two. The statement 'it is unclear who is responsible for creating and maintaining traceability during development'. Especially the answer between Expert 4 and Expert 2 and Expert 3, the answers differ the most. Expert 4, you agree on this statement, that this is indeed unclear.

Expert 4: Today, I learned that the SDM is the one who is responsible for this, so now it is clear to me. I wouldn't be able to draw this process out to explain it to you. This would mean that is not that clear to me.

Moderator 1: I see, yes. I think we can go to the last statement, which is about the need for improvement upon traceability for our development. Expert 3, you agree on that, that you feel there is a need for improvement. Can you explain yourself about this?

Expert 3: Yes, this goes back to an earlier question we had. I think we can do a bit more tracing beyond Jira issues, in that area I think it is fine, except for the tooling. For now, that is custom code per team. The main reason for my answer is that, if we want to trace further and have a bit more like a structural way of linking things together, might be useful.

Moderator 1: I see. And for the others, they all somewhat disagree or strongly disagree on this statement.

Expert 2: Yes, for me it was either somewhat disagree or somewhat agree. Basically, what Expert 3 said. If you consider only Jira and commit, I don't think there is much room on improvements there. But, we can definitely improve the part where we go from product ideation to user stories. But I also think different teams and different units have different ways for that. So, I think there is an opportunity to structure that a bit. However, I also feel like teams have the freedom to choose their preferred way of running the team.

Expert 3: We also see that it already works way better than it did a couple of years ago.

Moderator 1: Let's move from the statement to the last frame.

[Frame: Create more value]

Moderator 2: This frame is an elaboration on the last statement, right Moderator 1?

Moderator 1: Yes, that correct. On this board, I want to identify how we can create more value to traceability. You can again use the sticky notes and stars as you did previously.

A question to Expert 3. You mention in one of your sticky notes that tooling should work out of the box. Can you explain what you mean by that?

Expert 3: That is about that we have to create this traceability logic that we use right now ourselves. We have written the script that enforces the commit message to include the identifier. That is only done for our team specifically. So, my team benefits from this, other teams don't. It doesn't mean it is the most efficient solution, but it relates to the sticky note from Expert 4 that is about improved automation and standardized processes.

Moderator 1: Yes, and maybe also to the sticky note from Expert 1, about having every team work in the same way.

Expert 3: Yes, that would help for this.

Moderator 1: Expert 2, you mentioned that there is still some manual work.

Expert 2: You have to look for the identifier of the story that you have been working on. Maybe you have to back to Jira or search for it in your IDE. So, if the identifier is somehow recommendation that would be nice. If it picked the most relevant user story from your current sprint. If his happens when you are doing the work, then you still now about that established link. If the system is going to decide for you, for instance at the end of a spring, you really have to look at what you did and it becomes more work than when you did it at the start.

Moderator 1: It also might help to prevent mistakes in the Jira identifier.

Expert 2: I think it would be really a great win if the system provides you with several options and you select the right one. Then it is just a click.

Expert 3: I think the burden on a commit is quite low and it will give more confidence than a system that is not transparently deciding on what things should be linked.

Moderator 1: Then I would like to discuss the note by Expert 3. You mention you want to be able to more intrusively navigate from a commit to a Jira. Do you than mean a dedicated platform where it is clearly visualized, the traces between a Jira issue and a commit for instance.

Expert 3: For our current Git, I am not sure where you would do this. It is more like, right now you find a commit and you find a user story and then you have to go to Jira yourself. It would be nice to have some sort integrate experience which is of course nice. We did have in sprinter is a developer portal, which is part of the unit of Expert 1 and Expert 2. Once what we had was in Studio Pro was before the commit, you could optionally select the stories that were in the current sprint a being what that commit was about. And then those commits' meta data found its way to sprinter were then you could see for a commit which story it belonged to.

Moderator 1: I think what we see in this frame makes sense considering all the stuff we have been talking about. Also, the last one from Expert 2 fits in this picture, to include more into traceability as currently is the practice.

Moderator 2: Maybe a quick question from my side, did the acquisition by Siemens changed this process at all?

Expert 3: No, all was already in place. If this was not in place, they probably have changed it indeed.

Expert 2: What was not mentioned until now is that you really have to see the benefits of doing this. Otherwise, I think is a lot of resistance to start this.

Moderator 1: I think we can wrap up our session. One last question is, are we still missing something?

[Frame: Missing anything?]

Expert 1: If you have more time for a chat, I think it is important with someone who is an SDM and see how they experience it. Also, maybe someone who is responsible for the audits to also get their point of view.

Moderator 1: That's something good to keep in mind! Thank you all for your participation and sharing your expertise!

T2 Focus group session B

Session details
Attendance: Moderator 1, Moderator 2, Expert 5, Expert 6, Expert 7, and Expert 8
Date: 06/12/2022 at 15:00
Where: Online

Note on the transcript:
The transcript does not include the introduction presentation and the first (welcome) frame on the Miro board. All participants of the focus group are anonymised.

Start of transcript
[Frame: defining traceability]

Moderator 1: I would like to start with the first frame, which is about defining traceability. On the screen, you see several sticky notes. Each one is assigned to one of you, with a separate color as well. These colors are used throughout the Miro board. I would like you to explain how you would define traceability to a colleague.

Moderator 2: Yes, and there is no correct or wrong answer. It is mostly about getting each other's perspective and get a shared understanding.

Moderator 1: So, we see some similar definitions, where you are talking about tracking new and old changes made to the software. Also, to find the underlying requirements or change requests where you want to be able to prove why changes have happened. Indeed, between certain artefacts, like requirements, user stories, changes made, features to design elements. So, I think we can carefully state that we are on the same page at this point, which is good. If everybody is done providing their input, we can move onto the next frame by pressing the next button.

[Frame: view on traceability]

Moderator 1: At this frame, you can see the same sticky notes as we saw before. On top of the sticky notes, you can see some start which you can use to express that you are agreeing on a point that somebody else is making. My question here is to think about what your view on traceability is. What are your experiences with traceability, across the whole development process. Again, we are not limited to traceability only on commits and Jira issues.

Expert 7: I'm not really sure if I can answer any of these things here in terms of software development. I stop really early on, with gathering some requirements. Where it continues in the software development is not really my field.

Moderator 1: And within that part that you are involved in, can we speak of traceability aspects in that sense? Or maybe you have experiences in the field of traceability from the past?

Expert 7: I'm not sure. Let's see what the others are writing and I will see if I can relate to some of them.

Expert 8: Does it have to be specific to a particular platform, like Mendix?

Moderator 1: Well, our main focus is on traceability in low code. So, it could also include other low code platforms as long as it stays within the low code spectrum.

Expert 8: Okay, thanks.

Moderator 1: You also might want to include aspects of model driven development, as that is also related to low code, but really within those respective fields.

I see from both Expert 5 and Expert 6, maybe these sticky notes are related to each other. These are both about making a commit, you include the Jira identifier/reference id in the commit.

And Expert 5, if I may ask you a question on your second sticky note. I will let you first finish the note that you are working on right now.

Expert 5: Yes, what is your question Moderator 1.

Moderator 1: Yes, so you mentioned that on a release level, you create release version in Jira and then assign a certain name or label to a specific app version. And that is than so that for each version of the app, you are able to trace that back to a certain requirement or a user story that is defined somewhere?

Expert 5: Yeah, so what we do, we work on a couple of stories. So, where the requirements are written. Then every story we link to a version, we make use in Jira of the release capability. What we simply do is we just name that release to the specific app version. So, if we have an app version like 1.2.1, and when we do a next release the week after, then all the stories we include in that release, we link them to that version and name it accordingly. There is not really a hard link, but kind of in naming you could look back if would go to your, for example in Mendix terms, to your sprinter environment, like we are now running on production version 1.2.0.1. Then we could look back in Jira, what happened in that release and see all the stories related to that. That kind of traceability is what we have as well. That is what we are currently doing at least within our team. I'm actually used to another environment to do that even a bit stricter. There, for every commit we did, that automatically resulted in automatically build of the model and the deployment to a test environment. All the committed code resulted in a build of a container of the whole Mendix app. And then we wanted that to promote that to a higher environment, we actually need to run release pipelines. And these release pipelines where then also linked to the stories. So, we could also trace back to which release the story was related to. That was a manual activity. But the release pipelines that also triggered towards production also automatically triggered a change request into a 'service now instance' and in that service now instance, people had to approve. So even there they could see if there were support tickets, they could see for example, last week there was a release. And then they could see, from a build perspective, with the release pipelines and build perspectives, which artefacts were there. And then you could trace that back to the commits, though this required two different systems. But what so far has happened, but maybe Expert 6 can how they do that a bit more, but before you do a build, as a developer you already did a bunch of commits and there, people make mostly use of the comments before you are committing. And there is then the obligation to mention the Jira ticket in the comment. So, not really visually linking them but it is more naming-wise. Expert 6, I think you also are doing some traceability with branches, right?

Expert 6: Yeah, we used story branches depending on the size of the story. Really, sometimes it is on an Epic level, if logic needs to be combined into a single branch. We use the Jira ticket number for that and add a short description. Every commit we do to that branch also contains the story number and the changes that have been made to that commit. After, we go through the reviewing process, the co-review we have and the testing. The changes get merged back to our development line, also mentioning the Epic or story number, the one that is relevant at that point. That also gets ported to main, so every change is pretty much traceable throughout the entire process using the Jira ticket number.

Moderator 1: And is that something that is mandatory, like can you continue with the commit without including these Jira identifiers.

Expert 6: No, it is something that we agreed upon as our best practice, to always do that. You need the traceability for instance when a change has been made and you can't trace it back to the story. Then it can get messy quite quickly. For example, if changes have been made to an endpoint. At some point, it was determined that another team was not ready for those changes. But we didn't decide the version before that. That was decided after the issue. This would cause issues for the other team if we would continue releasing the version, because they could not yet handle those changes. By using the ticket number, we could find all the files that were affected and could roll back the changes that were necessary. This is not a good example of best practice, but the ticket number allowed us to trace back and fix what went wrong basically. We definitely see it as required, but there is no system in place that checks if we actually do include Jira identifiers to the commit.

Moderator 1: This sounds like a very good recent example of why it has indeed good value to create these traceability aspects.

And Expert 8, in one of your sticky notes you mention that you write unit test and then attach it to user stories. Can you explain how that is done, how are you attaching them to user stories?

Expert 8: So, previously we were using the Jira dashboard. Now, we are using a different in-house platform for managing the user stories and features. For these user stories, we are attaching the test cases, which are covering all the use cases for that feature. These are attached to the user story manually, in Excel. What we observed is that our code reviews are not mandatory in Mendix, being it a low code platform probably. What we found is, even without the code reviews by peers, developers might check in the code. In my second sticky note, I have mentioned about the difficulties for traceability. What I observed was, while doing development, can see the changes made to the original source code and can mark out the differences. Once you commit to your code (using Mendix), you don't get to see what changes were made between one version to the other. In my third note, I see difference between code full environments, where I see the difference between two source code files. In low code, I don't think there is a comparison functionality build into the development environment.

Moderator 1: I see Expert 6 shaking his head, confirming the things that you are mentioning.

Expert 7, do you see one of the notes that you can relate to?

Expert 7: Yes, of course. I can relate most to what Expert 8 is saying, because that leads to problems on my side. Where a manager tries to release this app as a SaaS product to customers. That would lead to problems when you need to upgrade or when a customer wants any customization. Maybe not in a SaaS product, but when we try to deliver it on premises. If we would allow any customization to this, it would be difficult to handle. This is actually, currently, proponing our release date, to understand how to work with that.

Expert 5: Maybe, it is a bit strange of course to place like comments in low code. In full code you are used to also sometimes place a commit, mentioning that you are doing this change on behalf of this request number. Then they could look it up in some kind of ticket. And indeed, if you do then some code comparison, it could clarify what then was actually changed. What we sometimes do is in the microflows, we place these comments in. For instance, at which date we did that and for what reason we did that.

Expert 6: Yes, to add on that. For the more complicated flows, what we do we add comments there with the date and the person that actually made the changes to clarify the process. That can be useful as well sometimes, when we look for changes.

Moderator 1: But that is then not necessarily to be traced? It is only there if you need that information when you are working on that microflow.

Expert 6: we have tried to also use the Jira ticket numbers. We have been experimenting with different ways to do this, but at the moment we went back to the naming system. But it is not directly linked to a ticket, so that makes traceability a bit harder. Unless, the also place comments in the commit message and refer to that for example. To add as well on the comparison between the code. I feel that is definitely something that is missing in low code, but there are ways to work around it. You can see what files have been changed in your history. What some people do is to create a separate branch and merge the changes to that branch to see what has changed, that is also an option. But the convenience that you can sometimes see, when you make a pull request and then see the changes there, that is not something that exist, at least in Mendix.

Expert 5: And what I used to see, when we just started with low code at another company. Before working for several days on different things, and then committing. But what we did over time in most of the teams, that user stories and the changes and so on, become much smaller. It became smaller, so it was easier to see what has been changed, before and after. I feel it is also a bit of the characteristics of the project that you are working on.

Moderator 1: That it is kind of evolving over time?

Expert 5: Yes. It had something to do with the practices, but people did change over change over change. Then it was really hard to see what did you really do.

Expert 6: There is still some traceability in there. When you merge the changes to for example the development line, you branch of, you make a change there and then merge it back, you can see the specific elements. For example, if you click on a microflow, you see which values have been removed, which values have been added. That is probably the closest thing you can get to having a comparison between the new one and old one.

Moderator 1: Alright. I think we can leave this frame and move onto the next one.

[Frame: contributing to traceability]

Moderator 1: For this frame, I am interested in how you are contributing or using software traceability in your daily work. We also discussed a few points in the previous frame that might link a bit to the thing that we are discussing here. You can again make use of the sticky notes as last time.

Moderator 2: I think since the previous frame already contained a lot of information. Here, the idea is just to add if there is anything where you see yourself contributing to traceability that was not mentioned before. And actively contributing to traceability means that you help tracing the artefacts, where using means you are using the links created by someone else.

Expert 7: Again, I do feature and then the PO does stories. I think in this case, we should involve the PO more in this topic. This is not what I do, I do not create the traces or make use of it.

Moderator 1: And, in what way are you maybe use the links to find where these challenges are? Or these traces are not part of your problem-solving process when you find certain issues?

Expert 7: It's more the generic, okay if we have a problem and we have to update the customer. What does this mean to our product and our possibility to release it on premise, what is our usual process in Team Center to do that. But, I don't really interfere with the actual traceability between a user story and the actual code. I never go down to the code level.

Moderator 1: I see, yes. In that sense, it might be hard to include your experiences on this frame.

Moderator 2: So, Moderator 1. I think these scenarios were elicited in the previous frame.

Moderator 1: Yes, I agree. I think we can move further to our next frame.

[Frame: Rate the statements]

Moderator 1: When you press the play-button in a second, you are presented a Google Form that presents 8 statements. I want you to provide your view on these statements and when everyone is finished, we can discuss them.

[Frame: Survey answer discussion]

Moderator 1: For these statements, it is more interesting to discuss the statement where we see the biggest differences. I think we got a clear picture on the first statement, where everyone sees the importance of traceability in software development. If we move on to the second statement, this was about the way of traceability being created and that this is not properly managed and not done in a structural manner. Especially, Expert 8 and Expert 5 and Expert 6 differ the most, where Expert 5 and Expert 6 are disagreeing on the statement and Expert 8 is agreeing on it. Expert 5, would you like your arguments, why you disagreed on the statement.

Expert 5: Yes, I was a bit mixed about it. I feel that, still too much, the fact there is traceability depends of user discipline, where you have to manually have to include certain Jira identifiers in commit messages. Of course, it is standard practice. So, there is some structure and sort of governed a bit. Still, it is error-prone, I would say. Hence, I was doubting if I can really say it is done in a structural manner. After some time, people forget about it and if you would fill in something different, people wouldn't notice or figure it out late. That was why I was disagreeing a bit.

Moderator 1: Does it happen often or is it a bit issue to solve, when an error is made to create a trace?

Expert 5: I think it is not so that stuff doesn't work. It's like, when it happens we identify that we should do it better. It was when you really have a pipeline and you can't push it to the next state as you are not meeting certain criteria. That makes such things stricter. It is also a bit of a responsibility.

Moderator 1: I see. And Expert 8, you said that you agreed on this. So, in your opinion it is not properly managed. Can you explain a bit about your answer?

Expert 8: What we observed during things we encountered during development when we started developing on Mendix. The first thing is the creation of branches, which is only allowed in Mendix Studio Pro. So, it is like only a developer can create a branch and there is a flat list of branches. So, everything is manually here, you have to name your branch manually. It will note be shown as a hierarchy of branches. That makes the life of the branch management person very difficult. Then, what I observed is that anyone can check-in on any branch in Mendix today. That makes it difficult to manage and control, since the low code branches were all open and not secured. Say, you create a hotfix branch or a release branch, typically it was needed to be frozen. So that no other changes can be made to a certain branch. Here, those things make life of a developer very difficult. Second thing is, the code reviews are not mandated in low code, I don't know why. There is always a chance that code will go in without a code review. I worked on enterprise application, where without a code review done, no single line can go into the main branch. These restrictions are not there, which makes me nerves. It is a too open system. These are the things why I thought things are not properly managed. Then again, when it comes to data migration, Mendix did not give any guidance on it. What I observed, when we upgrade a particular application, for instance, the schema down a database gets upgraded, the data remains on the older schema. If you are in the production environment, you have to make sure that your data is migrated to your new schema. Mendix, being a low code platform, there are no constraints on the database tables, like a foreign key. What I observed is there is an association table between two entities and all the constraints are controlled by the Mendix domain model. Because you can have the cascading rules and cardinalities set up in the domain model, while the database remains very open. In code full like applications, you typically design the database first, or at least you have the option to run migration scripts in DC or PL/SQL and you can fix all the data and bring it to your schema, which is not guided by Mendix. What we are thinking you is creating our own microflows and hook it up to on-start microflow in Mendix. The first thing it then does when starting your application is to older data to your new schema. So that are some key highlights of what we have to take care.

Moderator 1: Thank you for your elaborate answer. So, for our third statement, all are agreeing on the same level. Where you all agree on that traceability is worth the time. Expert 6, could you maybe briefly explain your answer on this one?

Expert 6: Yes, I already mentioned some points on a previous slide. But, it's a great tool to have, just to track where things have happened or have changed. It is really useful when you have to check things or when you have to solve problems and check status. You can't go without it.

Moderator 1: I think we can all agree on that. For the forth statement, we see again some differences in answer. The statement is about the current way of doing is not sufficient enough. Especially Expert 6 is disagreeing on this, while others are agreeing. Maybe we can start with Expert 8 and hear what made you strongly agree to this statement?

Expert 8: Yes, sure. I think there is definitely a value in managing the traceability. But current way of doing is indifferent in certain ways. The code full way of software development is pretty advanced by now and many things are available now for the development community, from examples to tooling support. For low code, the simple things are good when it comes only to development. But when you start thinking about the real applications, not like MVP's or prototypes, but real world or enterprise applications, it really needs a lot more tooling and support which is lacking today. For example, the branch creation, grabbing and giving, on the Mendix online platform itself, the sprinter environment. We see the environment and the team creation and the apps on the cloud. Those simple things could have helped the life of the operations team easier. Right now, if somebody of the operations team, which is different to the development team, has to manage the branches, they need a source code full access along the Mendix Studio Pro. This should not be required, but today that is the case. This is one example of the things that are lagging still in the development environment.

Moderator 1: Alright, thank you. And Expert 6, since you disagreed on the statement. Maybe you could elaborate on this?

Expert 6: Well, let me elaborate indeed. I felt some difficulty on this one. I think there are two perspectives. Looking at how our team currently copes with the work and traceability, I think we are managing what we are having, but I also strongly agree on what was just said. I think the tool can be greatly improved on that part. It is definitely lagging on some areas. This was maybe from a team perspective how we are coping with the instruments we have at the moment.

Moderator 1: Yes, that's why I think it is good for us to discuss our answer on these statements. With that, I would like to move to our fifth statement. This one was about there is no collaboration between the involved stakeholders in regards to establishing and maintaining traceability. Expert 7, you somewhat disagree on this.

Expert 7: For me it is more about the manual process, that is depends today on the collaboration on what you do to establish anything. As mentioned, there are ways where it depends on people following certain processes, what would require some level of collaboration.

Moderator 1: Alright, thank you. Because of the time, I would suggest we move over to the next one. This one might be interesting, which is about the tools for traceability not satisfying my needs or the needs of my team. Can Expert 5 maybe share your viewpoint, as you are disagreeing on this?

Expert 5: I feel it works for us. For me it is often to see when did we change what. That does satisfy my needs. I feel that you always need a certain process around it, or somethings you have to do. I can image that our tools to not cover all the aspects of traceability and remains as manual tasks. If we are using Mendix, it is a platform, as for Azure DevOps, you have your requirements, your testability, your releases, your code. It is all in one place while often linking to things, or integrating things and improve traceability. Because it is all in one tool, these things can talk to each other. In all other setups, you often depend on heavily integrated that or a process around it. In those perspective, Mendix is a full development platform that includes all those things. But as you mentioned in the beginning, Jira is really outside the Mendix platform. There is already one thing that that I need to process or some kind of integration to make sure that this traceability requirement you have as a software team, that these requirements need to be fulfilled, mostly by a process. Then out of the box tool does not directly support it, if that was really the case then I would strongly disagree. There is still room for improvement. What is maybe interesting, in Mendix there is an ability to define stories in the sprinter environment. I think there is an integration out of the box. But I also doubt if people are using that, maybe if you are working on your own.

Expert 6: We used to use that.

Expert 5: What is also sometimes tricky is configuration. You can build the code, but most of the issues we have is mainly configuration data.

Moderator 1:	Right, that's clear. So, briefly looking at our last statement. This is about there is a need to improve upon traceability for our development. Expert 8, you already mentioned a few things that we might improve upon. As you agree in the statement, are there certain aspects on traceability where you feel we might benefit the most on improvements in that respective field?
Expert 8:	I think the main thing is is the ability to see the changes which we made, without to go back and download the code from the last check-in or branch on my machine. I just simply want to be able to compare it.
Moderator 1:	Like you mentioned earlier, you want to be able to have them side-by-side to make those comparisons.
Expert 8:	Yes.
Moderator 1:	Okay. This is it for the statements. I would like to continue to our last main frame for today.
	[Frame: Create more value]
Moderator 1:	In this frame, I'm interested in your perspective how we can create more value out of traceability. Maybe, you feel that you are already getting the most value out of it.
Moderator 2:	As already many things have been said, we can ask the participants to mention things that we not mentioned.
Expert 8:	We did not discuss the defects. How do you guys add defects in Jira and then manage it. Do you find traceability differences with the user story and the code and the defects and the code, or is it the same?
Expert 6:	Those get handled different from the other stories. At least in the process, they are essentially the same but a different tag. Depending on the kind of fix, if it is a hotfix it is handled in a separate branch. If it can wait until the next release, it is handled in a fix branch and will merge when we release the other stories. These are the main differences, so mainly the ID's.
Expert 8:	Okay, thanks.
Moderator 1:	When Expert 5 is finished with his last note, then we can wrap up and go to our last frame.
	[Frame: Missing anything?]
Moderator 1:	If you feel we missed anything important for this topic, you are free to add something to the board. Otherwise, I want to thank you for your participation and sharing your expertise and all the input during today's session.