Utrecht University

Master's Thesis

A Machine Learning Approach for Requirement Traceability in Model-Driven Development

---

A thesis submitted in partial fulfilment of the requirements for the degree of Master of Science
in
Business Informatics
Faculty of Science
Department of Information and Computing Sciences

Author:
Randell Rasiman
4281209

Supervisors:
Dr. Fabiano Dalpiaz
Dr. Sergio España Cubillo

August 2021

Abstract

[Context & Motivation] Requirements Traceability (RT) aims to follow and describe the lifecycle of a requirement. A multitude of standards require RT practices because they provide benefits in project management, project visibility, maintenance, and verification and validation.

[Problem] Many of these RT practices are carried out manually, which poses significant risks. Manual tracing techniques are prone to mistakes, vulnerable to changes, time-consuming, and difficult to maintain. The task of recovering traces should not be done manually but should instead be automated. However, this is an issue since existing automatic tracing tools have shortcomings, as evidenced by the low tool penetration.

[Results] We propose to tackle this problem by using machine learning (ML) techniques. This research presents the design of a tracing tool for automatically recovering traces between JIRA issues and commits in a model-driven development (MDD) context. Using process and text-based data, we created 154 features to train a ML classifier. This classifier was then validated using four real MDD industry datasets. We were able to get an average F2-sore of 73.48 with the best tested configuration, for a situation where we could recommend traces to a developer. An F0.5-score of 77.32 was obtained in the scenario of automatically maintaining traces of a current project.

[Contribution] The findings of this study demonstrate that state-of-the-art trace recovery techniques can successfully be implemented in an MDD-context, bridging the gap between academia and industry.

Keywords: Requirements Traceability, Machine Learning, Trace Link Recovery

Contents

List of Figures

List of Tables

Acknowledgements

This was the end of an eight-month journey in which I wrote my master thesis. In those eight months, in between the lockdowns, curfews, and vaccination appointments, I suppose I have learned a thing or two about requirements traceability, model driven development, and applied machine learning. However, this also marks the end of my time as a student. And at the end of such a life-defining phase, I am grateful to the individuals that brought me here.

First, I would like to thank my supervisors. Dr. Fabiano Dalpiaz, who guided me throughout the entire journey and probably is the sole reason I am still graduating this academic year. Thank you for your supervision, patience, and faith in me. Dr. Sergio España Cubillo, who was a great supervisor, but also a great teacher. I still recall your enthusiasm during the ICT advisory course, and for that I am grateful.

Then my acknowledgments to the people at Mendix. Toine Hurkmans, who introduced me to Mendix. Thank you for the meetings and the assistance. My thanks also go to Joep, Benny, Arjan, Thomas, Jonathan, Omar, Huib and other individuals at Mendix for sharing their expertise.

Then I ‘d like to extend my thanks to two of my friends, who were of great help to me. Amir, thanks for the countless grammar mistakes you’ve fixed. Have a good time in The Hague and stop by for a game of tennis. Daniel, thank you for all the library sessions, and for proof reading. Now it is your turn!

Finally, I'd want to thank my family. Thank you, Mom and Dad, for inspiring me to pursue my academic goals. And Abigail for simply being present.

1. Introduction

Requirements Engineering (RE) is the discipline concerned with the identification, management, and evolution of requirements [1]. RE activities form the foundation of every software engineering project [2]. They define and communicate the needs of the stakeholders involved and what a software system must do in order to satisfy that need. During the many phases of a software project, numerous requirement artefacts are created. The documentation of these requirement artefacts may range from user stories to class diagrams [3]. It is therefore not a surprise that RE is a topic of research.

One classical topic in RE research and practice is that of requirements traceability (RT). It is defined as "the ability to describe and follow the life of a requirement" [4]. RT practices are mandated by commonly accepted standards such as CMM, ISO 9000, and IEEE Std. 830-1998 [5], [6]. As a consequence, organizations who wish to or are required to comply with such standards embrace RT practices.

A reason why RT practices are included in a plethora of standards is because adopting them is expected to deliver several benefits during a software project [7]. The first benefit is in the area of project management. During a project, requirements are bound to change. However, before changes can be implemented the impact of those changes need to be assessed. By adopting RT, the change affected requirement artefacts can be identified.

Second, RT also benefits the project visibility [7], [8]. Trace information can be shared with the whole project team. Utilising this, all team members have access to finer context and rationale behind the requirement. A lower-level requirement might be puzzling for an engineer but reading the higher-level requirement may provide the necessary context. In addition, the increased project visibility benefits in the onboarding of new team members.

Third, RT practices provide benefits during the maintenance phase [7]. Requirements often change during a project. Implementing change requests has its impact on multiple other requirements, code, and test cases. Using RT, it is easier to identify which artefacts need to be updated and thus provides insight in the impact of the change request. This insight makes maintenance tasks more efficiently and increase the quality of work [9].

Finally, and most significantly, the benefits of RT are realized during the phase of verification and validation [7]. An increase in the level of traceability decreases the expected defect rate in developed software [10]. This leads towards an increased implementation quality.

These benefits make RT a topic of interest in multiple areas of software engineering research [11]. One of those areas is model-driven development (MDD). MDD "is a development paradigm that uses models as the primary artefact of the development process" [12]. It aims to raise the level of abstraction during development [13]. By doing so, MDD makes software artefacts more accessible to a wider range of people. In an ideal MDD implementation, the necessity for highly skilled software developers will be reserved for complicated projects, while daily projects may be developed by the organization's existing personnel, resulting in increased productivity.

1.2 Problem Statement

MDD offers many opportunities for practicing traceability [11]. Unfortunately, the practice of RT is not self-evident. RT activities are found to be "time-consuming, tedious and fallible" [14]. The lack of awareness is one of the reasons why RT activities are poorly used or not adopted altogether [5], [15]. Other reasons include financial, political, customer, and operational factors.

However, even when organizations do see the benefits of RT, many industry practitioners prefer to use manual traceability techniques over traceability tools [7]. This is a problem because manual tracing methods are error-prone, vulnerable to changes, time-consuming, and impossible to maintain [16]. In short, these manual tracing methods do not suit the needs of the software engineering industry.

So why are these unsuitable manual methods still favoured over traceability tools? Gotel and Finkelstein [4], found that this preference for manual tracing methods was due to the shortcomings of available traceability tools. Kannenberg and Saiedian [7] concluded this was still apparent, since the tool penetration was still low.

Therefore, there is a need for enhanced traceability tools, that address the shortcoming of the current traceability tools. Recent advances in the machine learning domain provide opportunities, that may be able to address these limitations [14], [17]. These technologies should aim to facilitate the recovery of trace artefacts commonly used in modern software development [17].

1.3 Research Objective and Questions

In this research we aim to design a new RT tool, which incorporates technologies from the machine learning domain, in order to improve the aforementioned problems. Specifically, the research will focus on the MDD domain. MDD is predominately focussed on tracing models to models. However, traceability in MDD needs to consider how models can be traced to non-model (e.g., requirements) artefacts [18].

To study this, we collaborate with Mendix, a MDD-platform producer [19]. An exploratory research on RT was already done at Mendix and opportunities for a new RT tool were identified [20]. This research will build on top of this earlier acquired knowledge, to design the new RT tool. The new tool will be investigated and constructed iteratively. We hope to bridge the gap between industry and research by forming this cooperation. To clarify the objective of the research, the template of Wieringa [21] is utilised to come up with the following research objective:

> The goal of the research is to **improve** the perceived usefulness of RT tools, **by** automating the recovery of requirement traces in an MDD-context, **that** incorporates techniques from the machine learning domain, **so that** the MDD industry can better cope with changing requirements.

To achieve the research objective, the research is guided by the main research question (MQ).

MQ	How to automate tracing between requirements and models in a Model Driven Environment?

This MQ is decomposed into five sub-research questions (SQs). They assist in answering the MQ. The first two SQs aim to build a thorough understanding of the relevant literature.

SQ1	What is the state-of-the-art in Requirements Traceability?

SQ1 builds a thorough understanding of the literature on RT. We aim to discuss the fundamentals of RT and establish a theoretical framework. This is achieved by means of a semi-structured literature review.

SQ2	What algorithms are needed to automatically trace artefacts?

Once the fundamentals on RT are established, the research can be scoped to finding the state-of-the-art on automatically tracing artefacts. Specifically, SQ2 aims to identify current approaches and compiles a list of the existing algorithms that these approaches employ, which may be employed in our treatment design.

SQ3	How do MDD artefacts and requirements co-evolve in an MDD company?

After understanding the theory, knowledge of the domain needs to be built. This will be done by answering SQ3. The RT tool is supposed to interact and operate in the MDD-domain, rather than in an isolated setting. To optimally design for this scenario, the MDD-domain needs to be understood. The answering of SQ3 is operationalized by means of two activities. The first activity is to review the literature on the fundamentals of MDD. Parallel to this activity, exploratory interviews are held at Mendix. Goal of these is to find out the needs and opportunities.

Once knowledge about both the theory and practice is known, the next phase can be initiated. In this phase, the design and development of the RT tool take a central stage. The activities in this phase serve to answer the following SQs.

SQ4	What are the resources available to design and construct a RT tool for the MDD domain?

Through the collaboration with Mendix, we gain access to datasets on both requirements and MDD artefacts. SQ4 aims to create an overview of these resources and how these can be utilised for the design and construction of the RT tool.

SQ5	How to embed automatic tracing algorithms in a RT tool for the MDD domain?

An overview of trace recovery approaches has been created. SQ5 aims to find a way how (parts) of these approaches can be embedded in the design of the RT tool. Ultimately, it should produce a prototype which is able to take 1) requirements and 2) models as input and output the trace links between them.

SQ6	How do we validate the effectiveness of a RT tool for the MDD domain?

Finally, the produced prototype needs to be validated. SQ6 aims to find out how the effectiveness can be measured and thereafter report the results of the prototype.

1.4 Thesis Outline

The thesis report is structured as follows: Chapter 2 will discuss the Design Science method that we used for this research. It goes over the three phases of Design Science: Problem Investigation, Treatment Design, and Treatment Validation, as well as how we have used them in our study. Then, in Chapter 3, key terminology on RT and MDD will be presented. In addition, it will also discuss the state-of-the-art in trace link recovery. This concludes the literature review, which will be followed by the results of the Design Science phases.

Chapter 4 will through the results of the Problem Investigation phase. It explains the context in which the to-be-created artefact must function. In addition, two scenarios are described, which the artefact should support. Then, Chapter 5 describes the outcomes of the Treatment Design phase. It addresses the design of the artefact that we developed in order to support the scenarios described in Chapter 4. In here, we will go through all the components that make up the treatment artefact. The experiments of determining the most optimal configuration of these components are described in Chapter 6. It provides the results of the best configuration for each of the two scenarios. This is followed by Chapter 7, which discusses the key contributions of our study, as well as any threats to the validity and how we attempted to mitigate them. Finally, the study is concluded in Chapter 8, which additionally provides directions for future research.

2. Research Method

2.1 Design Science

We stated in our research objective that we wanted to automate "the recovery of requirement traces in an MDD-context". To accomplish so, we will need to design an artefact that is capable of doing so. This brings us to the field of Design Science, which is concerned with the creation and investigation of artefacts in context [21]. An important component of Design Science is the Design Cycle. It is a rational problem-solving framework specifically developed for software engineering and information systems research. It is part of the Engineering Cycle and consists of three activities that are iterated over Problem Investigation, Treatment Design, and Treatment Validation. Figure 1 illustrates the relationship between the Engineering cycle and the Design cycle, together with its phases. These phases were well suited to the requirements of our research and were therefore employed to structure it.

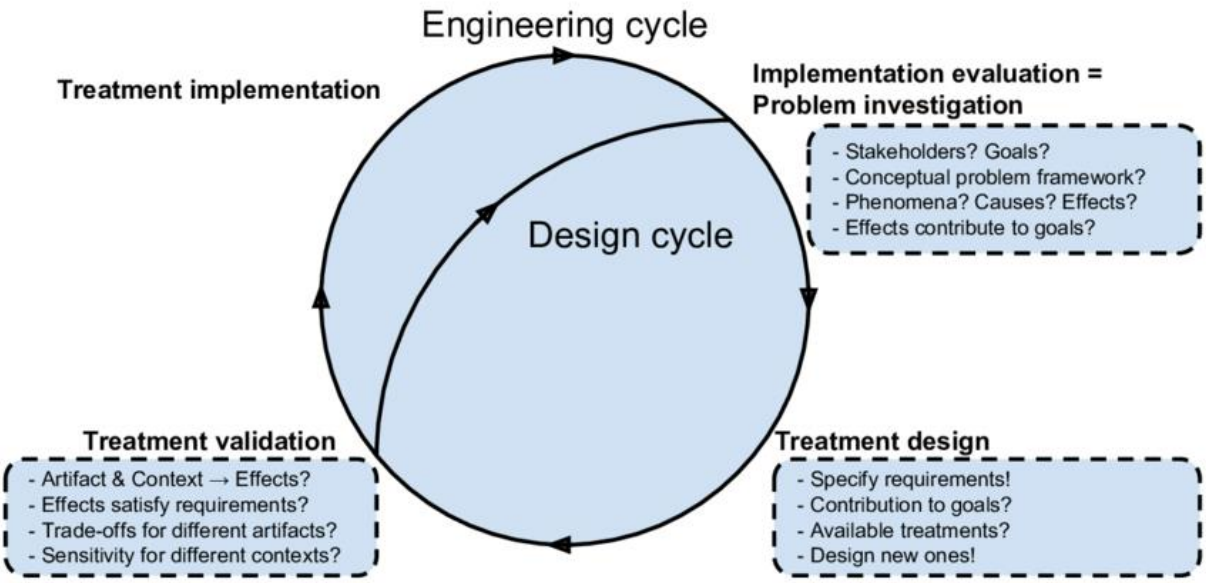


Figure 1: Design Cycle [21]

Problem Investigation
The first task in the cycle is the problem investigation. The goal of this task is to understand the problem. We have operationalized this task by first doing a semi-systematic literature review, further elaborated in Section 3.1. Its goal was to identify what the current problems are in RT research, together with their proposed solution, answering SQ1 and SQ2. Simultaneously, we conducted semi-structured interviews with stakeholders of a case company, further elaborated in Section 3.2. The goal of this was to provide an answer to SQ3 by identifying the problems occurring in industry.

Following the semi-systematic literature study and semi-structured interview, we attempted to determine which problem was present in both the literature and the case company. This problem was then selected as the problem for which we intended to find a solution.

Treatment Design
This solution was developed during the Treatment Design phase. We took all of the knowledge obtained during the Problem research phase and combined it with resources (e.g., data, expertise) from Mendix to build a treatment for the problem. This phase generated the information needed to answer SQ4 and SQ5, which was subsequently materialized as a software artefact.

Treatment Validation
Finally, the developed software artefact was evaluated to determine whether the intended treatment produced the desired results. This was accomplished by conducting experiments and quantifying performance using metrics found during the Problem Investigation Phase. This phase provided us with the answers required for solving SQ6.

2.2 Semi-Systematic Literature Review

To research the state-of-the-art of requirements traceability (SQ1) and the state-of-the-practice of model-driven development (SQ2), relevant literature was reviewed. The selection of literature was done according to the guidelines of a systematic literature review (SLR), proposed by Kitchenham et al. [22]. It needs to be taken into account that the primary goal of the SLR was to gain a general understanding of the problem, with the aim of designing a treatment, rather than an exhaustive mapping of literature. Therefore a selection and adaptation of components of a SLR was made [22]. This selection was then used as a guideline. The selection includes seven tasks, which are specified in Table 1.

Table 1: Overview of all tasks executed during the systematic literature review

Task	Description
Specify the research questions	The SLR seeks to provide an answer to SQ1, SQ2, and SQ3 as defined earlier: **SQ1** What is the state-of-the-art in Requirements Traceability? **SQ2** What algorithms are needed to automatically trace requirements to MDD artefacts? **SQ3** How do MDD artefacts and requirements co-evolve in an MDD company?
Define Search Terms	The following search terms are considered: requirements traceability, technique, tool, automated, automating, software traceability, traceability, model driven engineering, model driven development, model driven architecture, MDE, MDD, and MDA.
Define Search Queries	• (Requirements Traceability OR 'Software Traceability') • ('Requirements Traceability') AND ('Technique' OR 'Tool') • ('Automated' OR 'Automating') AND ('Requirements Traceability' OR 'Traceability') • ('Model Driven Engineering' OR 'Model Driven Development' OR 'Model Driven Architecture' OR 'MDE' OR 'MDD' OR 'MDA')
Select Sources	All search queries are done on Google Scholar.
Query Sources	The defined search queries were applied on Google Scholar.

Apply Inclusion Criteria	Literature was included if one of the following criteria was met: • The article has one of the terms or synonyms as a keyword. • The focus of the article is on requirements traceability. • The focus of the article is on model-driven development.
Apply Exclusion Criteria	Literature is excluded if one of the following criteria was met: • Literature not available in English. • Literature not available in Dutch. • Literature only available in the form of an abstract.

2.3 Backward and Forward Snowballing

The semi-systematic literature study provides the initial set of literature. This literature was used as a starting point for the snowballing procedure. Snowballing is the systematic search for "primary studies based on references to and from other studies" [23]. The procedure differentiates 2 types: backwards snowballing and forward snowballing. The former refers to looking at the papers cited by the 'starting article' and the latter refers to looking at papers citing the 'starting article'. For this research at most two rounds of both backwards and forward snowballing was used. For the articles to be considered the inclusion and exclusion criteria defined in Table 1 are used.

2.4 Case Study

To come up with a more elaborate answer to SQ2 and to answer SQ3, interviews were held with different experts from Mendix. Founded in 2005, it is now the leading low-code software development platform [24]. The organisation has over 1500 employees worldwide and has over 4000 companies using their platform.

The interviews followed the guidelines for semi-structured interviewing by Longhurst [25]. First, questions were formulated around 3 themes: the Mendix Platform, requirements, and tracing requirements to MDD artefacts. These can be found in the interview protocol, found in Appendix A. This protocol was then shown to a contact person at Mendix, who provided interviewees to perform the interviews with. These were held with four experts within Mendix, shown in Table 2, either in English or Dutch, and were audio recorded. Within one week, the recordings were played back and transcribed.

Table 2: Overview of all Mendix employees interviewed

Interviewee ID	Interviewee Function	Goal of Interview
1	Principle Engineer	SQ2
2	Principle Engineer	SQ3, SQ4
3	Solution Architect	SQ3
4	Mendix Developer	SQ3, SQ4

3. Literature Review

3.1 Fundamentals of Requirements Traceability

In 1968 the NATO Science Committee organised a conference on software engineering. Its goal was to identify current problems in software engineering, and discuss possible techniques, methods, and developments which might solve those problems. During that conference the importance of traceability was already recognised [26].

Gotel et al. [27] present a terminology on traceability. They define **traceability** as the potential for traces to be established and used. A **trace** is comprised of two elements, displayed in Figure 2: **trace artefact** and t**race link**.

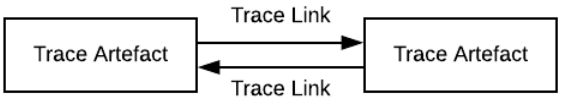


Figure 2: Two Trace Artefacts Connected Through a Trace Link Forming a Trace

The Trace Artefact is a traceable unit of data. An example of a trace artefact is a single requirement or a Python class. The Trace Link is a specified relation that is used to interrelate a pair of trace artefacts. Trace Artefacts can be differentiated into categories with the same or similar structure and/or purpose, called **trace artefact types**. For instance, requirements may be a distinct artefact type.

Depending on which trace artefact type is the object of interest, several types of traceability can be delineated. For example in test-to-code traceability, one is exclusively interest in tracing unit tests and tested classes [28].

Research on traceability has greatly focused on requirements traceability [29]. Since its inception it has been an important topic in the requirements engineering research community. In 2005, the Center of Excellence for Software & Systems Traceability (CoEST) was founded to encourage and foster RT research [30]. Researchers and practitioners of the CoEST envision traceability to be ubiquitous in software and system development [31]. Once the vision of ubiquitous traceability is fulfilled the following scenario described by Cleland-Huang et al. [32] should be commonplace:

> *"A new developer joins an agile team and is assigned a user story to implement. She uses automatically captured trace information to explore the impact of the new story on the system. Results are quickly visualized in ways that help her to understand which parts of the codebase might need to be changed, potential side effects on existing user stories and test cases, and a list of fellow team members who have previously worked with the code and could be considered expert consultants".*

For us to study RT, we need to establish a definition which explains the concepts. Several definitions have been proposed by multiple authors. For instance Pinheiro [33] defines RT as:

> **Requirements Traceability** "*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".*

Another earlier definition was given by Gotel & Finkelstein [4]:

> **Requirements Traceability** *"refers to the ability to describe and follow the life of a requirement, in both a forwards and backwards direction (i.e., from its origins, through its development and specification, to its subsequent deployment and use, and through all periods of on-going refinement and iteration in any of these phases)".*

The definition coined by Gotel & Finkelstein became the most prominent definition for RT and is consequently used by several other studies [11], [34], [35]. For the same reason, this work will utilise the definition by Gotel & Finkelstein.

3.1.1 Requirement Traceability Delineations

In the literature there are several common delineations made for tracing requirements.

Backward and Forward Traceability

Traceability can be delineated in the direction they are able to trace. This can be in either a forward or a backward direction [27], illustrated in Figure 3, and is defined as follows [33]:

> **Backward traceability** *"The ability to trace a requirement to its source, i.e., to a person, institution, law, argument etc."*

> **Forward traceability** "*The ability to trace a requirement to components of a design or implementation".*

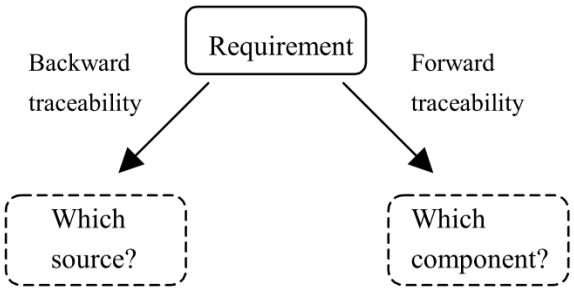


Figure 3: Backward and Forward Traceability [33]

Pre- and Post-Requirements Specification Traceability

The results of an elicitation process are collected in a requirements specification (RS). Requirements in the RS can be traced into two directions: 1) To information prior to its inclusion in the RS or 2) to information after its inclusion in the RS. This distinction is illustrated in Figure 4 and is defined as [33]:

> **Pre-RS Traceability:** "*refers to those aspects of a requirement's life prior to its inclusion in the requirements specification".*

> **Post-RS Traceability:** *"refers to those aspects of a requirement's life that result from inclusion in the requirement specification".*

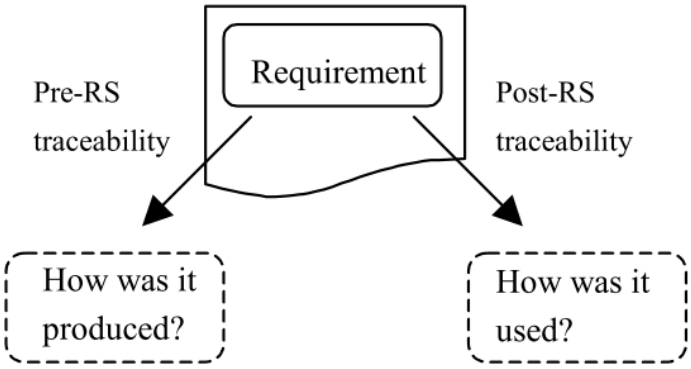


Figure 4: Pre-RS and post-RS traceability [33]

Manual and Computed Traceability

Trace links can originate in either two ways: manual and computed and is defined as [35]:

> **Manual Traceability** "*Trace links are established by a human user".*

> **Computed Traceability** *"Trace links are established by an algorithm".*

Computed traceability is established by means of automated reasoning. Examples of this include information retrieval or machine learning algorithms. More on this will be discussed in 3.3.

3.2 Fundamentals Model-Driven Development

Within academia many acronyms are used in the Model-Driven Paradigm. Therefore, we first need to establish a common understanding in the different used terms.

Model-Driven Development (MDD) "is a development paradigm that uses models as the primary artefact of the development process" [12]. The Object Management Group (OMG) adjusted this definition to fit other OMG standards. This particular vision is called Model-Driven Architecture (MDA), which "provides guidelines for structuring software specifications that are expressed as models" [36]. For this reason, MDA can be seen as a subset of MDD.

On the other hand, MDD can be regarded as a subset of Model-Driven Engineering (MDE). The former focuses purely on the development activities, whereas the latter encompasses all the model based tasks in a software engineering process [12]. The relationship between MDE, MDD, and MDD is summarized in Figure 5.

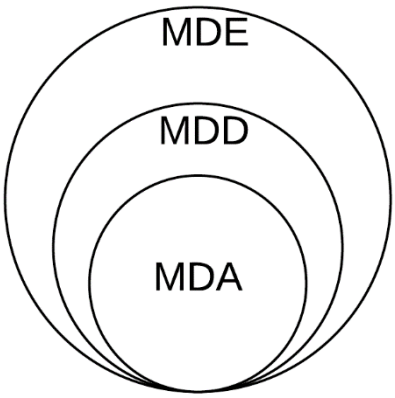


Figure 5: Relationships between MDE, MDD, and MDA

Model-Driven Engineering consist of two key components: models, and model transformations.

Models

Models can be defined by a conceptual and a technical definition. This distinction is important according to Holtmann et al. [35]. They argue that 1) both definitions do not always correspond to each other, and 2) an unambiguous understanding of model boundaries (i.e., whether trace artefacts belong to the same model) is needed.

To effectively communicate our ideas, we must first settle on the precise definition to which we will adhere. Academia offers a number of conceptual definitions of a model. For example, the following definition was provided by Wasowski and Berger [35].

> **Model** *"an abstraction of reality made with a given purpose in mind".*

Although it explains the concept, it does not help to differentiate whether two artefacts are part of the same model or part of two distinct models. Therefore, a more technical definition is needed. Holtmann et al. [35] coined the following definition:

> **Model** *represents an aspect of a system under development captured in a specific instance of a formal language that serves a purpose within the development lifecycle.*

The definition implies, that all artefacts expressed in instances of formal languages are considered models. For instance, when a requirement documented in natural language is structured by a meta-model for requirements, it is considered a model. In addition, because it defines a system rather than a mental model, this definition is more appropriate for our goal.

Model Transformations
The main task of MDD is to transform higher-level models into platform-specific models [37]. This is done by applying transformations rules which dictate how the source-model should be transferred to the target-model. This automated process is referred to as model transformation.

3.3 Computer-Assisted Trace Link Recovery

The software engineering task focused on establishing trace links between related artefacts is called Traceability Link Recovery (TLR) [38]. The process exclusively focusing on establishing trace links between requirements and other artefacts is requirements traceability recovery (RTR) [39]. According to Aung et al. [17], the approaches for TLR and RTR can be separated into four orthogonal categories: Information Retrieval based, Heuristic-Based, Machine Learning, and Deep Learning.

This section will first discuss the measures that are commonly used to compare the performance of the TLR and RTR methods. Then it will discuss the fundamentals of each category defined by Aung [17]. Following that, a discussion of some notable TLR approaches will take place. Finally, we will use that discussion to draw lessons for the design of our treatment.

3.3.1 Metrics

Many different computed TLR and RTR approaches have been proposed. To compare and benchmark these approaches it is necessary to define evaluation methods and metrics. Shin et al. [40] did a systematic literature on current evaluation practices on requirements traceability techniques. They found that traceability is typically measured using either *classification accuracy metrics* or *rank accuracy measures.* In Figure 6 the occurrences of the different metrics are given. We will now discuss the top 5 most occurring metrics.

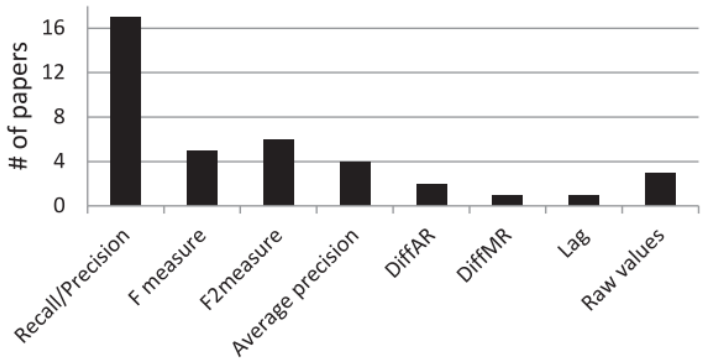


Figure 6: Overview of the eight most popular metrics for benchmarking computer-assisted TLR [40]

Classification Accuracy metrics

This set of metrics count the number of correctly or incorrectly retrieved links. Commonly used classification accuracy metrics include *recall*, *precision*, and *F-Measure*. Recall denotes the fraction of relevant documents that are correctly retrieved and is defined as follows:

Equation 1: Recall

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

Precision denotes the percentage of retrieved trace links which are valid and is defined as follows:

Equation 2: Precision

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

Recall and Precision are always given together. The reason is that between both, there is a trade-off. As one increases the other decreases. The average between both is given by the F-measure, which knows two variants: *F_1-Measure* and *F_β-Measure*. F_1-Measure denotes the harmonic mean of precision and recall. It is denoted as follows:

Equation 3: F_1-Measure

$$F_1 Measure = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

F_β-Measure is a simplified version of F_1-Measure. It applies a real weighting factor β, valuing either precision or recall more than the other. In case β > 1, more emphasis is put on the importance of the recall. F_β-Measure is defined as follows:

Equation 4: F_β-Measure

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

Rank Accuracy Measures

The accuracy of the relative ordering of correct links in a ranked list are measured using rank accuracy metrics. These include: Average Precision, DiffAR, DiffMR, and Lag.

The Average Precision denotes the extent to which relevant links are placed towards the top of a ranked list. It is defined as follows:

Equation 5: Average Precision

$$Average\ Precision = \frac{\sum_{r=1}^{N}(P(r) * isRelevant(r))}{True\ Positives + False\ Negatives}$$

In the equation, r is the rank of a document in the ordered list of retrieved results from N documents. isRelevant() is a binary function, which assigns 1 if the rank is relevant and 0 if otherwise. P(r) denotes the precision, computed after truncating the list immediately below that ranked position.

DiffAR is a measurement for the difference between the average relevance scores of correct and incorrect trace links retrieved. It is defined as follows:

Equation 6: DiffAR

$$\text{DiffAR} = \frac{\sum_{(q,d) \in \text{true positives}} rel(q,d)}{True\ Positives} - \frac{\sum_{(q,d) \in \text{false positives}} rel(q,d)}{False\ Positives}$$

In the equation, q denotes a query, d denotes a document, and rel(q,d) denotes the relevance score between q and d.

3.3.2 Information Retrieval

The objective of an information retrieval (IR) system is to find relevant information from an organised collection of documents [41]. Today, these IR systems are omnipresent: when searching the recipe for apple pie on Google; when searching a movie on Netflix; or searching a product on Amazon. These systems rely on IR algorithms, which need to present the user only those documents which are semantically similar to the search query.

In a similar way, these IR algorithms are used by most modern semi- or automatic TLR approaches [6], [42]–[44]. These techniques work on the premise that when two artefacts have high textual similarity, they probably should be traced to each other [45]. Therefore, to recover trace links, the algorithm computes the textual similarity between software artefacts and assigns a score to them. Trace links scoring above a given threshold are considered a valid link. Common used IR algorithms include Vector Space Models, Latent Semantic Indexing, Jenson and Shannon Models, Latent Dirichlet Allocation [17]. Although good results have been achieved, recovering trace links reliably may be difficult. Because the algorithms heavily depend on the text quality, considerate pre-processing is required.

Vector Space Models (VSM)

In VSM, the goal is to represent each document d as a vector in a vector space [46]. This is illustrated in Figure 7. Vectors close to each other are semantically similar, whereas vectors distant from each other are semantically different. When a query q is ran, a point in space is identified, and documents close to the point are returned. This is operationalised as follows: 1) all unique terms occurring in query q and document d is represented as a vector T = $\{t_1 \ldots t_n\}$. 2) A weighting scheme is chosen, of which TF-IDF is a common one. TF-IDF is the product of two statistics: *term frequency* and *inverse document frequency* [47]. Term frequency refers to the number of times t occurs in d. The inverse document frequency refers to the rarity of a term t across all documents. It is defined as follows:

$$\boldsymbol{idf_t = log \frac{N}{df_t}}$$

Where N is the total number of documents and df_t the number of documents t appears in. 3) All documents are represented as vectors d = $\{w_{1,d}, \ldots, w_{nd}\}$ and the TF-IDF of term i in document d is calculated. 4) Finally, the similarity is calculated as the cosine of the angle between query q and document d as follows:

$$sim(Q, D) = \frac{\sum_{i=1}^{n} q_i \times d_i}{\sqrt{\sum_{i=1}^{n} q_i^2 \times \sum_{i=i}^{n} d_i^2}}$$

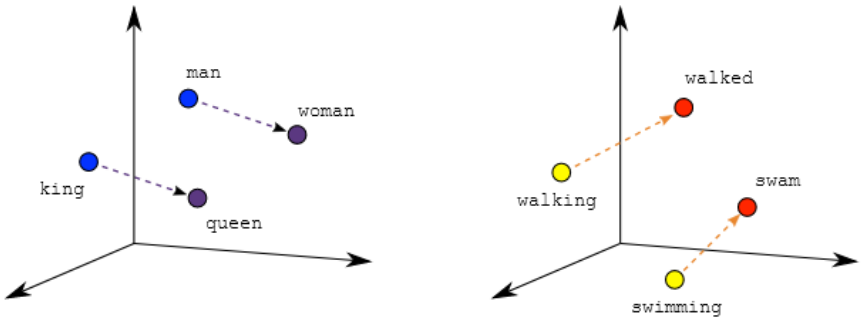


Figure 7: Graphical representation of spatial distance between terms [48]

Latent Semantic Indexing

A fundamental problem in IR is that searchers employ query terms that are not the same as the ones used to index the documents they seek. This issue can broadly be traced back to the notion of synonymy and polysemy. Synonymy refers to the problem that people use different terms for the same object (e.g., 'drawing' and 'illustration'), and decreases the recall. Polysemy refers to the problem that terms have multiple definition (e.g. 'fall'), and decreases precision [49].

The problem can be illustrated as follows: We have a set of documents. 100 documents contain the word 'drawing', and 100 documents contain the word 'illustration'. 95 documents contain both the words "drawing" and 'illustration'. When running a query 'drawing', we want to retrieve all documents containing the word 'drawing'. However, we also want to retrieve documents only containing the word 'illustration', because there likely related to each other.

The example illustrates that the occurrence of the terms 'drawing' or 'illustration' on their own are bad indicators of a relevant document. We need to sort out how to predict what terms in a query 'actually' mean (i.e., 'latent semantics') and replace them by an implicit higher-order structure, which is a more reliable indicator. Deerwester et al. [49] introduced an Latent Semantic Indexing (LSI), an algorithm which is able to recover these 'latent semantics'. It utilises a technique called Singular Value Decomposition, to approximate the original set of terms by a 'latent' structure. This latent structure can then be used a feature set, which better reflects major sociative data patterns and ignores less important influences.

Jenson and Shannon Models

In Jenson and Shannon Models (JSM) documents are treated as a probabilistic distribution [50], [51]. The probability of its states is given by the empirical distribution of the terms occurring in the document. Like VSM, the similarity between the query q and the document d is measured by the 'distance' between both. However, while VSM measures the distance using the cosine of the angle between q and d, JSM measures the distance using the Jenson-Shannon Divergence. It is defined as follows:

$$JS(q,d) \triangleq \left(\frac{\hat{p}_q + \hat{p}_d}{2}\right) - \left(\frac{H(\hat{p}_q) + H(\hat{p}_d)}{2}\right)$$

$$\mathrm{H(p)} \triangleq \sum_{w \in W} h(p(w)), h(x) \triangleq \text{-x log x},$$

where H(p) is the entropy (a measure of uncertainty) of the empirical distribution, $\hat{p}_q$is the empirical distribution of q and $\hat{p}_d$ the distribution of d.

Latent Dirichlet Allocation
Latent Dirichlet Allocation (LDA) is a probabilistic model [52]. It takes a m × n *term-by-document* matrix as input, where m is the number of terms occurring in the entire corpus, and n is the number of documents in the corpus. LDA is then able to identify the latent topics and generates a k × n *topic-by-document* matrix, where k is the number of latent topics. Using this reduced space, LDA clusters documents with the same relevant topics.

3.3.3 Heuristics
Many of the IR techniques still require intervention of the user, which make them only semi-automatic [39]. Another attempt for automatic RTR is Heuristic Search (HS). HS aims to find an acceptable approximate solution to a specific problem in a search space [53]. This technique is usually utilised whenever an exact algorithmic solution is absent or is too complex and time-consuming. Examples such as hill climbing, simulated annealing, and genetic algorithms belong the paradigm of HS techniques.

3.3.4 Machine Learning
Machine Learning (ML) is defined as "the automated detection of meaningful patterns in data" [54]. A well-known application of ML is the filtering of spam e-mails. To do this, the machine is given a set of emails which are labelled 'not spam' or 'spam'. It then runs an algorithm over the data and it 'learns' which features make up a spam e-mail. This process results in a model, which takes an e-mail as input and outputs whether the e-mail is spam or not.

In recent years, developments from the ML domain have been utilized in automatic TLR [17]. ML approaches treat the TLR process as a classification task [38], [55]. Given two artefacts it needs to label the link between them as valid or invalid. In case of the former, there is a trace link. For this to work, the ML classifiers need to be trained on data. This training data arises as follows: Given two artefact sets A_1 and A_2, the Cartesian product $A_1 \times A_2$ is computed. Each element of $A_1 \times A_2$ represents a trace link between $a \in A_1$ and $a \in A_2$. These are either valid or invalid, which is the label the classifier needs to learn. Therefore, for each trace link a vector representation is computed, derived from features. Most ML RLT approaches use similarity scores of IR-based methods as features [38], [55], [56] and are able to outperform IR-based TLR approaches [38].

Data imbalance
When computing $A_1 \times A_2$, it is expected that most trace links are invalid. Therefore, the training data is highly imbalanced, which makes the training of a classifier problematic [57]. For example, a training set could consist of 100 trace links, of which only 3 trace links are valid. A classifier achieves the greatest performance, when it classifies 100 percent of the trace links as invalid. While 97 trace links are correctly classified, it results in the misclassification of the trace link of interest.

It is therefore important to rebalance the data. One possibility is to use undersampling. Undersampling is data-reduction method that reduces the majority class by selecting only a subset of its datapoint for training. Another possibility is to use oversampling. This technique artificially creates new data points of the minority class, based on the original data.

3.3.5 Deep Learning

Deep learning (DL) finds its origin in Artificial Neural Networks (ANN). ANNs approximate the human brain by connecting many simple computational units, called neurons, in a multi-layered structure. Several neural network structures exist, each targeted at a specific learning task. For example, convolution neural networks are well suited for image recognition tasks, while recurrent neural networks (RNN) are targeted tasks concerned with sequential inputs, such as NLP.

Because TLR deals with natural language, RNNs are mostly used for this task. Figure 8 shows the typical architecture RNNs follow. RNNs allows outputs to be used as inputs, and essentially mimic memory [58], [59]. This is operationalised by the activation function $a^{<t>}$, which takes two inputs: 1) the current time step t (i.e., word embedding), and 2) the output of the previous activation function $a^{<t-1>}$. The output of the RNN is therefore determined by its current and prior input.

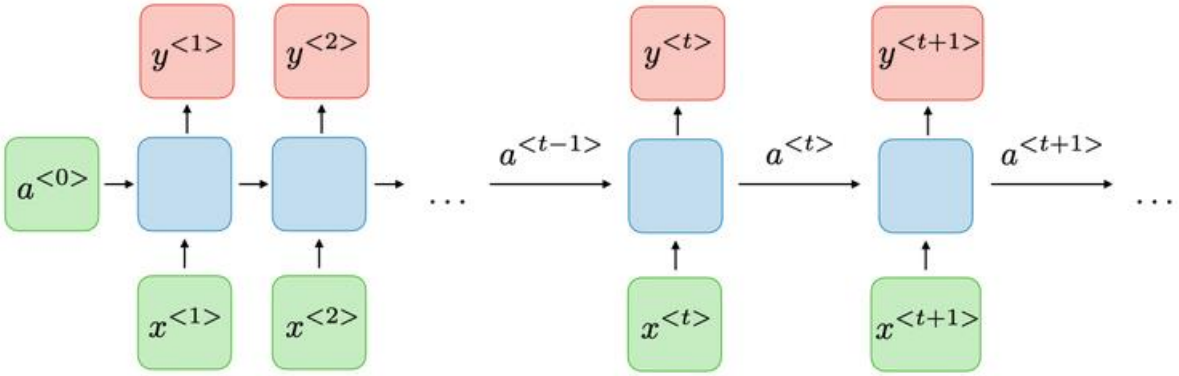


Figure 8: Architecture of a recurrent neural network [59]

Common activation functions used in RNNs include the Sigmoid, Tanh, and Rectified Linear Unit (RELU) functions. These are described in Figure 9.

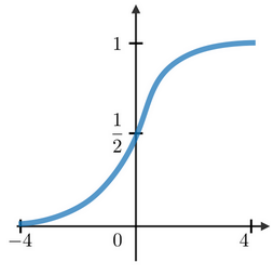
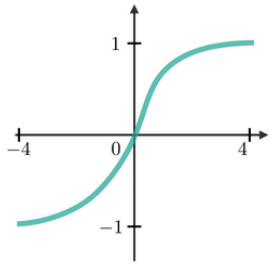
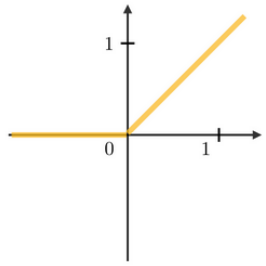
Sigmoid	Tanh	RELU
$g(z) = \frac{1}{1+e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0, z)$
		

Figure 9: The formulas and plots of Sigmoid, Tanh, and RELU functions

3.3.6 Automated Traceability Approaches

Aung et al. [17] did a systematic literature review on automated TLR approaches and identified 33 relevant studies published between 2012 and 2019. These studies were mapped into categories. In this section, we will discuss the studies mapped to either 'Machine Learning', 'Information Retrieval + Machine Learning', 'Deep Learning', or 'Information Retrieval + Deep Learning'. These mappings were chosen because literature agrees that both ML-based and DL-based TLR approaches outperform IR-based TLR approaches [14], [58], [60].

Tracing Features to Code Commits Using Machine Learning

Abukwaik et al. [61] proposed a recommender system for annotating features to code commits. Whenever a developer commits code to a version control system, the system must recommend possible features the code commit may belongs to.

To create the system a Java tool was constructed, which reuses IR libraries from the Lucene search engine and ML libraries from the WEKA toolkit. The tool does 3 tasks: data pre-processing, generating ML classification models and creating evaluation results. Figure 10 shows all individual steps and are elaborated below.

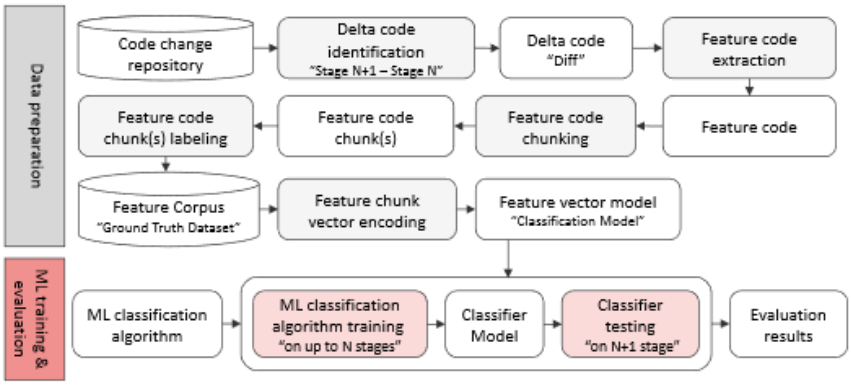


Figure 10: Overview of the ML simulation experiments conducted by Abukwaik et al. [61]

Data Pre-processing

The approach starts with preparing the data. Each delta code is extracted from the repository. Thereafter, code related to a feature (e.g. `//&line[System Monitor]`) is extracted from the delta code, which is then associated with the respective feature (System Monitor). Finally, each delta is chunked to the granularity of a single line of code and is labelled with the name of their related features. These chunks are saved in the Feature Corpus.

Generating ML Classification Models

For every chunk in the Feature Corpus, a data vector D is calculated, which consists of four metrics, explained in Table 3. These were then used to train a kNN, SVM, and Decision Tree classifying model.

Creating Evaluation Results

The input of a classifier is a new set of data vectors for an unclassified code block. The output is an indication of which features it may belong to. For every stage the N in the code repository the experiment is run, and the performance is recorded.

Table 3: Overview of feature families used by Abukwaik [61]

Classification Feature	Description
Feature Presence Metric ($\vec{f}$)	Indicates if the code block belongs to a feature, indicated by a 1 or 0.
Cosine Text Similarity($\vec{c}$)	Represents the similarity between two vectors, measured as the cosine of the angle between them.
Source Code Localisation Distance($\vec{s}$)	The relative distance of the code block to already known feature locations
Number of Already Existing Annotations metric($\vec{n}$)	The summation of the number of existing annotations for a feature

Results
The best results were achieved when applying kNN, with a F1-measure between 50% and 60%, which was already achieved after training on 60 commit change-sets, with an average of 5 annotations each. The authors did not mention the number of k used when training on the data. Results for SVM were unsatisfying with a F1-Measure of 7% and 15%.

Maintaining Traceability Information using Machine Learning

Mills et al. [38] proposes an approach, called TRAIL (TRAceability lInk cLassifier), with the goal of automatically verifying the validity of ranked trace links by IR models, using ML classifiers. Their method starts with typical data pre-processing tasks [43], which consists of four steps: 1) all identifiers were split using camelCase and under_score algorithms, 2) common English, Italian, and Java keywords were removed, 3) the remaining keywords were stemmed to their root form, 4) the approach rebalanced the training data. Both undersampling as oversampling was conducted, for which they used Synthetic Minority Oversampling Technique (SMOTE) and Random Majority Undersampling.

Once the data was prepared, the cartesian product of both artefact sets was computed, of which each element is a vector. These vectors consist of 131 features, which fall into three distinct categories. These categories are elaborated on in Table 4.

To decrease the dimensionality of the feature space, five feature selection algorithms are considered: Correlation-based Feature Subset Selection, Pearson's Correlation, Gain Ratio, Information Gain, and Symmetrical Uncertainty. Finally, each potential trace link is classified by a set of ML algorithms, which include k-Nearest Neighbours with k = 5 (5NN), Naive Bayes (NB), Logistical Regression, Random Forest (RF), Support Vector Machines (SVM), and a Voting ensemble.

Results
The results of TRAIL are displayed in
Table 5. The configuration of Random Forest as classifier, and Pearson correlation for feature selection and SMOTE for data rebalancing, results in the best performance with an average F-score of 75.18%.

Table 4: Overview of feature families used in TRAIL [38]

Feature	Description
IR-Based	For every potential trace link, the similarity score is calculated using 7 different IR-algorithms. Since the retrieval direction significantly impacts the results of TLR [62], the similarity score is calculated for each direction for each IR-algorithm. This results in a total of 14 IR-based features.
Query Quality	It matters if a candidate link has a low similarity score, because they are indeed invalid, or that the artefact is generally hard to trace. This is quantified in 28 Quality Quantity (QQ) metrics [63]. These QQs can be divided into 21 pre-retrieval QQs and 7 post-retrieval QQs, for which the former is applied before running a query and the latter after running the query.
Document Statistics Features	For each document the three statistics are calculated: a) number of unique terms, total number of terms, c) percentage of overlapping terms between two documents in a candidate link. These statistics are then used to calculate five classifications in the following way: Given artefact A_1 and A_2. For both A_1 and A_2, statistic a and b are calculated. Then for the trace link between A_1 and A_2 feature c is calculated.

Table 5: Average F-score (in percentage) achieved by TRAIL [38]

Rebalancing Technique	Feature Selection	Classifier					
		5NN	Logistic Regression	NB	RF	SVM	Vote
none	None	47.43	50.19	39.49	67.18	0.00	55.96
	cfs	59.72	40.25	39.22	63.14	0.79	53.84
	correlation	47.43	50.19	39.49	67.22	0.00	56.06
	GainRatio	61.22	61.00	40.17	72.03	0.00	66.84
	InfoGain	61.22	61.00	40.17	72.29	0.00	66.96
	Symmetrical	61.22	61.00	40.17	72.07	0.00	66.89
undersampling	None	31.18	34.60	36.13	51.37	31.22	38.24
	cfs	39.88	37.65	35.65	47.43	34.77	38.35
	correlation	31.18	34.59	36.13	51.42	31.41	38.28
	GainRatio	37.63	38.05	37.81	51.38	35.69	41.82
	InfoGain	37.63	38.05	37.81	51.34	35.69	41.83
	Symmetrical	37.63	38.05	37.81	51.41	35.69	41.83
smote	None	56.19	56.31	38.05	74.80	46.77	56.94
	cfs	54.07	41.04	37.46	62.74	41.44	45.42
	correlation	56.15	56.33	38.05	**75.18**	46.99	56.99
	GainRatio	63.10	58.05	39.74	73.89	47.68	57.74
	InfoGain	63.10	57.95	39.75	73.88	47.67	57.74
	Symmetrical	63.09	58.06	39.77	73.85	47.69	57.77
5050	None	49.59	51.19	38.03	72.09	43.70	53.36
	cfs	51.24	40.80	37.35	61.47	40.34	44.33
	correlation	49.60	51.17	38.02	72.33	43.97	53.38
	GainRatio	57.04	56.64	39.67	70.62	45.55	55.01
	InfoGain	57.10	56.63	39.69	70.64	45.58	54.99

Symmetrical	57.04	56.64	39.68	70.60	45.55	55.02

Tracing Requirements top Source Code Using Machine Learning

Falessi et al. [64] aimed to present and evaluate a novel family of metrics to predict the set of classes by a new requirement, called Similarity to Class's Requirements Set (R2RS). This was compared to four other families of metrics, all elaborated in Table 6.

Common data pre-processing steps like the usage of camelCase, under_score algorithms, stemming etc. were used on the training data. Furthermore, the training data was rebalanced by undersampling all valid requirement-class pairs and on an equal number of invalid requirement-class pairs.

To examine which metric family provided the best prediction results, the proportion of times a metric was selected by the automated metric selection, using default WEKA parameters SubsetEvaluation and BestFirst. On average, TLCC had the highest selection proportion, followed by R2RS, R2C, CKJM, and SQ. The study did not provide a clear indication which ML classifier performed bests.

Table 6: Overview of feature families used by Falessi et al. [64].

Classification Feature	Description
Similarity to Class's Requirements Set (R2RS)	Given is an existing code class C, which associated with a set of previously implemented requirements R. The idea is that a new requirement which is semantically like R, is more likely to impact C. This idea is captured in 18 R2RS metrics.
Requirement-to-Class Similarity (R2C)	Vector Space Model and Jensen Shannon Divergence were used to calculate the similarity between requirements and classes.
Temporal Locality of Class Changes (TLCC)	If a class is frequently changed in the past, then these are likely to be impacted by future change. TLCC takes the class's modification history in consideration in three different measurements.
Complexity via CKJM (CKJM)	Classes with low cohesion, or lots of public methods, are likely to be changed. Therefor common coupling and cohesion metrics are calculated.
Bad Smells via SonarQube (SQ)	Code smells are patterns in code which indicate a possibility for refactoring. Using SonarQube, each class was analysed for code smells directly after a new version.

Tracing Code Commits to Issues Using Machine Learning

Rath et al. [56] presented and evaluated an approach to recover trace links between code commits and issues. The authors tested multiple ML classifiers, trained on process-related information and textual similarity data, further elaborated in Table 7. Part of the research was to evaluate which feature set yielded the best results. Therefore, four experiments were done: 1) solely the process-related information, 2) solely the similarity data, 3) all features, 4) automatically selected features using Weka's inbuilt auto-selection feature.

Table 7: Overview of feature families used by Rath et al. [56]

Classification Feature	Description
Process-related information	16 metrics are defined which are related to the process of committing. These include stakeholder-related information, temporal relations between issue and commits, closest previous linked commits, closest subsequent linked commit, number of issues and existing links.
Textual similarity between artefacts	VSM, VSM with N-gram enhancements, and LSI were used to compute the cosine similarity. Each document was treated as an unstructured bag of terms. Common pre-processing steps like, removal of stop words, stemming, splitting on camel case and snake case words. Each term is weighted using TF-IDF.

Their approach needed to support two different scenarios. In the first scenario, the approach is used as a recommender system. Whenever the developer commits a change, the system should present a list of at most three related issues. Subsequently, the developer can manually trace the related issue. For this scenario to work, a high recall is important. One wants to make sure that the three recommended issues are indeed valid. In the second scenario, the system should provide full automated augmentation of trace links between commits and issues. The goal for this scenario is a high precision.

The results indicated the approach performed best using the Random Forests algorithm trained on all features, achieving an average recall of 96%.

Tracing Requirements to Design Documents using Deep Learning

The goal of the study is to trace requirements to design documents [58]. Their approach was designed with three scenarios in mind. In the first scenario, the approach is trained on manually constructed trace links, which then can be used to automate the production of other trace links. In the second scenario, the approach is learned on a complete set of trace links, which then can be used to find missed trace links, Finally, in the third scenario the approach is trained on a complete set of trace links, which could be used to identify trace links in a project in a similar domain.

The approach is divided into two phases: a word embedding mapping layer, and a semantic-relation evaluation layer. Its architecture is illustrated in Figure 11.

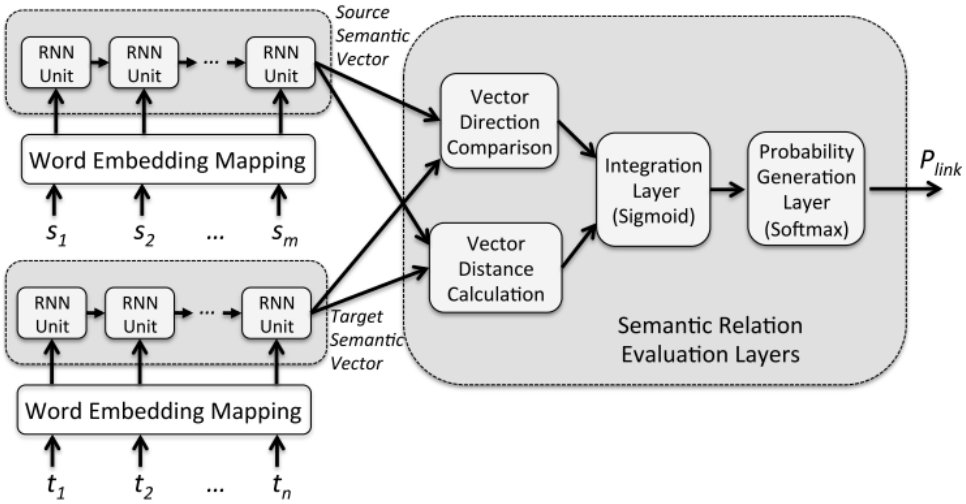


Figure 11: Architecture of the tracing method of Guo et al. using a RNN [58]

Word embedding mapping layer
First an unsupervised learner is trained on set of requirements, which returns a vector containing word embeddings. Then a set of labelled trace links is used to train a tracing network. Within the Tracing Network, Recurrent Neural Network algorithm (RNN) learns the representation of artefact semantics. For each requirement, the word-embedding vector is fed into the RNN, which in return outputs a vector representing the semantic information of the requirement.

This process is repeated for the design documents. The results of this process were passed to the next layer.

Semantic-relation evaluation layer
In this layer, the tracing network compares the semantic vectors of two artefacts, by calculation the direction and distance between pairs. The resulting vector is then passed to the sigmoid and softmax functions, which then output the probability that they are linked.

Results
To benchmark the effectiveness of the tracing algorithm they calculated the Mean Average Precision (MAP). This was done by calculating the average precision of each individual query, followed by taking the mean. The results indicate that deep learning approaches can be used for TLR and perform significantly better (MAP = .834) than IR-based TLR approaches, like VSM (MAP= .625; $p < .001$) and LSI (MAP = .637; $p < .001$).

3.3.7 Lessons Learned
From the existing approaches, we can extract some lessons learned.

1) The approaches for trace link recovery support different scenarios, and they have therefore different requirements regarding performance. For instance, for semi-automatic tracing high recall is important, while a fully automated system would benefit more from high precision.

2) Defining correct feature families. Every dataset and domains offer new possibilities in terms of features. It is necessary to empirically study what feature families work best for the MDD domain.
3) Mills et al. [38] demonstrated the impact the rebalancing technique has on the performance of the classifier. We must carefully consider, which method to employ.
4) Multiple classifying algorithms need to be examined. Although Rath et al. [56] and Mills et al. [38] concluded RF performed best, results of other algorithms need to be studies.

4. Problem Investigation

One of the key lessons from the literature review was that RTR systems could support various scenarios. Before designing a treatment, we must first establish which scenarios may be supported. To do so, we needed to study the current situation, which we accomplished by conducting semi-structured interviews at Mendix. During those interviews, we gained a deeper knowledge of the Mendix Platform and discovered that requirements were maintained in Atlassian JIRA.

This will be covered in greater depth in this chapter. First, Section 4.1 introduces the Mendix Platform. Then, Section 4.2 will discuss Atlassian JIRA. The section that follows describes the interaction between the Mendix Platform and Atlassian JIRA. Finally, the chapter concludes with a description of two identified scenarios a RTR system could support.

4.1 The Mendix Platform

4.1.1 Mendix Studio (Pro)

Mendix Studio is the area of the Mendix software suite in which the developer creates their application. A screenshot of the user interface is shown in Figure 12. The software is available in two editions: Studio and Studio Pro. They differ in the category of users it focuses on, where the former focuses on non-technical business users and the latter focuses on professional developers [65]. However, both editions utilize the principles of MDD: the developer creates an application model by means of pages, domain model, microflows, and navigation document. When creating a new application, a new Mendix Project File (.mpr) is created.

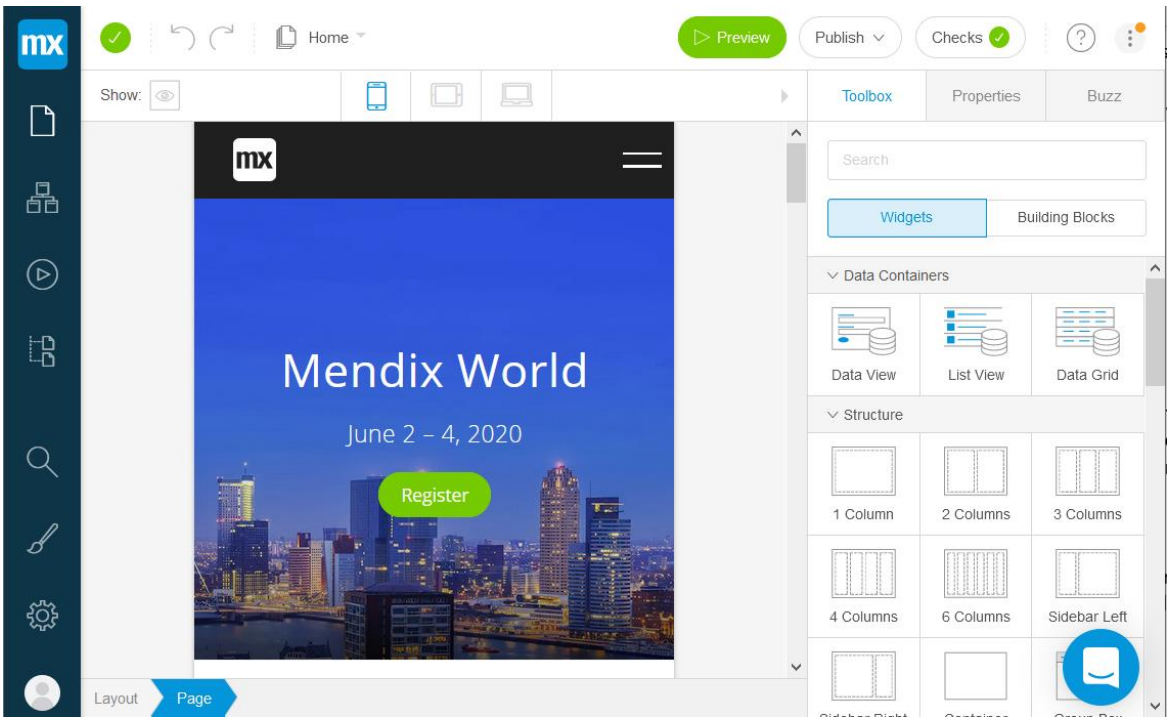


Figure 12: The user interface of Mendix Studio

4.1.2 Mendix Developer Portal

Mendix wants to support the entire Agile application life cycle. To do this, they offer a basic project management functionality in the form of the Developer Portal, shown in Figure 13. It offers tools to manage user stories, end-user feedback, and sprints. User stories can be added in the Stories tab. Whenever a new story is created, the user can fill in a form containing the following fields: title, description, story type, story points, related sprint, and story status. The user input is not restricted to any template.

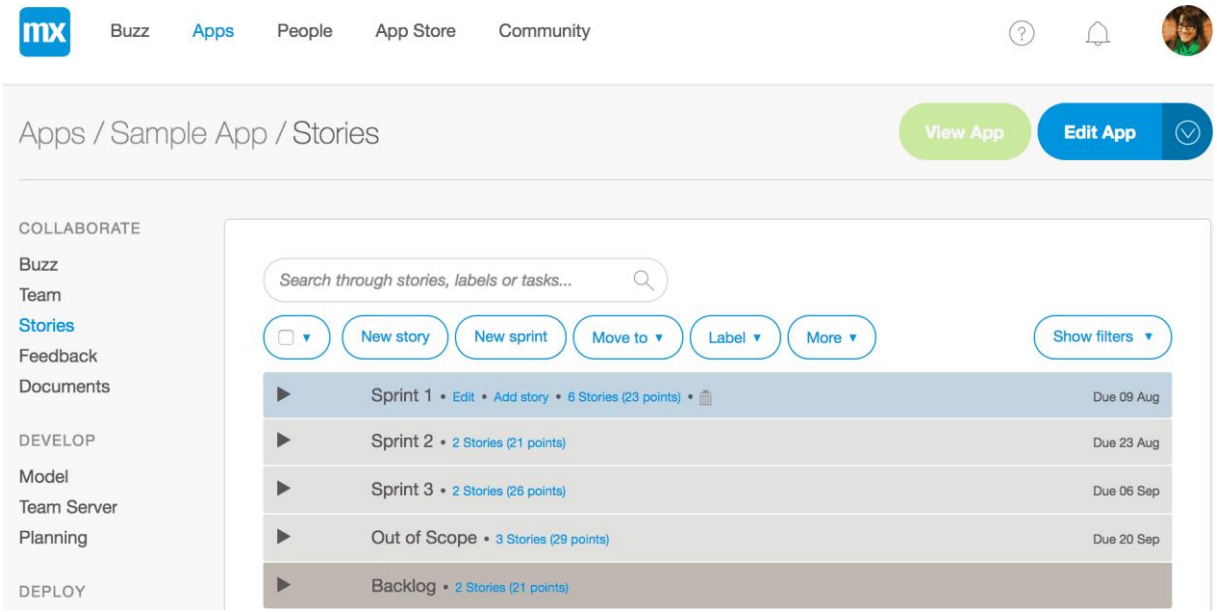


Figure 13: Developer Portal is the user interface of Mendix to support management functionality

4.1.3 Mendix Servers

For every application created a repository on the Mendix Team server is provided. In essence, this is an adapted version of the open-source version control system Apache Subversion. To date, all Mendix servers run this. In the future, the version control system will be migrated to git.

Whenever a change is made in the application model there are two versions of the application in existence: one locally stored in Mendix Studio and one remotely on the Team Server. To save the local changes to the remote repository, the developers need to commit the changes to the Team Server. In the backend, this is done using Subversion SVN, however the developer is only shown the GUI as shown in Figure 14. In the top you are shown in which branch you are committing, and you can provide a message, describing the changes you have made. Furthermore, there are 3 tabs visible: Related Stories, Changes in model, and Changes on disk. In the Related stories tab, you can relate stories to the commit. The stories shown, are the ones the user stories made in the Developer Portal. In the Changes in model tab, you see all units added, modified, or deleted in the commit. Finally, in the Changes on Disk tab you find an overview of all changes, outside of the .mpr file.

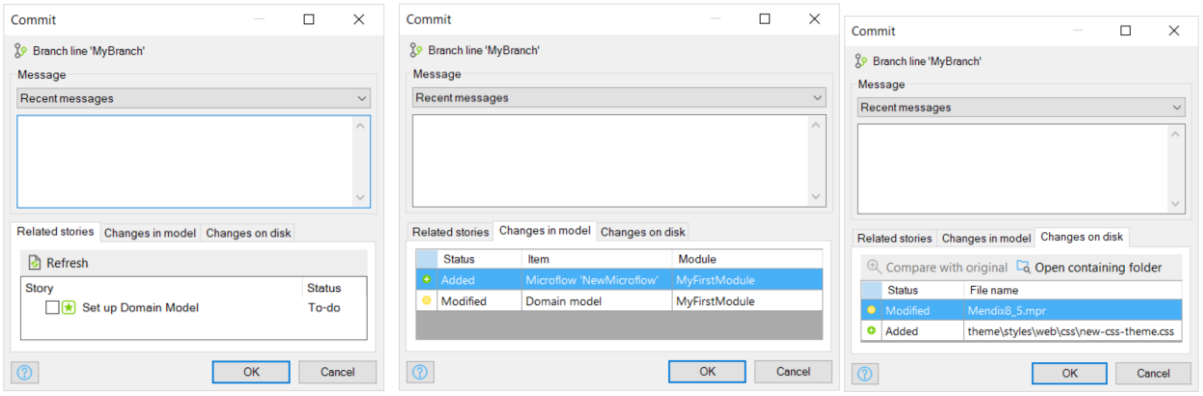


Figure 14: The GUI in Mendix Studio Pro to commit changes to the Team Server. The developer can see all open stories in SPRINTR (left), changes made to the model (middle), and all changes made on disk (right)

4.2 JIRA

Atlassian JIRA is a software tools designed for teams to manage their projects [66]. The software is in development since 2002 and it is used by more by 65,000 companies worldwide [67]. JIRA comes with a variety of features and templates, which can be tailored to the specific needs of the team. For instance, there are project templates available for human resources, finance, design, and more.

When creating a new project for a software development, templates for Kanban, Scrum, and Bug Tracking are available. Once the template is initialized, 'JIRA issues' can be placed on the board. JIRA issues are work items and are categorized into 5 types: Epic, Story, Bug, Task, and Sub-task. These types are set up like a hierarchy, which is shown in Figure 15.

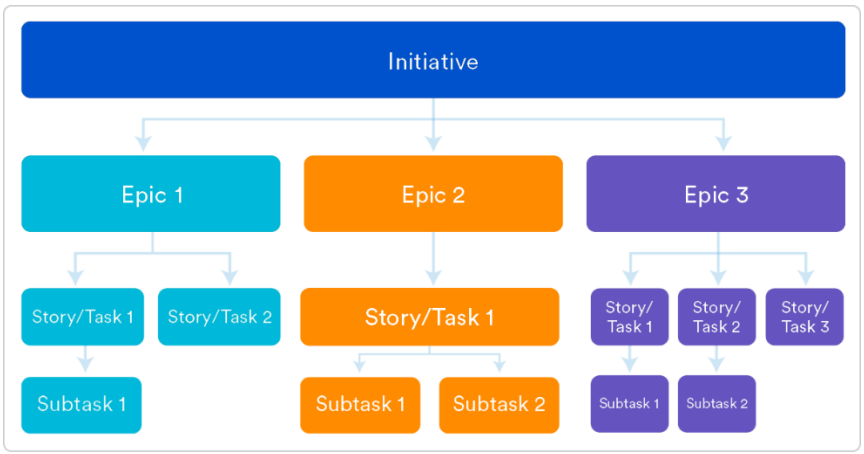


Figure 15: Overview of the JIRA issue hierarchy [68]

4.2.1 Epics and Stories

When working on a project, it is good practice to break down the project into smaller work items [69]. Within JIRA, an Epic assists in the breakdown of work, by offering the means to organize the work and create hierarchy. They can essentially be seen as a collection of related work items. Epics are meant to be flexible, as they extend over a set of sprints. During the project work items can be added and removed, purporting the new requirements. One of these work items are user stories.

A user story is a method for documenting requirements from the perspective of the user. In practice, 70% of the user stories follow the Connextra template: “As a <type of user>, I want to <some goal>, so that <some reason> [70]. It consists of three elements. First, the <type of user> relates to the role of the person for whom the requirement is created. The <some goal> concerns the objective the user wants to fulfil. Finally, the <some reason> provides the motive to why the user wants the requirement.

In JIRA, a user story is like any other work item and can thus be supplemented with extra details. For instance, it features the possibility to add a description or attachment, assign it to a specific team member, or add comments to it. Figure 16 shows an example of a JIRA user story.

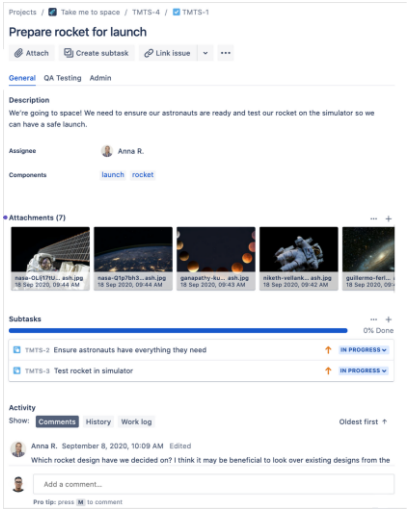


Figure 16: Example of a JIRA User Story

4.3 Case description of Mendix

At Mendix, developers are working with the SCRUM development process. The life cycle of a requirement: from a definition to implementation is described below. This lifecycle can be divided into two phases.

In the first phase the team work on the “definition of ready”, determines all elements necessary for a requirement to be considered for a sprint. These requirements are ideated during discussions between the product owners (POs) and the customers. Then these are documented as user stories, following the Connextra template, and added to the product backlog.

When the product owner wants the story implemented, the development team comes together for a refinement session. In the refinement session, the user story is refined by adding details of the technical and functional aspects (e.g., UX, software dependencies). Once the user story is fully refined, it can be considered for the sprint planning. In the sprint planning it gets decided which user stories are going to be implemented in the coming sprint. After the sprint planning is completed, the second phase can be started.

In the second phase the team is working on the “definition of done”, which defines all elements required for a user story to be considered implemented. This is operationalized during a sprint, when user stories are assigned to a single developer, who is subsequently responsible for the implementation.

The process of implementing a user story can be summarized into a number of activities. First the developers read the user story to get familiar with what needs to be done. Then he/she opens latest Mendix model and navigates to the modules, which need to be changed. These modules are changed until the pre-defined acceptance criteria, found in the JIRA issue, are met. Then the developer creates documentation for other developers to understand their work. Finally, the work can be committed to the Mendix Team server. When committing, the developer also puts the JIRA issue ID in the commit message. By doing this, they ensure a form of requirements traceability. When looking back at the history of past commits, the traces can be seen in the messages. An example is shown in Figure 17.

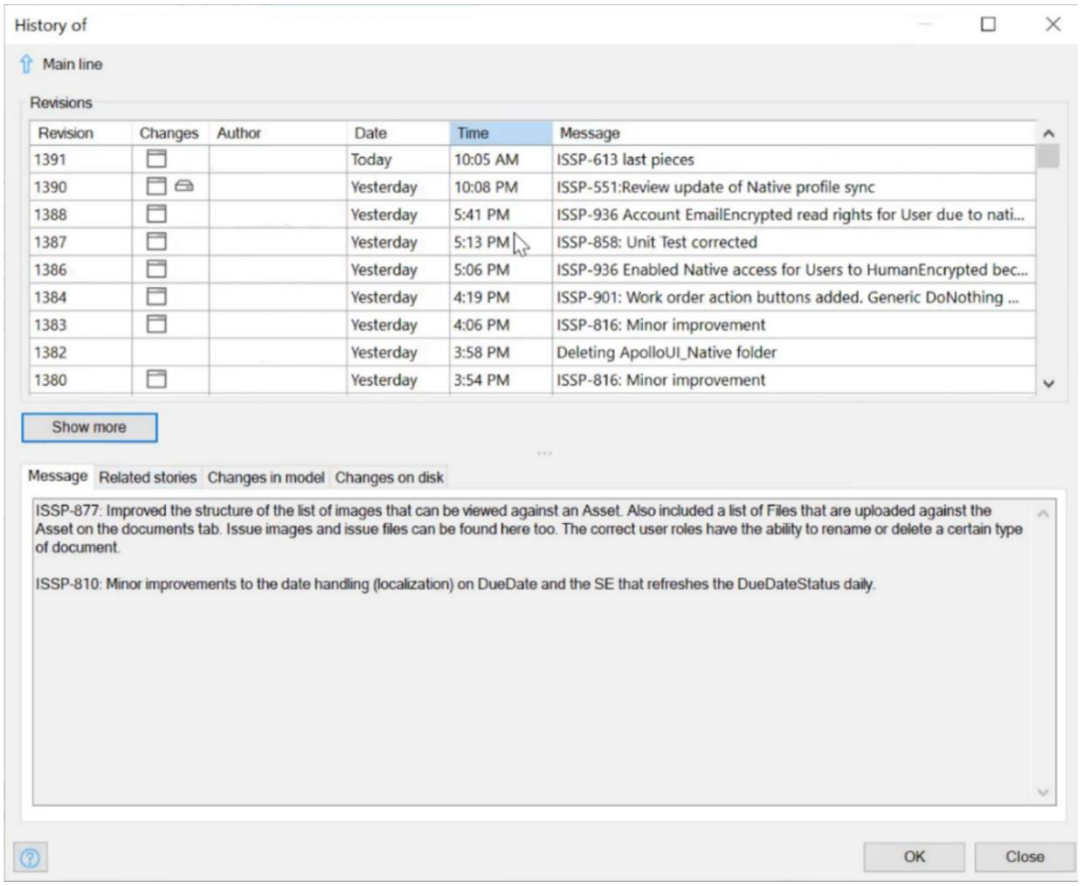


Figure 17: Commit history of a project in Mendix containing requirement traces

4.4 Desired solution

According to the interview findings, establishing traces from commits to JIRA issues is a manual process within Mendix. Evidenced by earlier research, this is prone to errors. In this study we have analysed data that originates from multiple projects. In this data, we observed errors, which include misspelling, incorrect values, misformatting, or absence of trace links. This is not unusual, because it is a human task.

It is apparent that these problems can be mitigated by introducing an automatic system to the process. After analysing this process, we have identified two scenarios that could offer opportunities to the case company.

1. When a developer needs to commit his/her changes to the Mendix Team Server. Recall, that the developer opens a commit dialog in which he/she describe their changes in the commit message together with a trace. In this situation, there is an opportunity to add a recommendation system into the commit dialog. This system can show the developer all possibly related JIRA issues. The only manual task remaining, is for the developer to check the valid traces. A mock-up of this recommendation system is shown in Figure 18**.** For this scenario to work, *high recall* is required. The reason for this is that for a developer to examine a valid trace, it must first appear in the list. Precision is of less importance in this scenario, since developers can leave invalid traces unchecked.

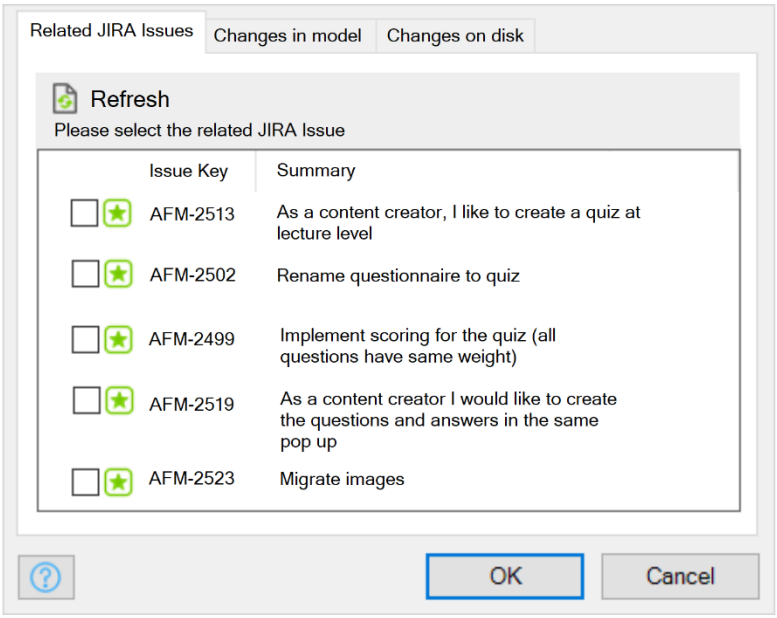


Figure 18: Mock-up of a trace recommendation system

2. Another problem is that not all commits are traced to a JIRA issue. For designing our solution, we have obtained two datasets for theory building. Only 86 percent of the commits in one dataset were tracked, whereas only 71 percent in the other. This shows that maintenance is required, to recover traces for the untraced commits. This is the goal of the second scenario: a fully automated trace maintenance system is introduced into the project. This system would periodically recover traces, which were forgotten by the developer, which would ultimately lead to a higher level of RT in the project. For this scenario to work, a high precision is needed. The reason for this is that there is no human intervention in this scenario to correct invalid traces. The system needs to ensure each predicted trace is truly a valid trace.

5. Treatment Design

In Section 4.4, we introduced two possible treatment scenarios, which can improve the current situation. In this chapter, we will go over the design and development of the treatment. First, the raw datasets acquired will be explored and described. Next, the procedure for pre-processing the data is outlined. This is followed by a section that describes all features that represent these traces. Then, the imbalance in the obtained data is demonstrated, as are the strategies for dealing with it. Finally, the classification algorithms are examined.

5.1 Initial Data

Mendix provided us with data on four of their internal software projects for the study: Project 1, Project 2, Project 3, and Project 4. For each project 2 datasets were given: the JIRA export data of the respective project, and a data dump of the Subversion dump file. Table 8 gives an overview of the data supplied for each project.

Table 8: Overview of the data quantity for each internal project we have obtained from Mendix

Project	**Number of Tuples**	
	JIRA export data	Subversion Dump File
Project 1	994	3663
Project 2	58	818
Project 3	173	2929
Project 4	634	713

5.1.1 JIRA Dataset

The JIRA datasets are delivered in either .xml or .csv. Each tuple represents a JIRA issue, together with its metadata. Below, we will discuss the metadata used for the study.

1) *Summary:* A concise description of maximum 255 characters. Within Mendix, this field is often used for documenting the user story (as… I want to… so that…), although this is not always the case.
2) *Issue key:* The unique id of the issue as specified by Mendix. It is formatted as the project code, followed by an incremental integer (e.g., AFM-3184 or AFM-3185).
3) *Assignee:* The person who is responsible for implementing the JIRA issue. It is documented as the first name plus last name (e.g., Randell Rasiman)
4) *Comment:* Remarks people have given on the JIRA issue. Each comment creates another column. For example: An issue with 2 comments contains the columns 'Comments' and 'Comments.1'. An issue with 3 comments has the columns 'Comments, 'Comments.1', and 'Comments.2'. It is important to note that, out of the four datasets received, only the Project 2 and Project 3 datasets contained comment data. The comment data was missing from the Project 1 and the Project 4 datasets.
5) *Description:* A written account, which further explains the requirements for implementing the JIRA issue.
6) *Resolved:* The datetime on which the JIRA issue its status was marked resolved.
7) *Created:* The datetime on which the JIRA issue was created.
8) *Updated:* The datetime on which the JIRA issue was last updated.

5.1.2 SVN Dataset

The Subversion dataset is provided as a .txt file. Each tuple represents a revision done in a project. Below we will describe the data used for the study. An overview of all revision data is given in Figure 19.

1) *revision-number:* an ascending integer, starting from 0. Revision-number 0 is reserved for the initialisation of the project. Revision-number 1 is reserved for the initialisation on the Mendix Teams server.
2) *author.* This is the user who committed the revision and is stored as an email address (e.g., randell.rasiman@mendix.com).
3) *log.* An optional log message of the commit, in which the user can describe the changes he or she has made. The author often includes the issue-key of the related JIRA issue inside the log.
4) *date.* That is the datetime on which the revision was committed by the user. It follows the ISO 8601 UTC Zulu standard.
5) *metadata.* This attribute provides metadata which include the branch name, modeler version, model changes, related stories, and whether it's made in Studio or Studio Pro. The 'ModelChanges' attribute in the metadata is formatted as a JSON-objects. For this study we mainly make use of the UnitName attribute. It contains the name given by the developer to a unit (e.g., microflow or form). The rationale behind this is that the given names are often describing the functionality of the unit.

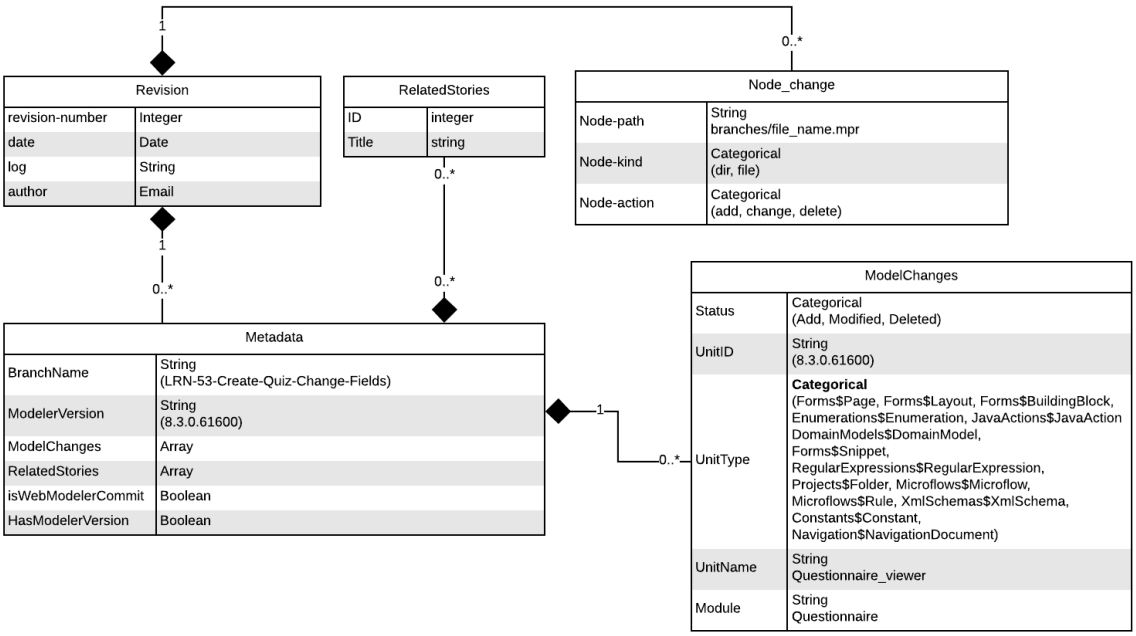


Figure 19: Overview of the data making up a revision

5.2 Pre-processing

5.2.1 Loading and pre-processing

The solution was created in a Jupyter Notebook [71], which can be found in de online Appendix. The data must be in tabular format for further processing. Because the acquired SVN data was given in text format, the data had to be transformed first. This was done using Regular Expressions (REGEX). Next to transforming, REGEX was also used to extract the issue-key(s) from the log message and store it in a distinct issue-key column. Because a classifier can only be trained using labelled data, only revisions containing an issue-key were retained, while revisions without were discarded. The implications of this decision are shown in Table 9. It is noticeable that the Project 3 and Project 4 contain significantly less labelled revisions than the other 2 project. This is due to the fact that multiple development teams, each with their own JIRA-project, may operate on a single software project. As a result, the commit history may contain JIRA issue-keys from multiple JIRA projects. This was especially prevalent in the Project 3 and Project 4 projects. For our research, we have solely focussed on tracing the commits to the JIRA issues present in one of our four obtained datasets. As a result, commits traced to issue-keys not belonging to the acquired JIRA projects, were marked as unlabelled and therefore discarded.

Table 9: Overview of the proportion of labelled data in the various datasets

	Total Revisions	Number of Labelled Revisions	Number of Discarded Revisions
Project 1	3663 (100%)	3159 (86.24%)	504 (13.76%)
Project 2	818 (100%)	583 (71.27%)	235 (28.73%)
Project 3	2929 (100%)	1495 (51.04%)	1434 (48.96%)
Project 4	713 (100%)	206 (28.29%)	507 (71.11%)

The JIRA datasets did not need any extra modifications were required for loading, since these were already in a tabular format. After loading the data into the environment, all natural text was pre-processed using six common pre-processing methods. This was done for JIRA as well as the SVN dataset.

1) All words were lowercased.
2) All the interpunction was removed.
3) All numeric characters were removed.
4) All sentences were tokenized with NLTK.
5) The stop words corpus from NLTK was used to eliminate all stop words.
6) All remaining terms were stemmed using the Porter Stemming Algorithm [72].

5.2.2. Trace Link Construction

After loading and pre-processing both datasets, we can construct the candidate trace links by calculating the Cartesian product between the JIRA dataset and the SVN dataset. For each trace link, the validity was determined by checking if the JIRA issue-key was present in the commit log. If the JIRA issue-key was present, the trace link was classified as valid; if the JIRA issue-key was not present, the trace link was classified as invalid. Furthermore, we applied causality filtering to the trace links [56]: when a trace link had a SVN commit that was committed prior to the creation of a JIRA issue, it was deemed invalid due to causality. Table 10 shows an overview of the outcomes of these activities.

Table 10: The amount of valid trace links in various acquired datasets

Dataset	Before/after Filtering	Number of traces	Valid traces
Project 1	Before	3,139,052	3104 (0.10%)
	After	1,375,042	3104 (0.23%)
Project 2	Before	33,756	451 (1.34%)
	After	27,815	451 (1.62%)
Project 3	Before	258,635	420 (0.16%)
	After	89,233	420 (0,47%)
Project 4	Before	129,970	86 (0,07%)
	After	33,627	86 (0.26%)

5.3 Feature Families

The previously produced set of candidate traces can now be used for training the classifier. However, for the classifier to distinguish the valid traces from the invalid traces, the candidate trace links need to be represented as a set of features. In total 154 features are engineered. In this section we will describe these features, which fall into 4 categories: Process-related, document statistics, information retrieval and query quality.

5.3.1 Process-Related

The process-related category is based on work of [56] and consists of four features. The first feature captures stakeholder information by indication whether the assignee of a JIRA issue assignee(I) is the same person as the author of a commit author (C). The remaining three features capture temporal information. This is accomplished in three ways:

1) The difference between the date of commit and the date of the JIRA issue was created.
2) The difference between the date of commit and the date the JIRA issue was last updated.
3) The difference between the date of commit and the date the JIRA issue was resolved.

5.3.2 Document Statistics

The document statistics is based on the work of [38], and include features to gauge document relevance and the information contained within the documents. Within this category seven metrics are included:

1) The total number of terms. This is both calculated for the JIRA issue and the commit.
2) The total number of unique terms. This too, is calculated for both the JIRA issue and the commit.
3) The overlap of terms between the JIRA issue and the commit. To calculate this metric, the overlap of terms is divided by the set of terms you are comparing it to. Because this may be accomplished in three distinct ways, each of these approaches is treated as a separate feature.
    a. Overlap of terms divided by the terms in the JIRA issue
    b. Overlap of terms divided by the terms in the commit
    c. Overlap of terms divided by the union of the terms between the JIRA issue and the commit.

5.3.3 Information Retrieval

The Information Retrieval feature set capture the semantic similarity between two trace artefacts. The was done by first applying VSM with TF-IDF weighting to transform the trace artefacts to a vector representation. Because we use TF-IDF weighting, the chosen corpus used for weighting impacts the resulting vector. For instance, the term 'want' occurs commonly in the JIRA summary, since Mendix developers put their user story in there. However, it might be a rare term when taking in account all the terms in a JIRA issue. Since we do not know which corpus best represents the trace artefact, we opted to explore multiple representations. As a result, we have constructed the JIRA issue vector representation with four corpora and the SVN commit with three corpora. This results in a total of 12 distinct pairs for each trace link candidate, as shown in Table 11**.** The cosine similarity of each pair was computed and utilized as a feature.

Mills and Haiduc [62] showed that the chosen trace direction (i.e. which artefact in the trace link is used as a query has an effect on performance, especially for traceability. For this reason, we calculated the cosine distance in either direction, resulting in a total of 24 IR-features. We used Scikit-learn [73] for TF-IDF weighting and SciPy [74] for calculating the cosine distance.

Table 11: VSM with TF-IDF weighting features

ID	Artefact 1	Artefact 2
1	SVN Log Message	JIRA Issue (Summary, Description, and comments)
2	SVN Log Message	JIRA Issue Summary
3	SVN Log Message	JIRA Issue Description
4	SVN Log Message	JIRA Issue Comments
5	SVN Unit Names	JIRA Issue (Summary, Description, and comments)
6	SVN Unit Names	JIRA Issue Summary
7	SVN Unit Names	JIRA Issue Description
8	SVN Unit Names	JIRA Issue Comments
9	SVN (Log Message, Unit Names)	JIRA Issue (Summary, Description, and comments)

10	SVN (Log Message, Unit Names)	JIRA Issue Summary
11	SVN (Log Message, Unit Names)	JIRA Issue Description
12	SVN (Log Message, Unit Names)	JIRA Issue Comments

5.3.4 Query Quality

In IR, queries are used to retrieve information from a document collection. However, the succession of finding the right documents also depends on the quality of the query. This also applies when IR is used for traceability. It makes a difference whether two artefacts in a candidate trace link have a low cosine similarity because a) they are truly an invalid trace pair or b) the quality of the query artefact is low.

Mills et al. [63] devised a number of metrics, which can infer the query quality (QQ). We have implemented 17 pre-retrieval QQ metrics, assessing three different aspects: 1) *specificity*, referring to the query its ability to capture the information need, 2) *similarity*, relating to the similarity between the query and the entire document collection, and 3) *term relatedness*, referring to how often terms in the query co-occur in the document collection.

We encountered issues while computing the QQ. Our RAM capacity of 16GB was insufficient to complete the term relatedness QQ calculation of the Project 1 dataset and resulted in a crash. Due to this, as well as a shortage of training time, forced us to discontinue further analysis of the dataset.

We did, however, manage to complete the computation of the 17 QQ metrics for the Project 2, Project 3, and Project 4 datasets. The computation was repeated, using all seven corpora mentioned in Section 5.3.3, since the outcome of several QQ metrics is dependent on the corpus of which the query is a part. This resulted in a total of 119 QQ features.

5.4 Data Normalisation

Normalisation of the data may lead to a reduction of estimation errors of the model in its hypothesis class, or can yield an faster algorithm [54]. Within the field automated RTR, some studies have included data normalization as part of their pre-processing procedure [38], while others have not [61]. However, none of the previous studies addressed the normalisation variable for evaluation, leaving the effect uncertain.

We wanted to investigate how data normalisation may enhance our treatment as part of our research. As a result, we produced two variants of our data. In the first variant, the data was not normalised and therefore remained unaltered. In the second variant, we normalised the data by applying a Min-Max [0, 1] scale to the values of all our features.

5.5 Rebalancing

As was explained in Section 3.3.4 Machine Learning, the construction of the candidate trace links result in a highly imbalanced dataset, which is also observable in Table 10. For this study, we evaluated four different strategies, proposed by Mills et al. [38], to deal with this imbalance:

1) *None*. There is no rebalancing method applied to the data.
2) *Oversampling*. The minority class is oversampled until it reaches the size of the majority class, by applying SMOTE. This is the default setting in Scikit-Learn.
3) *Undersampling*. The majority class is randomly undersampled until it has the same size as the minority class, by applying the random undersampling technique. This is the default setting in Scikit-Learn.
4) *5050*. In this strategy we are combine over- and undersampling. First, the minority class is oversampled using SMOTE with a sampling strategy of 0.5. Then undersampling is applied to the majority class until the sizes of both classes are equal.

5.6 Classification Algorithms

For the design of the treatment, we looked at two supervised machine learning algorithms for classifying trace links as valid or invalid. These were Random Forests and Gradient Boosted Decision Trees. These motivation for these two algorithms is twofold. First, Random Forests is shown to be the best classifier in RTR by earlier research [38], [56]. Second, Gradient Boosted Decision Trees have demonstrated to outperform Random Forests in other domains [75], [76].

To implement the Random Forest algorithm, we used the framework of Scikit Learn. To implement the Gradient Boosted Decision Trees we used two different frameworks: XGBoost, and LightGBM. These frameworks differ in two major respects [77], [78]. The first distinction is in the method of splitting. XGBoost splits the tree level-wise rather than leaf-wise, whereas LightGBM splits the tree leaf-wise. This distinction is illustrated in Figure 20.

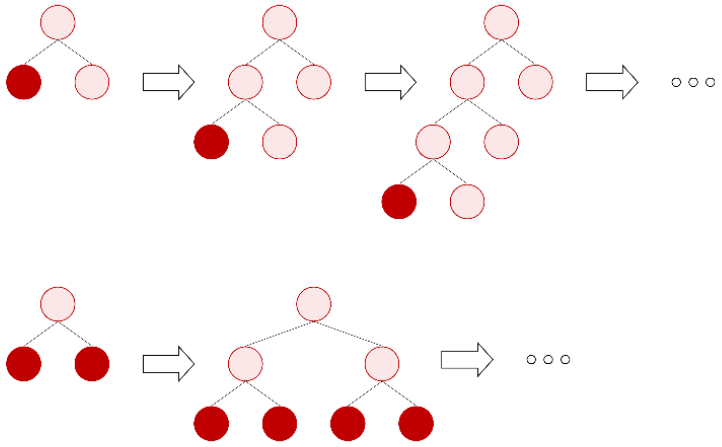


Figure 20: Leaf-wise Tree Growth (top) and Level-Wise Tree growth (bottom)

The second distinction is the method of determining the best split value. XGBoost uses histogram-based algorithm, which splits a feature its data points into discrete bins. These bins are then used to find the best split value. LightGBM uses a subset of the training data rather than the entire training dataset. It employs a sampling technique, called Gradient Based One Side Sampling, which samples the training data based on gradients, resulting to significant faster training times.

5.7 Hyperparameter tuning

All three classification frameworks have various hyperparameters which can be changed. Changing these hyperparameters may change the performance of the models. Due to the computational time needed to tune the hyperparameters, we only considered 5 hyperparameters per framework. These hyperparameters were chosen based on their popularity, which was determined as follows:

1) Query Google.com with "<Model Name> + hyperparameter tuning".
2) Collect all articles on page 1, from the domains: Medium, Towardsdatascience, or Analytics Vidhya.
3) Tally the hyperparameter mentions in the articles.

The top 5 most tallied hyperparameters are considered for tuning in the study. Sections 6.3.2 and 6.4.2 go into further detail on this subject.

5.8 Summary

In this Chapter we explained that we have represented the trace into a total of 154 features. These features are classified into four families: Process-Related (4), Document Statistics (7), IR-Related (24), and Query Quality Metrics (119). In addition, we demonstrated a variety of settings and strategies for selecting classification algorithms, rebalancing training data, adjusting hyperparameters, and data normalisation. In the next chapter, we will experiment with these settings and strategies to identify the best performing model to help with the scenarios described in Section 4.4.

6. Results

In the previous chapter we have covered all components of the treatment. These components can be configured in a variety of combinations, each of which yields a different result. In this chapter, we will experiment with these configuration combinations, in order to find the best configuration. First, we will determine the baseline results of our treatment. This baseline will serve as a reference point of the possible performance. Then, we discuss the experimentation with Min-Max [0,1] normalization of the training data to see if this process can significantly impact our performance. We considered all 154 features when obtaining these results.

From there, the scope of the experiment will be narrowed by focussing on the two configurations best suited to the scenarios, trace recommender system and trace recommender system, as specified in Section 4.4. We will provide an explanation in terms of feature importance for each of the two models. This explanation is followed by an experiment to find out if hyperparameter tuning is able to significantly enhance the performance.

6.1 Baseline Results

We wanted to get an initial evaluation of the performance of our treatment model, which we considered as the baseline result. We examined 12 alternative configurations of rebalancing techniques (discussed in Section 5.5) and classification algorithms (discussed in Section 5.6) for this evaluation. These 12 configurations are shown in Table 12.

Table 12: Evaluated configurations of the treatment

Rebalancing Technique	***Classification Algorithm***
None	Random Forests
	GX Boost
	LightGBM
SMOTE	Random Forests
	GX Boost
	LightGBM
Undersampling	Random Forests
	GX Boost
	LightGBM
5050	Random Forests
	GX Boost
	LightGBM

Each configuration was evaluated on every dataset independently. For this evaluation each dataset was first divided into a train and test set using an 80:20 split. Each of these splits was stratified, meaning that each class was distributed proportionally between the two splits. Then the model was trained with a stratified 10-fold cross validation on the train set and run once on the test set of which the test score is recorded. This procedure was repeated 10 times for each implementation, and averaged. A Process-Delivery Diagram [79] of this procedure is shown in Figure 21.

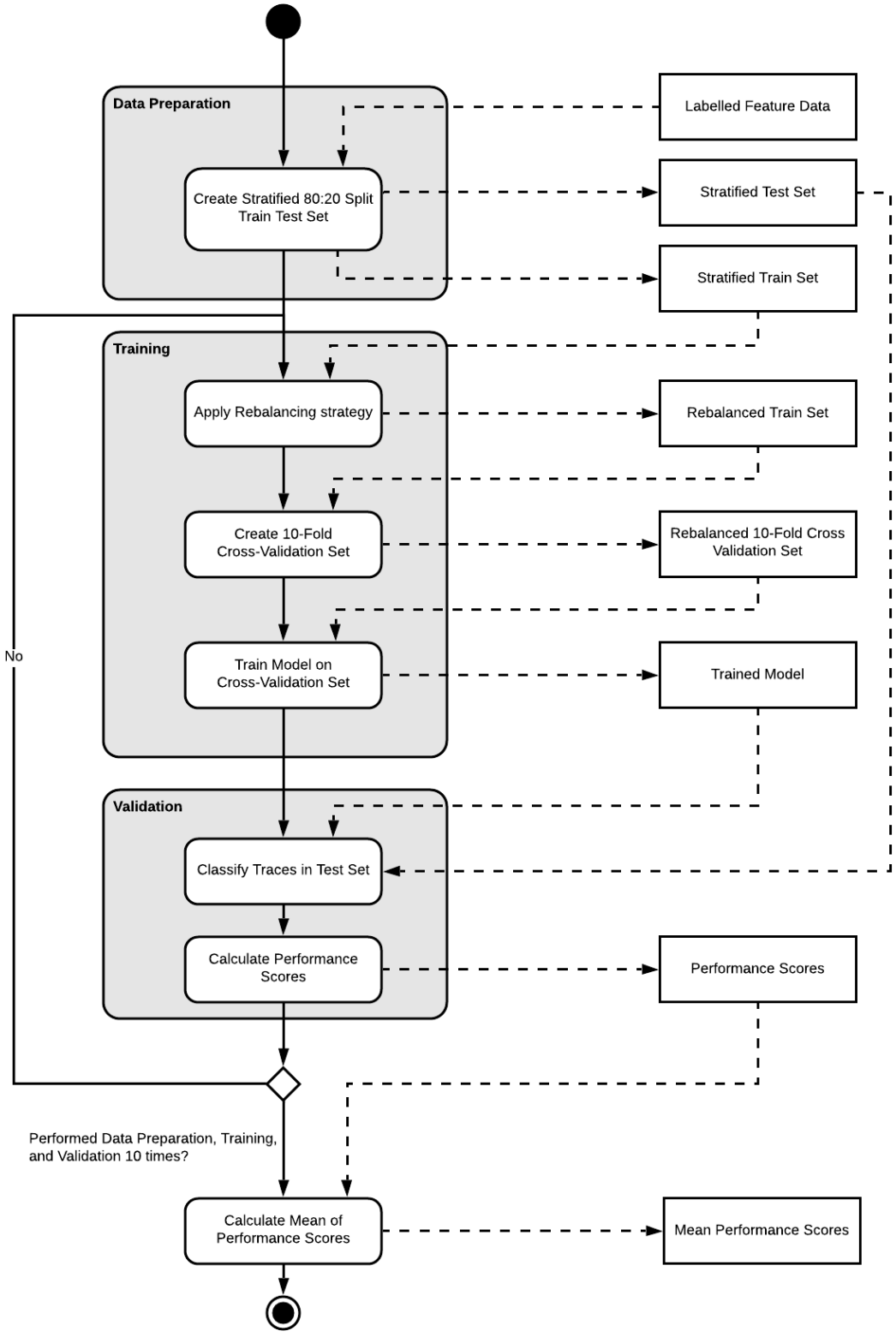


Figure 21: Process-Delivery Diagram depicting the method we used to obtain our results

The full results of these evaluations can be found in the online Appendix. Figure 22 depicts the average precision, recall, F0.5-measure, F1-measure, and F2-measure over all iterations for all configurations of the treatment design. The different rebalancing techniques are presented on the horizontal axis, while the outcomes for the various datasets are given on the vertical axis. The three different colors respresent the three different classification algorithms used. Table 13 shows the average F-measures for the three datasets.

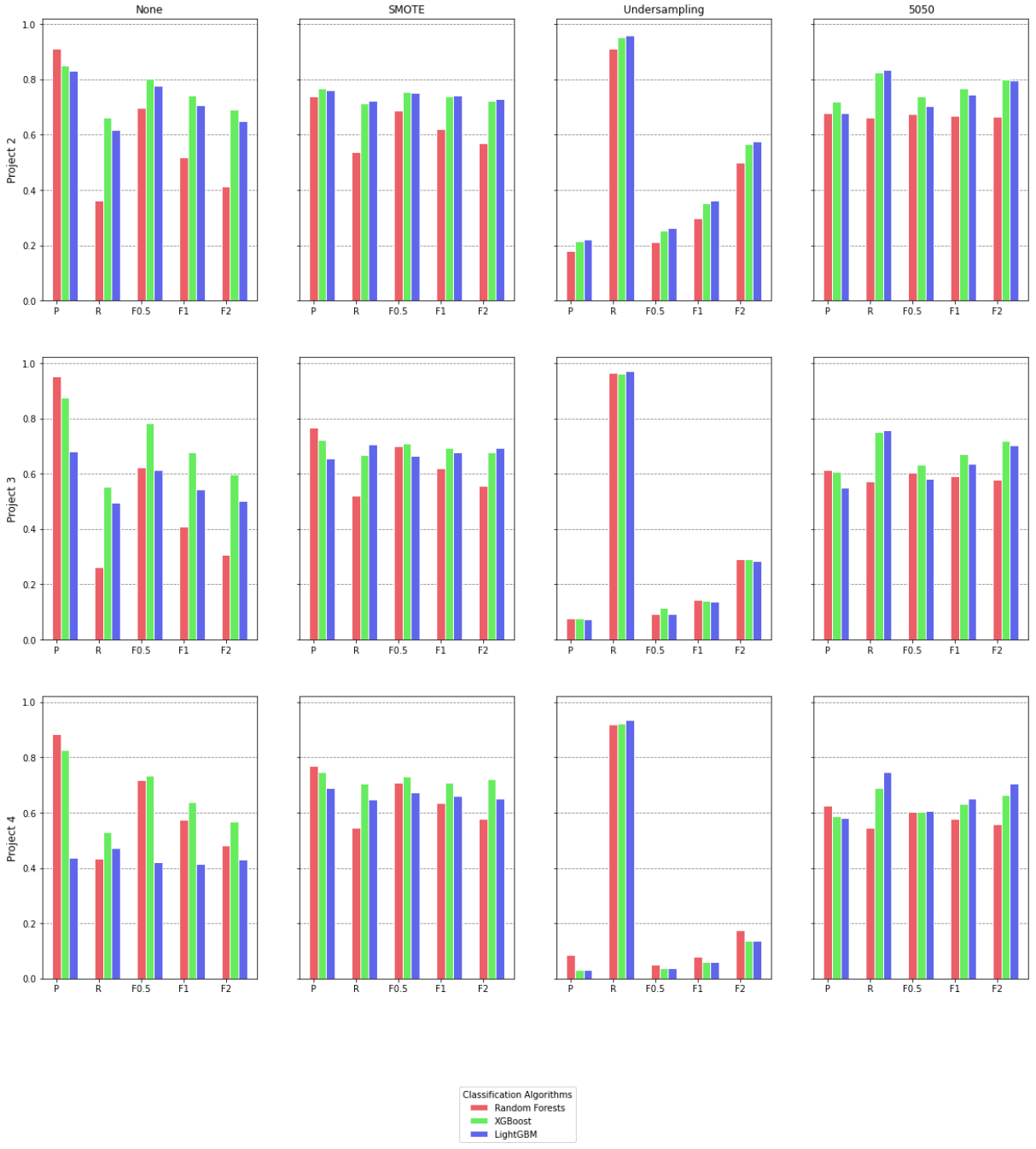


Figure 22: The mean precision, recall, F-0.5, F1, and F2 metrics for the various model configurations on non-normalized data

Table 13: Mean F0.5-measure, F1-measure, and F2-measure for the different non-normalized configurations. The asterisk denotes the combinations for which the greatest F-measure was obtained.

Algorithm / Rebalancing	Random Forests			XGBoost			LightGBM		
	F0.5	**F1**	**F2**	**F0.5**	**F1**	**F2**	**F0.5**	**F1**	**F2**
No Rebalancing	67.97	50.09	40.01	77.32*	51.41	46.38	60.35	55.45	52.69
SMOTE	69.79	62.52	56.77	73.23	71.35*	70.65	69.69	69.32	69.18
Undersampling	11.97	17.37	32.25	13.60	18.47	33.11	13.10	18.66	33.28
5050	62.76	61.25	60.03	65.80	69.03	72.73	63.05	67.78	73.48*

Upon initial review, the results indicate that using a different rebalancing approach produces the most notable variations in performance across all datasets. When no rebalancing is applied, the maximum precision can be achieved, while still maintaining a decent recall, resulting in the highest F0.5-measure (with a top value of 77.32 for XGBoost).

SMOTE reduces precision while increasing recall, resulting in the greatest F1-measure (with a peak value of 71.35 for XGBoost). This indicates that this is the optimum equilibrium between precision and recall. Our findings support the findings of Mills et al. [38], who demonstrated that using SMOTE as a rebalancing strategy results in the highest F1-score.

When we use Undersampling as a rebalancing strategy, we get the highest recall scores of any rebalancing strategy. However, the strategy produces impractical precision. As a results all undersampling configurations belong to the group with the lowest F-measures.

Altough Undersampling by itself is impractical, combining the strategy with SMOTE yields a better balance. The 5050 rebalancing strategy creates a better balance by trading recall for precision. This results in a performance that retains strong recall while providing a more practical precision. This is quantified by the F2-measure, which is greatest in the 5050 model configuration (with a top value of 73.48).

6.1.1 Statistical Comparison of Rebalancing Strategies

In this section we will evaluate if these differences were statistically significant. Because the primary aim is to establish if the rebalancing strategy has any significant effect in general, we used the F1-score rather than the F0.5-score or F2-score. Finetuning for each scenario is currently out of scope and will be explored later.

For the evaluation we ran a non-parametric Friedman test. It rendered a Chi-square score of 70.83 for Random Forests, a Chi-square score of 54.99 for XGBoost, and a Chi-square score of 75.00 for LightGBM. All three outcomes are significant with ($p < 0.01$). This indicates that the rebalancing strategies are significantly different. As a posthoc test, we use the Nemenyi test to identify which specific strategies have distinct means. These results are shown in Table 14, Table 15, and Table 16.

When setting Random Forests as a classifier, the choice of rebalancing strategy matters. The Nemenyi test scores in Table 14 indicate a couple of things: 1) None is significantly different ($p < 0.01$) than the other rebalancing methods. 2) SMOTE and 5050 are significantly different ($p < 0.01$) than None and Under. 3) 5050 and SMOTE are not significantly different ($p > 0.01$).

These results imply that the Random Forests algorithm should be used in combination with either the SMOTE or 5050 rebalancing strategies. This is because SMOTE has the greatest mean F1-score and 5050 does not perform significantly worse. It should, however, be emphasized, that 5050 should be the preferable option. Because this strategy includes undersampling, you have fewer data points to train on, resulting in a shorter training period.

Table 14: Nemenyi test results for Random Forests. The asterisk denotes the combinations which were found to be significantly different ($p < 0.01$).

	None	**SMOTE**	**Under**	**5050**
None	-	0.003144*	0.001497*	0.001808*
SMOTE	0.003144*	-	0.001000*	0.900000
Under	0.001497*	0.001000*	-	0.001000*
5050	0.001808*	0.900000	0.001000*	-

Two findings stand out, when we look at the Nemenyi test results for the XGBoost classifier in Table 15. First, the undersampling rebalancing approach differs significantly from the other three strategies. Second, the SMOTE, undersampling, and 5050 rebalancing strategies do not significantly differ from each other. Based on these two findings, as well as the fact that SMOTE gets the highest mean F1, SMOTE, undersampling, and 5050 can all be considered for use with XGBoost.

Table 15: Nemenyi test results for XGBoost. The asterisk denotes the combinations which were found to be significantly different ($p < 0.01$).

	None	**SMOTE**	**Under**	**5050**
None	-	0.77998	0.001*	0.90000
SMOTE	0.77998	-	0.001*	0.90000
Under	0.001*	0.001*	-	0.001*
5050	0.90000	0.90000	0.001*	-

The Neymyi results for the LightGBM, as seen in Table 16, imply three things: First, there is a significant difference between SMOTE and undersampling ($p < 0.01$). Second, the F1-results for SMOTE and undersampling vary significantly ($p < 0.01$). Third, 5050 and SMOTE do not differ significantly ($p > 0.01$).

This leads to the following conclusion. Table 13 shows that 5050 produced the highest average F1. With this in mind, and the fact that 5050 and SMOTE do not differ significantly, both rebalancing strategies can be considered for use with the LightGBM classifier.

Table 16: Nemenyi test results for LightGBM. The asterisk denotes the combinations which were found to be significantly different ($p < 0.01$).

	None	SMOTE	Under	5050
None	-	0.001000*	0.002623*	0.014361
SMOTE	0.001000*	-	0.001000*	0.438945
Under	0.002623*	0.001000*	-	0.001000*
5050	0.014361	0.438945	0.001000*	-

6.2.2 Statistical Comparison of Classification algorithms

When keeping the rebalancing strategy and dataset constant, we can observe that the performance disparities across the different classification algorithms are less pronounced, yet still present. In virtually all configurations, Random Forests performs significantly worse than XGBoost and LightGBM. This is a interesting finding considering the fact that Random Forests is often shown to be the best classifier in RTR [56], [63]. Depending on the specific dataset and rebalancing strategy, XGBoost performs better at times, while LightGBM performs better at others.

We have examined whether these differences were statistically significant, using the Friedman test and with posthoc Nemenyi test. However, in contrast to the previous section, where we examined the overall performance of the rebalancing strategies based on F1, we now concentrate on identifying the statistically best classifier specific for each of the two proposed scenarios. That is, we will look for the classifier with the best F0.5 performance as well as the classifier with the best F2 performance.

For identifying the best F0.5 scoring classifier, we considered the no rebalancing strategy. The reason for this was that the no rebalancing strategy (in combination with XGBoost) produced the highest mean F0.5-score, as was shown in Table 13. As a result, we performed the Friedman test on the F0.5-scores for the No rebalancing strategy. This rendered a Chi-square of 23.41 which was significant ($p < 0.01$). The results of the Nemenyi test are shown in Table 17. The results suggest, when configuring a model for the trace maintenance system that the performance of XGBoost is significantly different than that of Random forests and LightGBM.

Table 17: Nemenyi test results for none rebalancing strategy on F0.5. The asterisk denotes the combinations which were found to be significantly different ($p < 0.01$).

	Random Forests	XGBoost	LightGBM
Random Forests	-	0.001*	0.900
XGBoost	0.001*	-	0.001*
LightGBM	0.900	0.001*	-

In order to find best F2 scoring classifier, we evaluated the 5050-rebalancing strategy. This choice was motivated by the fact that the 5050 rebalancing strategy yielded the highest mean F2-score, as was shown in Table 13. On that account, we ran the Friedman test on the F2-scores for the 5050 rebalancing strategy. This rendered a Chi-square of 39.47 which was significant ($p < 0.01$). The results of the Nemenyi test are shown in Table 18. They indicate that Random Forests significantly differs from the Boosted Decision Trees algorithm. When opting to use the Boosted Decision Trees algorithm, it does not matter which framework you choose. XGBoost and LightGBM are not significantly different.

Table 18: Nemenyi test results for 5050 rebalancing strategy on F2. The asterisk denotes the combinations which were found to be significantly different (p < 0.01).

	Random Forests	XGBoost	LightGBM
Random Forests	-	0.00100*	0.00100*
XGBoost	0.001*	-	0.848105
LightGBM	0.001*	0.848105	-

6.2 Results of Normalising the baseline configuration

As explained in Section 5.4, we evaluate the effect of Min-Max [0,1] scaling to the values of our features. The mean precision, recall, and F-metrics for the normalized dataset are shown in Figure 23 and the mean F-metrics are shown in Table 19. These findings still show that using various rebalancing techniques makes the most significant difference in performance. The effect of normalizing the data before training, on the other hand, appears to be negligible. However, one distinction can be observed. When non-normalized, a combination of 5050 rebalancing and LightGBM produces the best F2 scores in all three datasets. However, when we normalize the data, we see that the performance of the Project 4 project suffers substantially.

To test the significance between these two methods, we have formulated the following hypotheses:

H0: Min-Max [0,1] normalizing does not change the F1 of the classification models (alpha=0.05)
H1: Min-Max [0,1] normalizing does change the F1 of the classification models (alpha=0.05)

For each of the 12 configurations we did a Mann-Whitney U test between the non-normalized F1 results with the normalized results for each of the 12 configurations. The U statistics (n=30) together with the P-values are given in Table 20. We find that normalisation has no significant effect on the F1-scores of the classification models, and therefore we fail to reject the null-hypothesis. However, it is worth noting that the configurations XGBoost + 5050 (p = .07), LightGBM + SMOTE (p = .02), and LightGBM + 5050 (p = .07 nearly passed the significance level.

Table 19: Mean F0.5-measure and F2-measure for the different normalized configurations

Algorithm / Rebalancing	Random Forests		XGBoost		LightGBM	
	F0.5	**F2**	**F0.5**	**F2**	**F0.5**	**F2**
No Rebalancing	65.91	27.79	78.40	48.20	61.93	42.18
SMOTE	70.28	43.70	73.53	52.49	65.28	50.68
Undersampling	11.73	23.58	12.77	24.49	13.25	25.06
5050	63.83	47.64	61.13	52.80	59.98	52.02

Table 20: The Mann-Whitney U statistics for the different configurations

Algorithm / Rebalancing	Random Forests	XGBoost	LightGBM
None	395.0 (p = .21)	390 (p = .19)	401 (p = .24)
SMOTE	417.5 (p = .32)	443.0 (p = .46)	304.5 (p = .02)
Under	420 (P = .33)	442 (p = .46)	441.0 (p = .45)
5050	397.5 (p = .22)	348 (p = .07)	350.0 (p = .07)

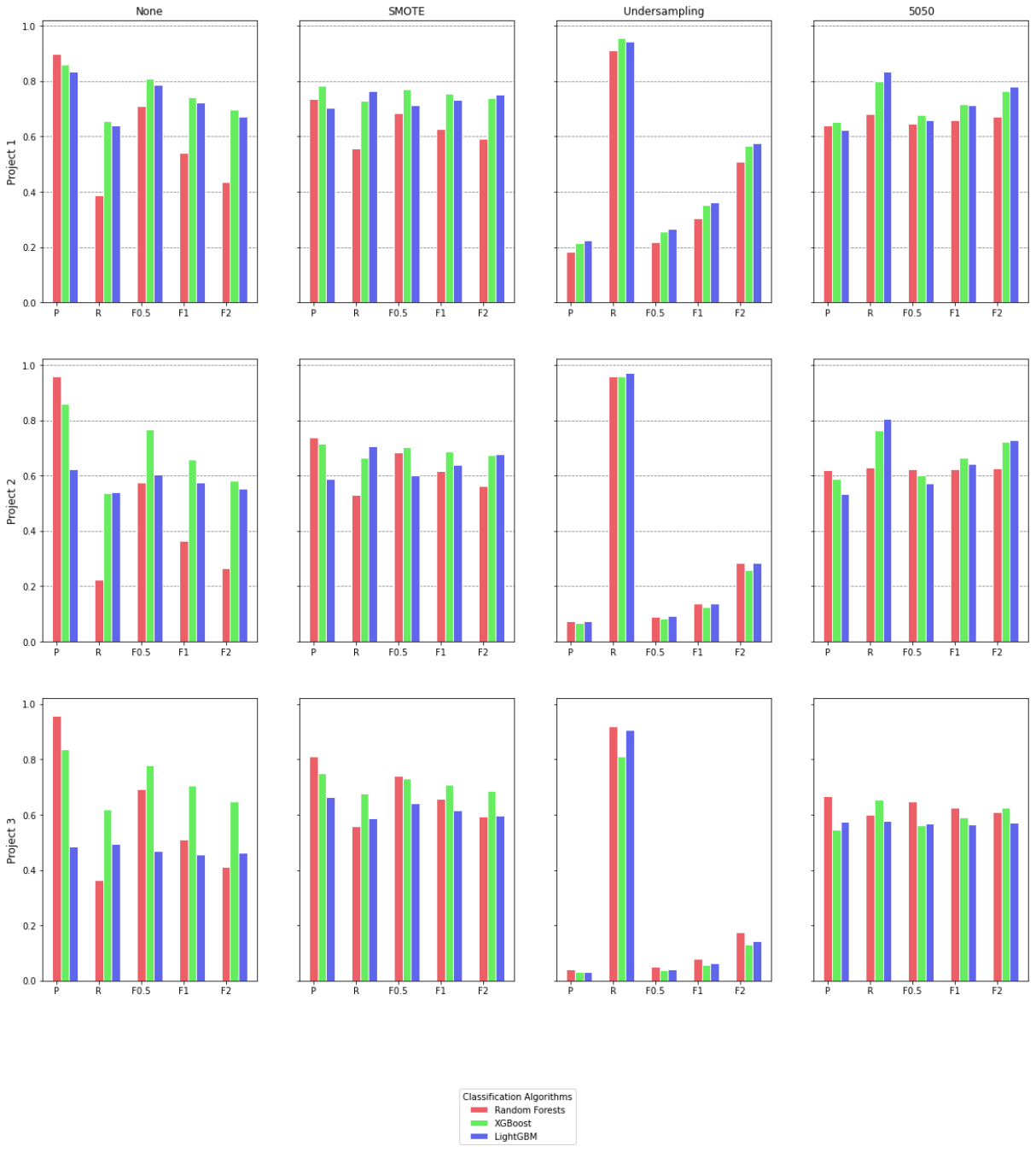


Figure 23: Mean precision, recall, F0.5, F1, and F2 metrics for the different model configurations using Min-Max [0,1] normalized data

6.3 Best Model for Trace Recommendation Scenario.

From our baseline results, we have chosen the best model fit (i.e., highest mean F2-score) for the trace recommendation scenario. This was a configuration with LightGBM as a classifier and 5050 as a rebalancing strategy, and no Min-Max [0,1] normalisation in the pre-processing, which

averaged (n = 30) an F2-score of 73.48 (SD = 6.64). In this section, we further explore this configuration of the model by doing another 25 runs to minimise sampling error. These results are then used to explain the most important features. Finally, a discussion follows about whether the model can be further improved by hyperparameter tuning.

6.3.1 Best Features of Trace recommendation features

In this Section we present the features which are deemed the most important. We report this in terms of total gain [80], which was the default metric of feature importance in LightGBM. Of each run we have logged the total gain of every individual feature and have averaged it. These averages were then used to find the top 5 most important features for each individual dataset. The results are shown in the boxplots of Figure 24.

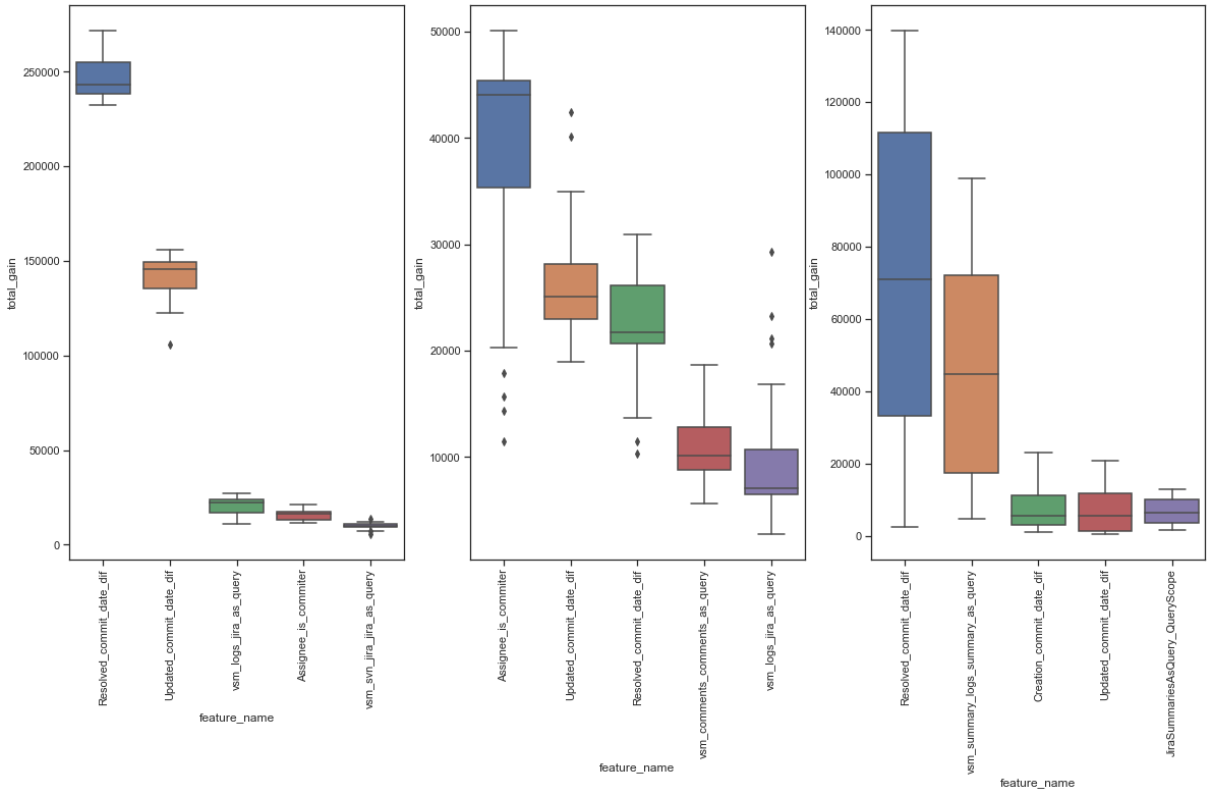


Figure 24: Total gain for the 5 most important features for the Project 3 (left), Project 2 (middle), and Project 4 (right) datasets

These results reveal a number of interesting findings. The first finding is that the process-related feature family provides a lot of information on whether a JIRA issue and a commit should be traced together. Both the *Resolved_commit_date_dif* and *Updated_commit_date_dif* features are among the top five most significant features in all three datasets. This suggests that the time between committing and altering the status of a JIRA issue is a significant indicator of whether these should be traced. The last process-related feature *assignee_is_commiter* is likewise a significant indicator for trace validity, ranking in the top five in two of the three datasets.

The second finding is that the TF-IDF representation of the trace artefact appears to be important. We mentioned in Section 5.3.3 that we examined various TF-IDF representations. For example,

one approach is to represent a commit as a whole by taking into account all natural language, such as logs and unit names. The second approach is to express the commit as individual subsets and represent the commit logs separately from the unit names. Most research [38], [56], [64] mainly consider the first approach. However, our findings indicate that the second approach would be more beneficial to the classifier. Figure 23 indicates that the cosine similarity between a commit log and a JIRA problem, *Vsm_logs_jira_as_query*, is the third most important feature for evaluating the authenticity of a trace, with a mean total gain of 20385.58 (SD = 4546.96). This is significantly higher than the cosine similarity between the whole commit and the JIRA issue, *vsm_svn_jira_jira_as_query*, which has a mean total gain of 9939.16 (SD = 1744.80). A similar argument can be made for additional top five features, such as *vsm_comments_comments_as_query* or *jira_summaries_as_query_queryscope*. As a result, we may argue that special attention should be paid to how we represent the trace artefacts.

The last finding is that the top five most important features differ between the three datasets. Since the datasets originate from different projects, the data might have been developed by different team composition. The data characteristics are likely to alter depending on the team makeup, resulting in various features being considered important by the model. As a consequence, provided resources and sufficient training data are available, we may propose training the model again for each new project, resulting in a project-finetuned model rather than a generic trained model.

6.3.2 The Results of Hyperparameter Tuning

To produce the baseline results, we have used the default settings of the classification algorithms. We wanted to examine whether these results could be further improved by optimizing the hyperparameters of the model, by means of a Randomized Search on the hyperparameters, for which the Scikit-Learn library was used.

For this experiment, we considered five hyperparameters:

1) Num_leaves: This controls the maximum number of leaves in one tree, and acts as the main parameter for determining the complexity of the model
2) Min_data_in_leaf: The minimum number of data points required in a leaf, in order for the leaf to be added to the tree.
3) Max_depth: The maximum depth a tree can have in the model.
4) Learning_rate: The shrinkage rate used in update
5) Max_bin: The maximum number of bins for each feature. A larger value can improve accuracy at the costs of training speed.

For each of the five hyperparameters we have specified a set of ten values. These values are shown in Table 21. A total of 100 samples were drawn from the search space. To produce the results, we once again divided the data into a train and test set using an 80:20 stratified split. We applied the Random Search Algorithm on a stratified 10-fold split. Both the best cross-validation F2-score and test F2-score were recorded. We have repeated this procedure 25 times, of which the results are shown in Figure 25.

Table 21: The considered search space for the Randomized Search on the LightGBM model

Hyperparameter	Value sets *indicates the default setting
Num Leaves	{11, 16, 21, 26, 31, 36, 41, 46, 51, 56}
Min_data_in_leaf	{5, 10, 15, 20*, 25, 30, 35, 40, 45, 50}
Max_depth	{-1, 100, 200, 300, 400, 500, 600, 700, 800, 900}
Learning_rate	{0.1*, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.9, 1.0}
Max_bin	{50, 100, 150, 200, 255*, 300, 350, 400, 450, 500}

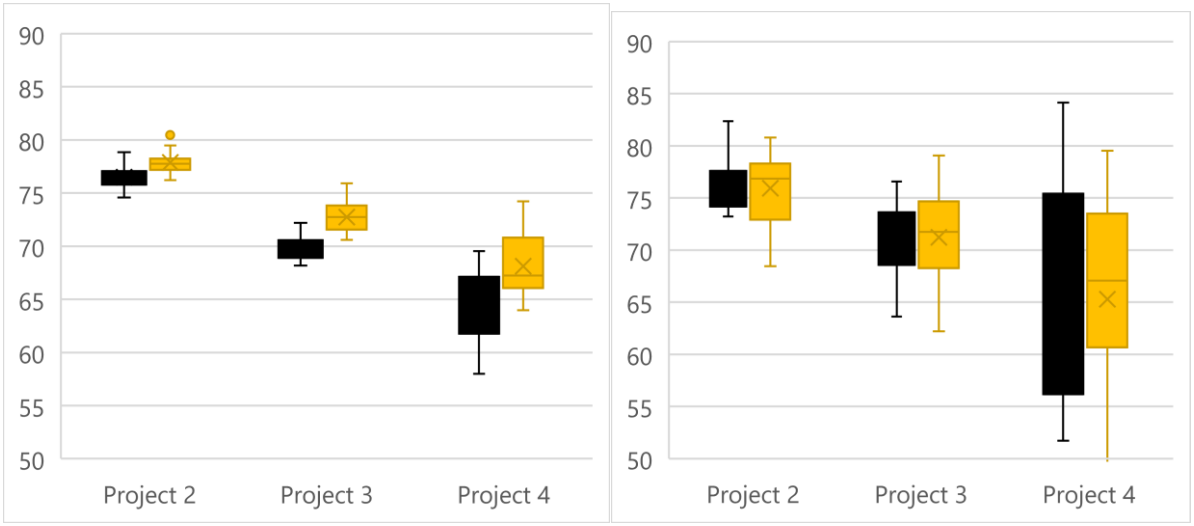


Figure 25: Comparison of F2-scores between default and hyperparameter tuned LightGBM for Cross-Validation (left) and Test (right)

We observe that the F2 scores on cross-validation were higher in the Hyperparameter tuned group than the default group. A Mann-Whitney U test showed significantly different results, U ($N_{Default}$=75, $N_{Hyperparameter\ tuned}$ = 75) = 1897, p < .01. However, the most interesting is of course the F2-scores on the test set. These turned out not to be significant, U ($N_{Default}$=75, $N_{Hyperparameter\ tuned}$ = 75) = 2787, p = 0.46. This possibly indicates that hyperparameter tuning leads to an overfit on the training data but does not necessarily impact the generalizability of the model.

6.4 Best Model for Trace Maintenance Scenario

The best model fit, in terms of mean F0.5, for the trace maintenance scenario, was a configuration with XGBoost as classifier, no rebalancing strategy applied, and a Min-Max [0,1] normalisation applied to the data. We will now discuss the model in terms of feature importance, as we did with the best trace recommendation model. This will be followed by the results of the hyperparameter tuning experiment.

6.4.1 Top 5 most important features

In this section we present the features which are deemed the most important for the scenario of trace maintenance. We used the same method for assessing feature importance as we did for assessing feature importance in the trace recommendation scenario. In contrast to that scenario, we now express feature importance in terms of average gain rather than total gain. This is because the default feature significance metric in XGBoost is average gain rather than total gain. It makes no difference because these metrics are always used to evaluate the relative importance of features within the same model, not between models. The top five most important features is shown in Figure 26.

A number of observations can be made from this. Once again, the project-related feature family is well-represented among the top five most important features. Remarkable, however, is the fact that none of them are present in the Project 4.

Furthermore, as compared to the model for trace recommendation, the query quality metrics are more prominent in the top 5. The most significant feature in the Project 2 dataset is *JiraSummariesAsQuery maxEntropy*, and the Project 4 datasets its top 3 entirely consists of query quality metrics.

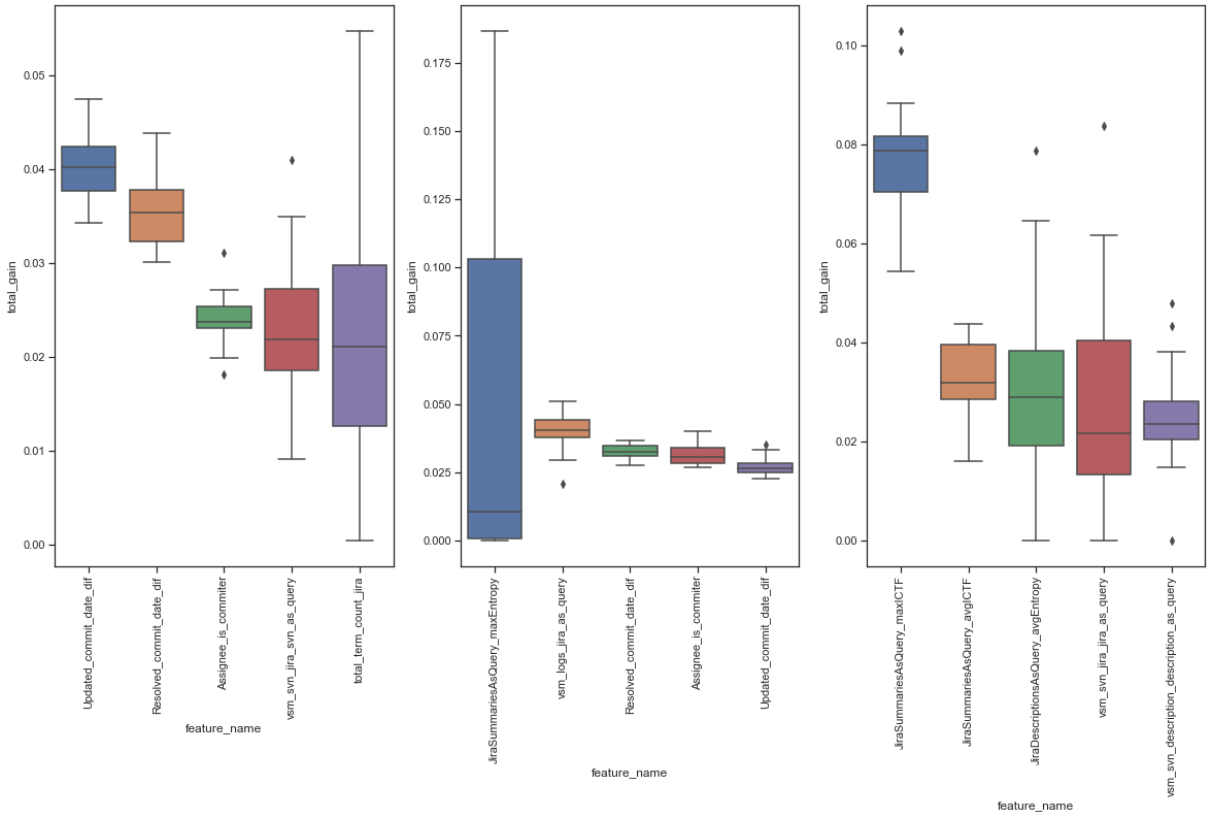


Figure 26: Average gain for the 5 most important features for the Project 3 (left), Project 2 (middle), and Project 4 (right) datasets

6.4.2 The Results of Hyperparameter Tuning

For hyperparameter tuning the best model for the trace maintenance scenario, we applied the same strategy as for the trace recommendation scenario. To the search space for the randomized search belonged five parameters.

1) Learning Rate: Identical to the learning rate parameter in the LightGBM Framework
2) Max Depth: Identical to the max_depth parameter in the LightGBM Framework
3) Min_child_weight: The minimum sum of instance weight required in a child.
4) Gamma: Minimum loss reduction required to make a further partition on a leaf node of the tree.
5) Colsample_byTree: The proportion of parameters utilized for training each tree

For each of the five parameters, 10 values were considered, which are shown in Table 22.

The results for the hyperparameter tuning are produced in an identical way as with the recommendation system. The results are shown in Figure 27. Our findings reaffirm the results of hyperparameter tuning for the trace recommendation scenario. The cross-validation F0.5-scores differ significantly, U ($N_{Default}$=75, $N_{Hyperparameter\ tuned}$ = 75) = 1106, p< .01, while the test F0.5-scores did not,U($N_{Default}$=75, $N_{Hyperparameter\ tuned}$ = 75) = 2500, p = 0.12).

Table 22:The considered search space for the Randomized Search on the XGBoost model

Hyperparameter	Range (Default value indicated by *)
Learning_rate	{0.15, 0.20, 0.25, 0.30*, 0.35, 0.40, 0.45, 0.50, 0.55, 0.60}
Max_depth	{2, 4, 6*, 8, 10, 12, 14, 16, 18, 20}
Min_child_weight	{1*, 2, 3, 4, 5, 6, 7, 8, 9, 10}
gamma	{0.0*, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9}
Colsample_bytree	{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0*}

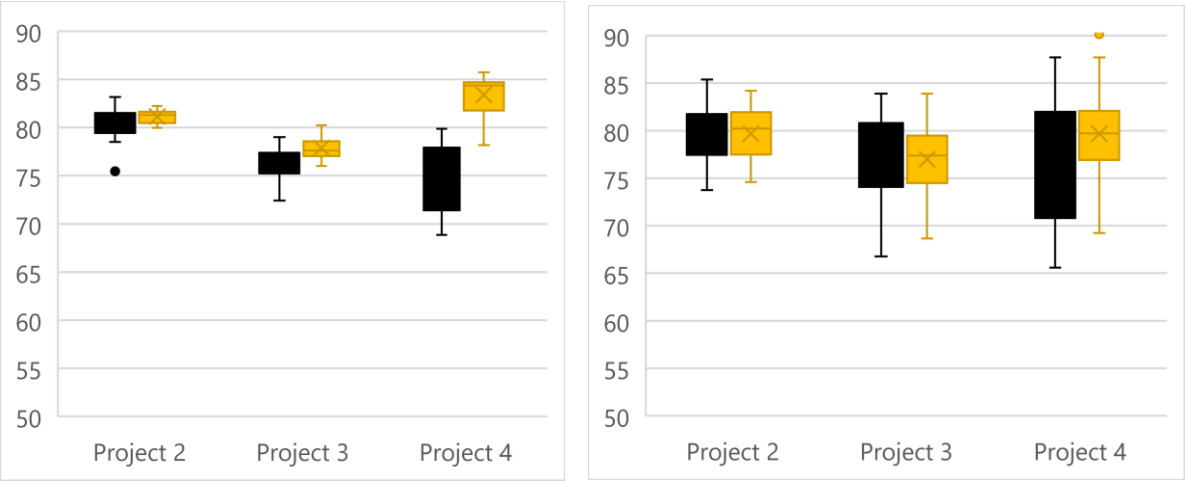


Figure 27: Comparison of F0.5-scores between default and hyperparameter tuned XGBoost for Cross-Validation (left) and Test (right)

7. Discussion

In this chapter, we will discuss the findings of our study. First, we will present the main contributions together with their implications. Then, the limitations of our method, together with their implications are discussed. Finally, all the threats to the validity are discussed, together with our strategy of mitigation.

7.1 Main Contributions

The main contribution of this study lies in the acquisition of new insights on the specific context of RE in MDD. These insights were then used to create a treatment specially tailored to the needs of this context. As a result, the gap between academic research and industrial demands has been narrowed, bringing us closer to the vision of ubiquitous requirements traceability [31]. This type of research adheres to the idea of context-driven research. finternalThis type of research is essential, because the applicability and scalability of software engineering solutions heavily affected by the contextual, organizational, and domain-related factors.

Furthermore, we contribute by providing insights on using Gradient Boosted Trees for RTR. We have presented data on how this class of algorithms (i.e., LightGBM or XGBoost) performed and how they compared to the Random Forests algorithm. This is, to the best of our knowledge, the first study to provide such an empirical comparison.

Additionally, we demonstrated that the representation of a trace artefact is important to the classifier. In our results, we found that features that used TF-IDF representation of a subset of a trace artefact (e.g., summary, description) were more prominent in the top important features.

Finally, this research contributes by advising which specific rebalancing technique belongs to which scenario. Mills et al. [38] already have shown that different rebalancing strategies yield different results. However, they were primarily interested in finding the most optimal balance between precision and recall, so results are more generalizable. In our research, we specifically looked at which rebalancing strategy was best for a trace recommendation and which rebalancing strategy was best for trace maintenance.

7.2 Threats to validity

The threats to the validity of this research are discussed using the four aspects of validity [23]. These consist of the Construct Validity, Internal Validity, External Validity, and Reliability. For each aspect, the threats and the strategy to overcome them are discussed in the following sections.

7.2.1 Construct Validity

The extent to which the operational constructs reflect the theoretical structures is referred to as construct validity. Several steps have been made to mitigate the threats. First, because all study was done by a single researcher, there was no possibility of misunderstanding of the data. Second, when available, opensource libraries, such as Scikit-Learn, have been used to operationalize the constructs.

However, it should be emphasized that not everything can be mitigated. In particular, there are threat to the construct validity of the query quality metrics [63]. No open-source Python library

was found to compute these metrics. As a result, these computations were coded by the researchers themselves. This poses threats in two ways. First, personal interpretations of the theory may result in operational constructs, which did not sufficiently reflect the theoretical constructs. Second, there might have been programming mistakes that influenced the results.

Another threat stems from the origin of our labelled data. For the labelled data we made use of the manually produced traces by the Mendix engineers. As we discussed previously in Chapter 1, this type of RT is prone to errors. Because we have not validated if the traces we created were legitimate, a garbage-in-garbage-out scenario is a possibility.

7.2.2 Internal Validity

The internal validity relates to the level to which the claims made in research are not caused by an unanticipated third factor. We attempted to mitigate threats by conducting a literature study and conducting semi-structured interviews to ensure that all relevant factors were identified. Despite these efforts mitigate the threats to some extent, two of those factors continue to pose a threat to the internal validity.

The first threat is related to how we compared our models to one another. We obtained three datasets for our research. Two datasets contained JIRA comment data, whereas one did not. despite the fact they are not directly comparable, we evaluated all three datasets in a comparison. We were aware of the discrepancy between them and have given great consideration to the claims we made based on this comparison.

The second threat relates to the data quality used for the study. The studied organisation has imposed a number of quality standards that its JIRA issues and commits must meet. To that extent, it is plausible that this level of data quality is required for the models to perform successfully. As a result, it is unclear if the model works effectively with less comprehensive data.

7.2.3 External Validity

The external validity is concerned with how generalizable the study results are and how valuable they are to individuals outside of the study. We did our best to mitigate this evaluating the treatment on datasets from three distinct projects. Furthermore, these projects were obtained from 2 separate teams, each with its unique set of procedures and practices. Finally, to minimize overfitting and enhance generalizability, we followed the standard practice of having a distinct training and test set.

Despite our best efforts to mitigate the threats, not everything can be accounted for. All the results were obtained from a single organisation. As a result, the external validity is threatened since the results are biased towards the examined organisation. This organization may have had unique practices that were not seen elsewhere. Consequently, we had to be cautious in how we expressed our conclusions.

7.2.4 Reliability

The degree to which the results are dependent on the researcher who conducted the research is referred to as the aspect of reliability. When another researcher does the same study, the outcomes should be the same. To accomplish this we followed the guidelines of peer-reviewed

methods by Wieringa [21], Kitchenham [22], and Longhurst [25]. Furthermore, the exact operationalisation of these methods was documented into great detail, and we have tried to be as unambiguous as possible, for other researchers to replicate. In addition, all relevant documents deemed useful (e.g., interview protocol, Jupyter Notebooks) are included in the Appendix. Finally, the raw results are accessible in the online Appendix.

8. Conclusion and Future Work

This research studied how requirements traceability can be improved in a model-driven development environment. This was done by studying relevant literature, conducting semi-structured interviews at Mendix, designing a trace link classifier for the MDD environment, and evaluating its performance.

This chapter concludes the research by answering the research questions formulated in Chapter 1. Furthermore, we outline directions for future research.

8.1 Sub Questions

Before answering the main research question, we will first go through the process of answering the sub questions.

SQ1 What is the state-of-the-art in Requirements Traceability?

The goal of this sub question was to get familiar with RT fundamentals, and to understand the present challenges identified by the research community. This was answered by reviewing the literature in a semi-systematic way.

There are several study areas in requirements traceability research. Several scholars believe the area that requires the most attention is that of automatically recovering trace links between requirements and some other artefact, which is known as the process of automatic trace recovery.

This works as follows: given are two trace artefacts, there is either a trace link between them or there is not. An algorithm is then tasked with automatically determining whether there is a trace link between them.

SQ2 What algorithms are needed to automatically trace artefacts?

The goal of this sub-question was to identify which algorithms are used for automatic trace recovery. This question was also answered by reviewing the literature.

Algorithms used for automatic trace recovery can be categorized into four orthogonal categories: information retrieval, heuristic, machine learning and deep learning [17]. Researchers believe that Machine Learning and Deep learning approaches belong to the state-to-the-art for establishing the trace links [17]. As a results, new RT tools should concentrate on incorporating these technologies.

In this study, we have concentrated on the algorithms from the machine learning paradigm. The performance of several machine learning classifying algorithms in RTR tasks have been researched, with Random Forests outperforming the others.

However, the Boosted Decision Trees algorithm was not included in these studies' comparisons. Studies in other domains demonstrated that this algorithm outperformed Random Forests [75], [76], therefore we evaluated it for the RTR task as well.

SQ3 How do MDD artefacts and requirements co-evolve in an MDD company?

The aim of this question was to get familiar with the MDD context, in which our treatment would operate in. At Mendix, semi-structured interviews were used to address this question.

According to the findings of these interviews, developers produce software to the principles of the Agile Manifesto and releasing software in sprints. During a sprint, developers implement a set of requirements, stored in JIRA issues, in their software. Mendix Studio, their IDE, is used for this task. Changes to the software are made locally with Mendix Studio. Each modification to the model is recorded in a commit, which is later synced with the Mendix Team server, saving all commits and model data.

SQ4 How to embed automatic tracing algorithms in a RT tool for the MDD domain?

Developers at Mendix manually trace their commits to JIRA issues by noting the associated JIRA issue ID in the commit log. The current situation of working with two separate systems (i.e. JIRA and Mendix Studio) is not optimal. This problem would be improved by embedding a trace system in Mendix Studio.

This embedding might support one of two envisioned scenarios. In the first scenario, the embedded system should recommend traces to the developer whenever he or she wants to commit their changes to the Team servers. In the second scenario, the system should serve as a trace maintenance tool. Its purpose is to recover traces between commits and JIRA issues in existing projects for commits which are now untraceable.

SQ5 What are the resources available to design and construct a RT tool for the MDD domain?

All commit data is stored on the Mendix Team Server, which we were able to obtain. Because developers include JIRA issue IDs in the log message, this data could be used to train a ML classifier. Corresponding commit data and JIRA data were obtained from four distinct projects.

This data is used to develop and build an RT tool using a combination of opensource Python libraries.

SQ6 How do we validate the effectiveness of a RT tool for the MDD domain?

When validating the performance of an automatic RTR tool, we must first formulate what we are most interested in. In the scenarios we have envisioned, this is the classification of valid traces (true positives). The two measures used to assess this are precision and recall. For this reason, both metrics belong to the most popular metrics used when validating RT tools.

Furthermore, it is important not to look at both metrics individually, but rather to look at the most ideal balance. The F-measure, which has several variants, is the metric used to quantify this balance. Depending on the scenario, a different variant must be considered. For the trace recommendation system, the F2-measure is important, while for the fully automatic trace maintainer the F0.5-measure is important.

8.2 Main Research Question

Now that all sub questions are answered, we can provide an answer to the Main Research Question:

MQ	How to automate tracing between requirements and models in a Model Driven Environment?

Before starting with automating, you first need to get clear what scenario the automatic tracer is used for. For the MDD environment that we studied, we identified 2 scenarios which are most beneficial. The first scenario recommends trace links between JIRA issues and SVN commits to the developer. In the second scenario, a tool maintains a project by recovering trace links between JIRA issues and SVN commits in a fully automatic way.

This was achieved by first producing the cartesian product between the JIRA issue set and the SVN commits set is created. Each element of this result is a candidate trace. Each trace is then represented as features of 4 categories: process-related, document statistics, information-retrieval, and query quality.

Because we are producing a Cartesian product, we are creating a highly imbalanced dataset. Depending on the scenario, it can be beneficial to rebalance the data. Once this is done, we can train a ML classifier to identify which of the candidate traces are valid.

In this study, we have successfully constructed a prototype of the design using data acquired from Mendix. We were able to get a mean F0.5-score of 77.32 employing XGBoost as the ML classifier, with no rebalancing on the training data, and Min-Max normalization [0,1] on the training data, when we configured our classifier for the scenario of Full automated maintenance.

When we configure our classifier for the scenario of trace recommendation, we get an F2-score of 73.48 by using the LightGBM ML classifier, the 5050-rebalancing strategy, and no Min-Max normalizing on our training data.

8.3 Future Work

This research has shown how traces between JIRA issues and MDD commits can be automatically recovered. Although the general concept is explored and applied, certain aspects can be studied in greater depth.

First, we have developed various IR-features. These features rely on the TF-IDF vector representation. For the TF-IDF calculations, we have considered multiple corpora. For representing a JIRA issue: entire JIRA issue, JIRA summary, JIRA comment, and JIRA description. For representing a commit issue, we utilized entire commit, commit log, and commit unit names. When producing the results, we have utilized all features and let the model determine which feature is most usable. A future study should investigate what vector representation provides the best results.

Furthermore, another aspect of the feature engineering process can be examined. To the family of query quality metrics, we have examined a total of 3 categories: specificity, similarity, term relatedness. However, Mills et al. [63] also identified metrics of the 'coherency' category. We have briefly considered implementing metrics of this category; however, it was infeasible due to limitations in time. A follow-up study could include this category into the feature set and evaluate its impact.

Additionally, more research can be done about which features to include in the model. For our study, we have considered all engineered features. However, during the feature engineering process it was noted that, on occasions, introducing new features degraded performance. One cause might be that a higher number of features will overfit the data. This is especially true for the query quality metrics, which comprise the vast majority of our feature set. Using feature selection methods to remove noise and reduce model complexity may enhance performance. Future research could investigate which strategies are most suited for the MDD domain.

Another direction for further research is to the use of Gradient Boosted Trees in RTR. We demonstrated that these algorithms outperform Random Forests in a MDD domain. Is this sort of algorithm particularly suited to RTR problems, or do they possess qualities that other algorithms do not? To assess this, we must run this algorithm on datasets accessible from CoEST that are frequently used in RT research [82]. This allows us to directly compare the performance to prior research.

Finally, we limited the scope of this research to trace JIRA issues to commits. It is a great first step, but further research needs to be done to see if it feasible to trace requirements directly to a model unit. These findings would enable new scenarios. For instance, the model could warn the developer whenever he she creates a model unit, which is not yet documented in JIRA. In addition, JIRA problems and version control systems are not limited to the MDD domain. Future studies can determine whether similar findings apply to other fields.

Bibliography

[1] B. H. C. Cheng and J. M. Atlee, "Research Directions in Requirements Engineering," in *Future of Software Engineering (FOSE '07)*, May 2007, no. c, pp. 285–303, doi: 10.1109/FOSE.2007.17.

[2] E. Hull, K. Jackson, and J. Dick, *Requirements Engineering*. London: Springer London, 2011.

[3] S. Wagner, D. M. Fernández, M. Felderer, and M. Kalinowski, "Requirements Engineering Practice and Problems in Agile Projects: Results from an International Survey," *CIbSE 2017 - XX Ibero-American Conf. Softw. Eng.*, pp. 85–98, Mar. 2017, doi: 10.7287/peerj.preprints.2038.

[4] O. C. Z. Gotel and C. W. Finkelstein, "An analysis of the requirements traceability problem," in *Proceedings of IEEE International Conference on Requirements Engineering*, Apr. 1994, pp. 94–101, doi: 10.1109/ICRE.1994.292398.

[5] F. Blaauboer, K. Sikkel, and M. N. Aydin, "Deciding to Adopt Requirements Traceability in Practice," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 4495 LNCS, 2007, pp. 294–308.

[6] J. Cleland-Huang, B. Berenbach, S. Clark, R. Settimi, and E. Romanova, "Best Practices for Automated Traceability," *Computer (Long. Beach. Calif).*, vol. 40, no. 6, pp. 27–35, Jun. 2007, doi: 10.1109/MC.2007.195.

[7] A. Kannenberg and H. Saiedian, "Traceability Remains a Challenge," *J. Def. Softw. Eng. CrossTalk J. Def. Softw. Eng.*, no. May, pp. 14–19, 2009, [Online]. Available: www.stsc.hill.af.mil.

[8] M. C. Panis, "Successful Deployment of Requirements Traceability in a Commercial Engineering Organization...Really," in *2010 18th IEEE International Requirements Engineering Conference*, Sep. 2010, no. January, pp. 303–307, doi: 10.1109/RE.2010.43.

[9] P. Mäder and A. Egyed, "Do developers benefit from requirements traceability when evolving and maintaining a software system?," *Empir. Softw. Eng.*, vol. 20, no. 2, pp. 413–441, Apr. 2015, doi: 10.1007/s10664-014-9314-z.

[10] P. Rempel and P. Mader, "Preventing Defects: The Impact of Requirements Traceability Completeness on Software Quality," *IEEE Trans. Softw. Eng.*, vol. 43, no. 8, pp. 777–797, Aug. 2017, doi: 10.1109/TSE.2016.2622264.

[11] S. Winkler and J. von Pilgrim, "A survey of traceability in requirements engineering and model-driven development," *Softw. Syst. Model.*, vol. 9, no. 4, pp. 529–565, Sep. 2010, doi: 10.1007/s10270-009-0145-0.

[12] M. Brambilla, J. Cabot, and M. Wimmer, "Model-Driven Software Engineering in Practice," *Synth. Lect. Softw. Eng.*, vol. 1, no. 1, pp. 1–182, Sep. 2012, doi: 10.2200/S00441ED1V01Y201208SWE001.

[13] C. Atkinson and T. Kuhne, "Model-driven development: a metamodeling foundation," *IEEE Softw.*, vol. 20, no. 5, pp. 36–41, Sep. 2003, doi: 10.1109/MS.2003.1231149.

[14] B. Wang, R. Peng, Y. Li, H. Lai, and Z. Wang, "Requirements traceability technologies and technology transfer decision support: A systematic review," *J. Syst. Softw.*, vol. 146, pp. 59–79, Dec. 2018, doi: 10.1016/j.jss.2018.09.001.

[15] P. Mäder, I. Philippow, and M. Riebisch, "Customizing traceability links for the unified process," in *International Conference on the Quality of Software Architectures*, 2007, pp. 53–71.

[16] B. Ramesh, "Factors influencing requirements traceability practice," *Commun. ACM*, vol. 41, no. 12, pp. 37–44, Dec. 1998, doi: 10.1145/290133.290147.

[17] T. W. W. Aung, H. Huo, and Y. Sui, "A Literature Review of Automatic Traceability Links Recovery for Software Change Impact Analysis," in *Proceedings of the 28th International Conference on Program Comprehension*, Jul. 2020, pp. 14–24, doi: 10.1145/3387904.3389251.

[18] R. F. Paige *et al.*, “Rigorous identification and encoding of trace-links in model-driven engineering,” *Softw. Syst. Model.*, vol. 10, no. 4, pp. 469–487, 2011, doi: 10.1007/s10270-010-0158-8.
[19] Mendix, “What is Model Driven Development (MDD)?,” 2020. https://www.mendix.com/model-driven-development/ (accessed Mar. 05, 2021).
[20] J. Ruis, “Exploring traceability between requirements and Low-Code Development design,” Utrecht University, 2020.
[21] R. J. Wieringa, *Design Science Methodology for Information Systems and Software Engineering*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014.
[22] B. Kitchenham and S. Charters, “Guidelines for performing systematic literature reviews in software engineering,” 2007.
[23] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering*, vol. 9783642290. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.
[24] Mendix, “Mendix: We Help Enterprises Achieve their Digital Goals with Low-code,” 2020. https://www.mendix.com/company/ (accessed Jan. 03, 2021).
[25] R. Longhurst, “Semi-structured Interviews and Focus Groups,” in *Key Methods in Geography*, 2003, pp. 117–132.
[26] P. Naur and B. Randell, “Report on a conference sponsored by the NATO SCIENCE COMMITTEE,” 1969.
[27] O. Gotel *et al.*, “Traceability Fundamentals,” in *Software and Systems Traceability*, London: Springer London, 2012, pp. 3–22.
[28] A. Qusef, G. Bavota, R. Oliveto, A. De Lucia, and D. Binkley, “SCOTCH: Test-to-code traceability using slicing and conceptual coupling,” in *2011 27th IEEE International Conference on Software Maintenance (ICSM)*, Sep. 2011, pp. 63–72, doi: 10.1109/ICSM.2011.6080773.
[29] S. Nair, J. L. de la Vara, and S. Sen, “A review of traceability research at the requirements engineering conferencere@21,” in *2013 21st IEEE International Requirements Engineering Conference (RE)*, Jul. 2013, pp. 222–229, doi: 10.1109/RE.2013.6636722.
[30] G. Antoniol, J. Cleland-Huang, J. H. Hayes, and M. Vierhauser, “Grand Challenges of Traceability: The Next Ten Years,” *CoRR*, vol. abs/1710.0, Oct. 2017, [Online]. Available: http://arxiv.org/abs/1710.03129.
[31] O. Gotel *et al.*, “The Grand Challenge of Traceability (v1.0),” in *Software and Systems Traceability*, J. Cleland-Huang, O. Gotel, and A. Zisman, Eds. London: Springer London, 2012, pp. 343–409.
[32] J. Cleland-Huang, O. C. Z. Gotel, J. Huffman Hayes, P. Mäder, and A. Zisman, “Software traceability: trends and future directions,” in *Future of Software Engineering Proceedings*, May 2014, pp. 55–69, doi: 10.1145/2593882.2593891.
[33] F. A. C. Pinheiro, “Requirements Traceability,” in *Perspectives on Software Requirements*, Boston, MA: Springer US, 2004, pp. 91–113.
[34] I. Galvao and A. Goknil, “Survey of Traceability Approaches in Model-Driven Engineering,” in *11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007)*, Oct. 2007, pp. 313–313, doi: 10.1109/EDOC.2007.4384003.
[35] J. Holtmann, J.-P. Steghofer, M. Rath, and D. Schmelter, “Cutting through the Jungle: Disambiguating Model-based Traceability Terminology,” in *2020 IEEE 28th International Requirements Engineering Conference (RE)*, Aug. 2020, vol. 2020-Augus, pp. 8–19, doi: 10.1109/RE48521.2020.00014.
[36] Object Management Group, “Model Driven Architecture (MDA) | Object Management Group.” https://www.omg.org/mda/ (accessed Dec. 15, 2020).
[37] S. Sendall and W. Kozaczynski, “Model transformation: the heart and soul of model-driven

software development," *IEEE Softw.*, vol. 20, no. 5, pp. 42–45, Sep. 2003, doi: 10.1109/MS.2003.1231150.

[38] C. Mills, J. Escobar-Avila, and S. Haiduc, "Automatic Traceability Maintenance via Machine Learning Classification," *2018 IEEE Int. Conf. Softw. Maint. Evol.*, pp. 369–380, Jul. 2018, doi: 10.1109/ICSME.2018.00045.

[39] A. Ghannem, M. S. Hamdi, M. Kessentini, and H. H. Ammar, "Search-based requirements traceability recovery: A multi-objective approach," in *2017 IEEE Congress on Evolutionary Computation (CEC)*, Jun. 2017, pp. 1183–1190, doi: 10.1109/CEC.2017.7969440.

[40] Y. Shin, J. H. Hayes, and J. Cleland-Huang, "Guidelines for Benchmarking Automated Software Traceability Techniques," in *Proceedings of the 8th International Symposium on Software and Systems Traceability*, 2015, pp. 61–67.

[41] G. G. Chowdhury, *Introduction to modern information retrieval*. Facet publishing, 2010.

[42] N. Ali, H. Cai, A. Hamou-Lhadj, and J. Hassine, "Exploiting Parts-of-Speech for effective automated requirements traceability," *Inf. Softw. Technol.*, vol. 106, pp. 126–141, Feb. 2019, doi: 10.1016/j.infsof.2018.09.009.

[43] M. Borg, P. Runeson, and A. Ardö, "Recovering from a decade: a systematic mapping of information retrieval approaches to software traceability," *Empir. Softw. Eng.*, vol. 19, no. 6, pp. 1565–1616, Dec. 2014, doi: 10.1007/s10664-013-9255-y.

[44] B. Wang, R. Peng, Z. Wang, X. Wang, and Y. Li, "An Automated Hybrid Approach for Generating Requirements Trace Links," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 30, no. 07, pp. 1005–1048, Jul. 2020, doi: 10.1142/S0218194020500278.

[45] R. Oliveto, M. Gethers, D. Poshyvanyk, and A. De Lucia, "On the Equivalence of Information Retrieval Methods for Automated Traceability Link Recovery," in *2010 IEEE 18th International Conference on Program Comprehension*, Jun. 2010, pp. 68–71, doi: 10.1109/ICPC.2010.20.

[46] P. D. Turney and P. Pantel, "From Frequency to Meaning: Vector Space Models of Semantics," *J. Artif. Intell. Res.*, vol. 37, pp. 141–188, Mar. 2010, doi: 10.1613/jair.2934.

[47] C. D. Manning, P. Raghavan, and H. Schutze, "Scoring, term weighting, and the vector space model," in *Introduction to Information Retrieval*, no. c, Cambridge: Cambridge University Press, 2008, pp. 100–123.

[48] Google, "Embeddings: Translating to a Lower-Dimensional Space," 2020. https://developers.google.com/machine-learning/crash-course/embeddings/translating-to-a-lower-dimensional-space (accessed Mar. 08, 2021).

[49] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, "Indexing by latent semantic analysis," *J. Am. Soc. Inf. Sci.*, vol. 41, no. 6, pp. 391–407, Sep. 1990, doi: 10.1002/(SICI)1097-4571(199009)41:6<391::AID-ASI1>3.0.CO;2-9.

[50] G. Capobianco, A. De Lucia, R. Oliveto, A. Panichella, and S. Panichella, "On the role of the nouns in IR-based traceability recovery," in *2009 IEEE 17th International Conference on Program Comprehension*, May 2009, pp. 148–157, doi: 10.1109/ICPC.2009.5090038.

[51] A. Abadi, M. Nisenson, and Y. Simionovici, "A Traceability Technique for Specifications," in *2008 16th IEEE International Conference on Program Comprehension*, Jun. 2008, pp. 103–112, doi: 10.1109/ICPC.2008.30.

[52] A. Panichella, B. Dit, R. Oliveto, M. Di Penta, D. Poshynanyk, and A. De Lucia, "How to effectively use topic models for software engineering tasks? An approach based on Genetic Algorithms," in *2013 35th International Conference on Software Engineering (ICSE)*, May 2013, pp. 522–531, doi: 10.1109/ICSE.2013.6606598.

[53] A. Ghannem, M. S. Hamdi, M. Kessentini, and H. H. Ammar, "Search-Based Requirements Traceability Recovery," in *Lecture Notes in Networks and Systems*, vol. 15, 2018, pp. 156–171.

[54] S. Shalev-Shwartz and S. Ben-David, *Understanding Machine Learning*, vol. 9781107057.

Cambridge: Cambridge University Press, 2014.
[55] D. Falessi, M. Di Penta, G. Canfora, and G. Cantone, “Estimating the number of remaining links in traceability recovery,” *Empir. Softw. Eng.*, vol. 22, no. 3, pp. 996–1027, 2017, doi: 10.1007/s10664-016-9460-6.
[56] M. Rath, J. Rendall, J. L. C. Guo, J. Cleland-Huang, and P. Maeder, “Traceability in the Wild: Automatically Augmenting Incomplete Trace Links,” *CoRR*, vol. abs/1804.0, Apr. 2018, [Online]. Available: http://arxiv.org/abs/1804.02433.
[57] N. V Chawla, “Data Mining for Imbalanced Datasets: An Overview,” in *Data Mining and Knowledge Discovery Handbook*, O. Maimon and L. Rokach, Eds. New York: Springer-Verlag, 2005, pp. 853–867.
[58] J. Guo, J. Cheng, and J. Cleland-Huang, “Semantically Enhanced Software Traceability Using Deep Learning Techniques,” in *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, May 2017, pp. 3–14, doi: 10.1109/ICSE.2017.9.
[59] A. Amidi and S. Amidi, “VIP Cheatsheet: Recurrent Neural Networks,” 2018. https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks (accessed Mar. 08, 2021).
[60] V. Csuvik, A. Kicsi, and L. Vidacs, “Source Code Level Word Embeddings in Aiding Semantic Test-to-Code Traceability,” in *2019 IEEE/ACM 10th International Symposium on Software and Systems Traceability (SST)*, May 2019, pp. 29–36, doi: 10.1109/SST.2019.00016.
[61] H. Abukwaik, A. Burger, B. K. Andam, and T. Berger, “Semi-Automated Feature Traceability with Embedded Annotations,” in *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Sep. 2018, pp. 529–533, doi: 10.1109/ICSME.2018.00049.
[62] C. Mills and S. Haiduc, “The Impact of Retrieval Direction on IR-Based Traceability Link Recovery,” in *2017 IEEE/ACM 39th International Conference on Software Engineering: New Ideas and Emerging Technologies Results Track (ICSE-NIER)*, May 2017, pp. 51–54, doi: 10.1109/ICSE-NIER.2017.14.
[63] C. Mills, G. Bavota, S. Haiduc, R. Oliveto, A. Marcus, and A. De Lucia, “Predicting Query Quality for Applications of Text Retrieval to Software Engineering Tasks,” *ACM Trans. Softw. Eng. Methodol.*, vol. 26, no. 1, pp. 1–45, Jul. 2017, doi: 10.1145/3078841.
[64] D. Falessi, J. Roll, J. Guo, and J. Cleland-Huang, “Leveraging Historical Associations between Requirements and Source Code to Identify Impacted Classes,” *IEEE Trans. Softw. Eng.*, vol. 46, no. 4, pp. 420–441, Aug. 2018, doi: 10.1109/TSE.2018.2861735.
[65] Mendix, “Mendix Announces Studio and Studio Pro; No-Code and Low-Code Visual Development Environments,” 2019. https://www.mendix.com/press/mendix-announces-studio-and-studio-pro-no-code-and-low-code-visual-development-environments/ (accessed Feb. 11, 2021).
[66] Atlassian, “Jira Overview | Products, Projects and Hosting.” https://www.atlassian.com/software/jira/guides/getting-started/overview#key-terms-to-know (accessed Jun. 30, 2021).
[67] Atlassian, “Who uses Jira?” https://www.atlassian.com/software/jira/guides/use-cases/who-uses-jira (accessed Jun. 30, 2021).
[68] M. Rehkopf, “User Stories | Examples and Template,” *Atlassian Agile Coach*, 2021. https://www.atlassian.com/agile/project-management/user-stories (accessed Jun. 30, 2021).
[69] Max Rehkopf, “Agile epics: definition, examples, and templates.” https://www.atlassian.com/agile/project-management/epics (accessed Jun. 30, 2021).
[70] G. Lucassen, M. Robeer, F. Dalpiaz, J. M. E. M. van der Werf, and S. Brinkkemper, “Extracting conceptual models from user stories with Visual Narrator,” *Requir. Eng.*, vol. 22, no. 3, pp. 339–358, Sep. 2017, doi: 10.1007/s00766-017-0270-1.

[71] T. Kluyver *et al.*, “Jupyter Notebooks—a publishing format for reproducible computational workflows,” in *Positioning and Power in Academic Publishing: Players, Agents and Agendas - Proceedings of the 20th International Conference on Electronic Publishing, ELPUB 2016*, 2016, pp. 87–90, doi: 10.3233/978-1-61499-649-1-87.

[72] M. F. Porter, “An algorithm for suffix stripping,” *Program*, 1980.

[73] F. Pedregosa *et al.*, “Scikit-learn: Machine learning in Python,” *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.

[74] P. Virtanen *et al.*, “SciPy 1.0: fundamental algorithms for scientific computing in Python,” *Nat. Methods*, vol. 17, no. 3, pp. 261–272, Mar. 2020, doi: 10.1038/s41592-019-0686-2.

[75] J. Yoon, “Forecasting of Real GDP Growth Using Machine Learning Models: Gradient Boosting and Random Forest Approach,” *Comput. Econ.*, vol. 57, no. 1, pp. 247–265, 2021, doi: 10.1007/s10614-020-10054-w.

[76] A. Callens, D. Morichon, S. Abadie, M. Delpey, and B. Liquet, “Using Random forest and Gradient boosting trees to improve wave forecast at a specific location,” *Appl. Ocean Res.*, vol. 104, no. September, 2020, doi: 10.1016/j.apor.2020.102339.

[77] A. Swalin, “CatBoost vs. Light GBM vs. XGBoost,” 2018. https://www.kdnuggets.com/2018/03/catboost-vs-light-gbm-vs-xgboost.html (accessed Jul. 22, 2021).

[78] S. N. Kasturi, “XGBOOST vs LightGBM: Which algorithm wins the race !!!,” 2019. https://towardsdatascience.com/lightgbm-vs-xgboost-which-algorithm-win-the-race-1ff7dd4917d (accessed Jul. 22, 2021).

[79] I. van de Weerd and S. Brinkkemper, “Meta-modeling for situational analysis and design methods,” *Handb. Res. Mod. Syst. Anal. Des. Technol. Appl.*, pp. 35–54, 2008, doi: 10.4018/978-1-59904-887-1.ch003.

[80] J. R. Quinlan, “Induction of decision trees,” *Mach. Learn.*, vol. 1, no. 1, pp. 81–106, Mar. 1986, doi: 10.1007/BF00116251.

[81] L. Briand, D. Bianculli, S. Nejati, F. Pastore, and M. Sabetzadeh, “The Case for Context-Driven Software Engineering Research: Generalizability Is Overrated,” *IEEE Softw.*, vol. 34, no. 5, pp. 72–75, 2017, doi: 10.1109/MS.2017.3571562.

[82] Center of Excellence for Software & Systems Traceability, “No Title.” http://coest.org/ (accessed Aug. 06, 2021).

Appendix A - Interview Protocol

1. Introduction Prior to the Interview
Thanks for participating in this interview. As [CONTACT PERSON AT MENDIX] may have told you, this interview is part of my research on automatic tracing of requirements to models. I want to get a grasp on 1) how customers use Mendix software to develop their software

Consent
Before we start. I will record the audio of this interview. This audio will be used to summarise the findings gained. After summarising the audio will be deleted. The summary will be shared with you and so you have the possibility to change/correct the summary. Is this okay with you?

2. General Background
Can you give me a brief description of your job and your team?
Can you explain all the phases of a typical project?
How does Studio fit in this?

3. Building an app using Mendix
Can you briefly describe how a typical Mendix app is build using Mendix software?

- As a customer of Mendix
- As an internal team of Mendix

Do many teams within Mendix use Studio to develop apps?
Why do some teams use Studio while others don't?

How does versioning work when building a Mendix app?
Where is the binary file stored?
When you make a revision (commit), what data do you send to server?
Would it be possible to find out which models were adjusted, when only looking at the data from a commit?

4. Requirements
What are the different ways requirements can be managed in Mendix Software?
Do you use systems, other than Mendix, to manage requirements?

- Why not everything in Mendix?
- Do you use JIRA?
- How do these systems work together?

Do you have a recommended way of documenting requirements?

- User stories?

Can you show me how a user story is made in SPRINTR?

- Are there naming conventions?
- These user stories could manually be related to a commit. Are there any other artefacts it can be related to?

5. Traceability
How do you cope with changing requirements?
How do you validate if requirements are implemented?
How do you link or trace software artefacts to each other, e.g., code to user stories

6. Finalization

This was the interview.

- Do you have any questions?
- Do you know colleagues which I should speak to for this research?

Appendix B – Feature Overview

ID	Feature Name	Feature Family	Description
1	Creation commit date dif	Process-Related	The time delta between date of commit creation and JIRA issue creation
2	Updated commit date dif	Process-Related	The time delta between date of commit creation and last JIRA issue update
3	Resolved commit date dif	Process-Related	The time delta between date of commit creation and date when JIRA issue was resolved
4	Assignee is commiter	Process-Related	Binary indicator indicating if the person who committed is the same person as the one who committed
5	vsm logs jira as query	IR-Related	Cosine similarity between commit log and Entire JIRA issue using JIRA issue as query
6	vsm logs log as query	IR-Related	Cosine similarity between commit log and Entire JIRA issue using log as query
7	vsm unit names jira as query	IR-Related	Cosine similarity between commit unit names and Entire JIRA issue using JIRA issue as query
8	vsm unit names log as query	IR-Related	Cosine similarity between commit unit names and Entire JIRA issue using unit names as query
9	vsm unitnames comments comments as query	IR-Related	Cosine similarity between commit unit names and JIRA comments using comments as query
10	vsm unitnames comments unitnames as query	IR-Related	Cosine similarity between commit unit names and JIRA comments using unit names as query
11	vsm unitnames description description as query	IR-Related	Cosine similarity between commit unit names and JIRA descriptions using descriptions as query
12	vsm unitnames description unitnames as query	IR-Related	Cosine similarity between commit unit names and JIRA descriptions using unit names as query
13	vsm summary logs summary as query	IR-Related	Cosine similarity between commit logs and JIRA summaries using summaries as query
14	vsm summary logs logs as query	IR-Related	Cosine similarity between commit logs and JIRA summaries using logs as query

15	vsm summary unitNames summary as query	IR-Related	Cosine similarity between commit unit names and JIRA summaries using summaries as query
16	vsm summary unitNames summary as query	IR-Related	Cosine similarity between commit unit names and JIRA summaries using logs as query
17	vsm description description as query	IR-Related	Cosine similarity between commit logs and JIRA descriptions using descriptions as query
18	vsm description log as query	IR-Related	Cosine similarity between commit logs and JIRA descriptions using logs as query
19	vsm comments comments as query	IR-Related	Cosine similarity between commit logs and JIRA comments using comments as query
20	vsm comments log as query	IR-Related	Cosine similarity between commit logs and JIRA comments using logs as query
21	vsm svn jira jira as query	IR-Related	Cosine similarity between commit and JIRA issue using JIRA issue as query
22	vsm svn jira svn as query	IR-Related	Cosine similarity between commit and JIRA issue using commit as query
23	vsm svn summary svn as query	IR-Related	Cosine similarity between commit and JIRA summaries using commits as query
24	vsm svn summary summary as query	IR-Related	Cosine similarity between commit and JIRA summaries using JIRA summaries as query
25	vsm svn description svn as query	IR-Related	Cosine similarity between commit and JIRA description using commit as query
26	vsm svn description description as query	IR-Related	Cosine similarity between commit and JIRA description using JIRA descriptions as query
27	vsm svn comments svn as query	IR-Related	Cosine similarity between commit and JIRA comments using commits as query
28	vsm svn comments comments as query	IR-Related	Cosine similarity between commit and JIRA comments using JIRA comments as query
29	unique term count jira	Document Statistics	Number of unique terms in a JIRA issue
30	unique term count svn	Document Statistics	Number of unique terms in a commit
31	total term count jira	Document Statistics	Total number of terms in a JIRA issue

32	total term count svn	Document Statistics	Total number of terms in a commit
33	overlap percentage compared to jira	Document Statistics	Overlap of terms between JIRA issue and commit compared to a JIRA issue
34	overlap percentage compared to svn	Document Statistics	Overlap of terms between JIRA issue and commit compared to a commit
35	overlap percentage compared to union	Document Statistics	Overlap of terms between JIRA issue and commit compared to the union of a commit and JIRA issue
36	SvnAsQuery avgIDF	Query Quality	Average IDF when using SVN as query
37	SvnAsQuery maxIDF	Query Quality	Maximum IDF when using SVN as query
38	SvnAsQuery devIDF	Query Quality	Standard Deviation of IDF when using SVN as query
39	SvnLogsAsQuery avgIDF	Query Quality	Average IDF when using SVN logs as query
40	SvnLogsAsQuery maxIDF	Query Quality	Maximum IDF when using SVN logs as query
41	SvnLogsAsQuery devIDF	Query Quality	Standard Deviation of IDF when using SVN logs as query
42	SvnUnitNamesAs Query avgIDF	Query Quality	Average IDF when using SVN unit names as query
43	SvnUnitNamesAs Query maxIDF	Query Quality	Maximum IDF when using SVN unit names as query
44	SvnUnitNamesAs Query devIDF	Query Quality	Standard Deviation of IDF when using SVN logs as query
45	JiraAsQuery avgIDF	Query Quality	Average IDF when using JIRA issues as query
46	JiraAsQuery maxIDF	Query Quality	Maximum IDF when using JIRA issues as query
47	JiraAsQuery devIDF	Query Quality	Standard Deviation of IDF when using JIRA issues as query
48	JiraSummariesAs Query avgIDF	Query Quality	Average IDF when using JIRA summaries as query
49	JiraSummariesAs Query maxIDF	Query Quality	Maximum IDF when using JIRA summaries as query
50	JiraSummariesAs Query devIDF	Query Quality	Standard Deviation of IDF when using JIRA sumaries as query
51	JiraDescriptions AsQuery avgIDF	Query Quality	Average IDF when using JIRA descriptions as query
52	JiraDescriptions AsQuery maxIDF	Query Quality	Maximum IDF when using JIRA descriptions as query

53	JiraDescriptions AsQuery_devIDF	Query Quality	Standard Deviation of IDF when using JIRA descriptions as query
54	JiraCommentsAs Query_avgIDF	Query Quality	Average IDF when using JIRA comments as query
55	JiraCommentsAs Query_maxIDF	Query Quality	Maximum IDF when using JIRA comments as query
56	JiraCommentsAs Query_devIDF	Query Quality	Standard Deviation of IDF when using JIRA comments as query
57	SvnAsQuery avgICTF	Query Quality	Average ICTF when using SVN as query
58	SvnAsQuery maxICTF	Query Quality	Maximum ICTF when using SVN as query
59	SvnAsQuery devICTF	Query Quality	Standard Deviation of ICTF when using SVN as query
60	SvnLogsAsQuery avgICTF	Query Quality	Average ICTF when using SVN logs as query
61	SvnLogsAsQuery maxICTF	Query Quality	Maximum ICTF when using SVN logs as query
62	SvnLogsAsQuery devICTF	Query Quality	Standard Deviation of ICTF when using SVN logs as query
63	SvnUnitNamesAs Query_avgICTF	Query Quality	Average ICTF when using SVN unit names as query
64	SvnUnitNamesAs Query_maxICTF	Query Quality	Maximum ICTF when using SVN unit names as query
65	SvnUnitNamesAs Query_devICTF	Query Quality	Standard Deviation of ICTF when using SVN unit names as query
66	JiraAsQuery avgICTF	Query Quality	Average ICTF when using JIRA issues as query
67	JiraAsQuery maxICTF	Query Quality	Maximum ICTF when using JIRA issues as query
68	JiraAsQuery devICTF	Query Quality	Standard Deviation of ICTF when using JIRA issues as query
69	JiraSummariesAs Query_avgICTF	Query Quality	Average ICTF when using JIRA summaries as query
70	JiraSummariesAs Query_maxICTF	Query Quality	Maximum ICTF when using JIRA summaries as query
71	JiraSummariesAs Query_devICTF	Query Quality	Standard Deviation of IDF when using JIRA summaries as query
72	JiraDescriptions AsQuery avgICTF	Query Quality	Average ICTF when using JIRA descriptions as query
73	JiraDescriptions AsQuery maxICTF	Query Quality	Maximum ICTF when using JIRA descriptions as query
74	JiraDescriptions AsQuery devICTF	Query Quality	Standard Deviation of ICTF when using JIRA descriptions as query
75	JiraCommentsAs Query_avgICTF	Query Quality	Average ICTF when using JIRA comments as query

76	JiraCommentsAsQuery_maxICTF	Query Quality	Maximum ICTF when using JIRA comments as query
77	JiraCommentsAsQuery_devICTF	Query Quality	Standard Deviation of ICTF when using JIRA comments as query
78	SvnAsQuery avgEntropy	Query Quality	Average Entropy when using commit as query
79	SvnAsQuery medEntropy	Query Quality	Median Entropy when using commit as query
80	SvnAsQuery maxEntropy	Query Quality	Maximum Entropy when using commit as query
81	SvnAsQuery devEntropy	Query Quality	Standard deviation of Entropy when using commit as query
82	SvnLogsAsQuery avgEntropy	Query Quality	Average Entropy when using commit logs as query
83	SvnLogsAsQuery medEntropy	Query Quality	Median Entropy when using commit logs as query
84	SvnLogsAsQuery maxEntropy	Query Quality	Maximum Entropy when using commit logs as query
85	SvnLogsAsQuery devEntropy	Query Quality	Standard deviation of Entropy when using commit logs as query
86	SvnUnitNamesAsQuery avgEntropy	Query Quality	Average Entropy when using commit unit names as query
87	SvnUnitNamesAsQuery medEntropy	Query Quality	Median Entropy when using commit unit names as query
88	SvnUnitNamesAsQuery maxEntropy	Query Quality	Maximum Entropy when using commit unit names as query
89	SvnUnitNamesAsQuery devEntropy	Query Quality	Standard deviation of Entropy when using commit unit names as query
90	JiraAsQuery avgEntropy	Query Quality	Average Entropy when using JIRA issue as query
91	JiraAsQuery medEntropy	Query Quality	Median Entropy when using JIRA issue as query
92	JiraAsQuery maxEntropy	Query Quality	Maximum Entropy when using JIRA issue as query
93	JiraAsQuery devEntropy	Query Quality	Standard deviation of Entropy when using JIRA issue as query
94	JiraSummariesAsQuery avgEntropy	Query Quality	Average Entropy when using JIRA summaries as query
95	JiraSummariesAsQuery medEntropy	Query Quality	Median Entropy when using JIRA summaries as query
96	JiraSummariesAsQuery maxEntropy	Query Quality	Maximum Entropy when using JIRA summaries as query

97	JiraSummariesAsQuery devEntropy	Query Quality	Standard deviation of Entropy when using JIRA summaries as query
98	JiraDescriptionsAsQuery avgEntropy	Query Quality	Average Entropy when using JIRA descriptions as query
99	JiraDescriptionsAsQuery medEntropy	Query Quality	Median Entropy when using JIRA descriptions as query
100	JiraDescriptionsAsQuery maxEntropy	Query Quality	Maximum Entropy when using JIRA descriptions as query
101	JiraDescriptionsAsQuery devEntropy	Query Quality	Standard deviation of Entropy when using JIRA descriptions as query
102	JiraCommentsAsQuery avgEntropy	Query Quality	Average Entropy when using JIRA comments as query
103	JiraCommentsAsQuery medEntropy	Query Quality	Median Entropy when using JIRA comments as query
104	JiraCommentsAsQuery maxEntropy	Query Quality	Maximum Entropy when using JIRA comments as query
105	JiraCommentsAsQuery devEntropy	Query Quality	Standard deviation of Entropy when using JIRA comments as query
106	SvnAsQuery QueryScope	Query Quality	Query Scope when using commit as query
107	SvnLogsAsQuery QueryScope	Query Quality	Query Scope when using commit logs as query
108	SvnUnitNamesAsQuery QueryScope	Query Quality	Query Scope when using commit unit names as query
109	JiraAsQuery QueryScope	Query Quality	Query Scope when using JIRA issue as query
110	JiraSummariesAsQuery QueryScope	Query Quality	Query Scope when using JIRA summaries as query
111	JiraDescriptionsAsQuery QueryScope	Query Quality	Query Scope when using JIRA descriptions as query
112	JiraCommentsAsQuery QueryScope	Query Quality	Query Scope when using JIRA comments as query
113	SvnAsQuery_SCS	Query Quality	SCS when using commit as query
114	SvnLogsAsQuery SCS	Query Quality	SCS when using commit logs as query
115	SvnUnitNamesAsQuery_SCS	Query Quality	SCS when using commit unit names as query

116	JiraAsQuery_SCS	Query Quality	SCS when using JIRA issue as query
117	JiraSummariesAs Query_SCS	Query Quality	SCS when using JIRA summaries as query
118	JiraDescriptions AsQuery_SCS	Query Quality	SCS when using JIRA descriptions as query
119	JiraCommentsAs Query_SCS	Query Quality	SCS when using JIRA comments as query
120	SvnAsQuery avgSCQ	Query Quality	Average SCQ when using commit as query
121	SvnAsQuery maxSCQ	Query Quality	Maximum SCQ when using commit as query
122	SvnAsQuery sumSCQ	Query Quality	Summation of SCQ when using commit as query
123	SvnLogsAsQuery avgSCQ	Query Quality	Average SCQ when using commit logs as query
124	SvnLogsAsQuery maxSCQ	Query Quality	Maximum SCQ when using commit as query
125	SvnLogsAsQuery sumSCQ	Query Quality	Summation of SCQ when using commit as query
126	SvnUnitNamesAs Query_avgSCQ	Query Quality	Average SCQ when using commit unit names as query
127	SvnUnitNamesAs Query_maxSCQ	Query Quality	Maximum SCQ when using commit unit names as query
128	SvnUnitNamesAs Query_sumSCQ	Query Quality	Summation of SCQ when using commit unit names as query
129	JiraAsQuery avgSCQ	Query Quality	Average SCQ when using JIRA issues as query
130	JiraAsQuery maxSCQ	Query Quality	Maximum SCQ when using JIRA issues as query
131	JiraAsQuery sumSCQ	Query Quality	Summation of SCQ when using JIRA issues as query
132	JiraSummariesAs Query_avgSCQ	Query Quality	Average SCQ when using JIRA summaries as query
133	JiraSummariesAs Query_maxSCQ	Query Quality	Maximum SCQ when using JIRA summaries as query
134	JiraSummariesAs Query_sumSCQ	Query Quality	Summation of SCQ when using JIRA summaries as query
135	JiraDescriptions AsQuery_avgSCQ	Query Quality	Average SCQ when using JIRA descriptions as query
136	JiraDescriptions AsQuery maxSCQ	Query Quality	Maximum SCQ when using JIRA descriptions as query
137	JiraDescriptions AsQuery sumSCQ	Query Quality	Summation of SCQ when using JIRA descriptions as query
138	JiraCommentsAs Query_avgSCQ	Query Quality	Average SCQ when using JIRA comments as query
139	JiraCommentsAs Query_maxSCQ	Query Quality	Maximum SCQ when using JIRA comments as query

140	JiraCommentsAsQuery_sumSCQ	Query Quality	Summation of SCQ when using JIRA comments as query
141	SvnAsQuery avgPMI	Query Quality	Average PMI when using commit as query
142	SvnAsQuery maxPMI	Query Quality	Maximum PMI when using commit as query
143	SvnLogsAsQuery avgPMI	Query Quality	Average PMI when using commit logs as query
144	SvnLogsAsQuery maxPMI	Query Quality	Maximum PMI when using commit logs as query
145	SvnUnitNamesAsQuery_avgPMI	Query Quality	Average PMI when using commit unit names as query
146	SvnUnitNamesAsQuery_maxPMI	Query Quality	Maximum PMI when using commit unit names as query
147	JiraAsQuery avgPMI	Query Quality	Average PMI when using JIRA issues as query
148	JiraAsQuery maxPMI	Query Quality	Maximum PMI when using JIRA issues as query
149	JiraSummariesAsQuery_avgPMI	Query Quality	Average PMI when using JIRA summaries as query
150	JiraSummariesAsQuery_maxPMI	Query Quality	Maximum PMI when using JIRA summaries as query
151	JiraDescriptionsAsQuery avgPMI	Query Quality	Average PMI when using JIRA descriptions as query
152	JiraDescriptionsAsQuery maxPMI	Query Quality	Maximum PMI when using JIRA descriptions as query
153	JiraCommentsAsQuery_avgPMI	Query Quality	Average PMI when using JIRA comments as query
154	JiraCommentssAsQuery_maxPMI	Query Quality	Maximum PMI when using JIRA comments as query