Semi-automatic reconstruction and documentation of software development methods

Marko Janković
Semi-automatic reconstruction and documentation of software development methods

A dissertation presented
by

Marko Janković

to
The Faculty of Computer and Information Science
in partial fulfilment of the requirements for the degree of
Doctor of Science
in
Computer and Information Science

Ljubljana, 2020
APPROVAL

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the award of any other degree or diploma of the university or other institute of higher learning, except where due acknowledgement has been made in the text.

— Marko Janković —
January 2020

The submission has been approved by

dr. Matjaž B. Jurič
Professor of Computer and Information Science
EXAMINER

dr. Damjan Vavpotič
Associate Professor of Computer and Information Science
EXAMINER

dr. Martin Pinzger
Professor of Computer Sciences
EXTERNAL EXAMINER
University of Klagenfurt

dr. Marko Bajec
Professor of Computer and Information Science
ADVISOR
PREVIOUS PUBLICATION

I hereby declare that the research reported herein was previously published/submitted for publication in peer reviewed journals or publicly presented at the following occasions:


I certify that I have obtained a written permission from the copyright owner(s) to include the above published material(s) in my thesis. I certify that the above material describes work completed during my registration as graduate student at the University of Ljubljana.
POVZETEK


Ključne besede programski repozitorij, metode razvoja programske opreme, razvojni proces, sistem za beleženje zahtevkov, sistem za nadzor verzij
Software development is a complex and creative process. In contrast to a typical business process it tends to be more dynamic and dependent on a number of circumstances. Empirical studies show that companies still don’t document their development practices, or if they do, these are not up-to-date and do not reflect how they really develop software. On the other hand, various supporting tools such as issue tracking system, revision control system, document management system, etc. are used by developers and project managers during their work, capturing a vast body of knowledge about how a software development process has been performed. The main objective of this dissertation is to propose an approach that can help companies in documenting their real development practice. Comparing to existing approaches that require substantial effort on the side of project members, our approach extracts information on development practice directly from software repositories. Five companies have been studied to identify information that can be retrieved from software repositories. Based on this, an approach to reconstruct development practice has been developed. The approach has been evaluated on a real software repository shared by an additional company. The results confirm that software repository information suffice for the reconstruction of various aspects of development process, i.e. disciplines, activities, user roles, and artifacts.

Keywords  software repository, software development method, development process, issue tracking system, revision control system
ACKNOWLEDGEMENTS

Firstly, I wish to thank my thesis advisor, Professor Marko Bajec, without whose support this dissertation would not have been possible. Furthermore, I sincerely thank Professor Martin Pinzger for the long and sometimes stressful discussions during my research visits, which always (one way or another) helped me move forward.

I thank all members of the Data Technologies Laboratory who made all the time I have spent doing research fun and relaxing.

Last but not least, I wish to thank my family and friends. Dear parents, thank you for all the support, help and love that helped me become who I am today. Beloved Manca, thank you for your love, encouragement, patience, and understanding. Filip and Zara, thank you for all your smiles and for reminding me of what truly matters. Dear friends, thank you for being there for me when I need you.

# CONTENTS

**Povzetek** vii  
**Abstract** ix  
**Acknowledgements** xi

## 1 Introduction
1.1 Motivation ........................................... 2  
1.2 Scientific Contributions .............................. 4  
1.3 Dissertation Overview ................................. 5

## 2 Related Work
2.1 Metod Engineering ..................................... 8  
2.1.1 Situational Method Engineering .................... 8  
2.2 Mining Software Repositories ......................... 10  
2.2.1 Topic Analysis ..................................... 12  
2.2.2 Software Process Recovery ......................... 13  
2.3 Software Process Mining ............................... 14  
2.4 Software Process Discovery ......................... 16  
2.5 Other Related Research Fields ....................... 18  
2.6 Our Approach Versus Existing Approaches ........... 19

## 3 Research Approach
3.1 Research Questions ................................. 22  
3.2 Data Collection Procedure ............................ 22  
3.3 Research Approach ................................... 23
4 Analysis of Software Repositories
   4.1 Information Stored in Software Repositories ........................................ 27
   4.2 Links Between Software Repositories .................................................... 30
      4.2.1 User Resolution .............................................................................. 30
   4.3 Real Software Repositories Data Analysis ............................................... 31
      4.3.1 Import of Data .............................................................................. 33
      4.3.2 Analysis and Results ..................................................................... 33
   4.4 Chapter Summary .................................................................................... 40

5 The Reconstruction of a Software Development Method .................................. 43
   5.1 Metamodel of Software Development Method ......................................... 44
   5.2 Construction of a Base Method ............................................................... 48
   5.3 Reconstruction of Software Development Method Elements ................... 50
      5.3.1 Reconstructing Artifacts .................................................................. 50
      5.3.2 Reconstructing Activities ............................................................... 51
      5.3.3 Reconstructing User Roles and Disciplines ..................................... 52
      5.3.4 Algorithm for Reconstructing Artifacts and Activities ..................... 52
   5.4 Limitations and prerequisites .................................................................. 55
   5.5 Chapter Summary .................................................................................... 56

6 Software Development Process Analysis ......................................................... 59
   6.1 Post-mortem Analysis .............................................................................. 60
      6.1.1 Time Spent per Discipline/Activity/User role (daily/aggregated) ........ 60
      6.1.2 Actual vs. Estimated Time ............................................................... 63
      6.1.3 Project Timeline ............................................................................. 64
   6.2 On-the-fly Analysis .................................................................................. 64
      6.2.1 Monitor, Control, and Guide ............................................................ 66
      6.2.2 Real-time Process Reconstruction & Analysis .................................... 66
   6.3 Chapter Summary .................................................................................... 67

7 Evaluation ...................................................................................................... 69
   7.1 Profile of the Company ............................................................................ 70
   7.2 Analysis of the Software Repository Content .......................................... 71
   7.3 Construction of the Base Method Draft .................................................... 73
   7.4 Reconstruction of the Project Software Development Method Elements .... 73
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.5 Checking Project Performance</td>
<td>76</td>
</tr>
<tr>
<td>7.6 Discussion</td>
<td>77</td>
</tr>
<tr>
<td>7.7 Threats to Validity</td>
<td>80</td>
</tr>
<tr>
<td>7.8 Chapter Summary</td>
<td>81</td>
</tr>
<tr>
<td><strong>8 Conclusion</strong></td>
<td></td>
</tr>
<tr>
<td>8.1 Principal Scientific Contributions</td>
<td>85</td>
</tr>
<tr>
<td>8.2 Future Work</td>
<td>85</td>
</tr>
<tr>
<td><strong>A iSPRToolset</strong></td>
<td></td>
</tr>
<tr>
<td>Tool functionalities</td>
<td>89</td>
</tr>
<tr>
<td><strong>Bibliography</strong></td>
<td>99</td>
</tr>
<tr>
<td><strong>Razširjeni povzetek</strong></td>
<td>107</td>
</tr>
</tbody>
</table>
Introduction
1.1 Motivation

Software development, as any other engineering activity, requires a systematic and disciplined approach to assure the quality of the process and its results, i.e. the software that we develop [1]. This has been recognized already in the early beginnings of the software development era and has led to the construction of many software development methods, which were created to enable the control over methods used in software development and to tell us how to develop a software step by step in a controlled manner. Over the years, it turned out that there is no ideal development method that could fit all kinds of projects, even in the context of a single company. How suitable a particular development method is, actually depends on many factors, ranging from project and company characteristics to the characteristics of the development team. These findings have been identified by many researchers, e.g. [2–8].

One of the research fields that emerged as a result of the aforementioned problems is method engineering. Researchers in this field devoted a lot of effort finding suitable solutions. One of them is the so called situational method engineering, which is a process of constructing development methods specifically attuned to the needs of projects [9]. Such development methods would either be composed of fragments of other development methods or created by tailoring the development method that is generally used and known to the company. Unfortunately, there are several obstacles that hinder the application of situational method engineering in practice [10, 11]. One is for instance that we need somebody who is capable of applying the method engineering process (must be familiar with various development methods, method fragments etc.) and what is even more challenging, we need enough time before starting the project so that this person can do the job. In real settings, where projects are almost always run in very tight schedules, this is rarely the case [12–14].

While the majority of software development companies adopt one of the known software development methods (like the Agile Unified Process [15], for example), empirical studies show that the way development happens in reality differs significantly from how it is prescribed in those methods [16–18]. Furthermore, many companies nowadays adopt the light and agile methods (like the Scrum [19] or Kanban [20], for example) that have arisen as a response to the complex and inflexible traditional methods [21–23]. However, agile methods often only describe a lightweight project management framework and leave the specification and documentation of the development practices that
should be followed by the companies and development teams. This means that in the majority of companies, the actual software development processes are poorly documented, if documented at all [18]. This considerably hinders the efficiency of software production, since similar mistakes might happen over and over again, best practices are not captured, there is no documentation from which new employees could learn, the knowledge that might represent a competitive asset for a company is not shared among the employees, it is not possible to perform project performance analysis, etc. Hence, we would expect companies to be highly motivated for documenting their software processes, but it turns out this is not the case either. One reason might be that doing this manually takes a lot of time and effort. This calls for solutions that are automatized and do not require developers and other team members to be highly involved.

One possible way of addressing the aforementioned challenge involves analysis of information that is produced during software development and stored in software repositories. We are aware that it is too ambitious to expect that all the knowledge and experience could be acquired just by analyzing corresponding software repositories. However, we do expect that the information available in such repositories suffices to develop a good understanding of how a particular development process was performed, and how software development is conducted in a given company. Several research fields have already started to exploit data stored in the software repositories to recover information about the software development process. However, they mainly focus on the high-level view of the development process workflow that consists of disciplines or activities and are not interested in recovering more detail information such as what artifacts have been created as part of each activity, which user roles were responsible, and how this is compliant with the development method that is prescribed inside a company.

Based on the previous studies that were focused on the implementation of the situational method engineering process in practice [16, 24], companies see as beneficial if they are able to document and monitor their actual work on development projects and check how much it complies with their prescribed methods. In this way, they can detect deviations if they occur. Doing this manually is however, very time-consuming. An approach is thus needed that does not require more than just a minimal effort from developers.
1.2 Scientific Contributions

The main objective of our research is to address the described challenges by reconstructing information about a particular project performance from its data that is captured in software repositories. We assume that software repositories contain enough data to reconstruct at least the main method elements (i.e. disciplines, activities, artifacts). Moreover, the objective is to support development process analysis to learn how a particular project was performed and to possibly identify its positive and negative aspects in relation to its outcome and also during the project performance, so as to detect situations that might lead to project failures.

The main scientific contribution presented in this dissertation is an approach for semi-automatic reconstruction and documentation of companies’ software development methods, which exploit the data created and left behind by developers during their work. This data are stored and thus available in software repositories. The proposed approach requires a minimal involvement of developers, which turns to be essential for its use in real practice.

The main goals of this approach are thus to support:

- semi-automatic acquisition and documentation (base method) of the software development knowledge the companies and their individuals possess,
- semi-automatic reconstruction of software development method followed on a particular project using the data from software repositories,
- software development process analysis using the reconstructed software development method information in combination with the other data available in the software repositories to support the work of project members.

As part of our work we also developed a supporting tool, called iSPRToolset (for more details see Appendix A) that supports the following functionalities:

**Import and link data from various software repositories** The tool supports the import of the data from Git, Jira, Subversion, Github, Trello, LogicalDoc and DevTrack. Furthermore, it supports linkage between software repositories using regular expressions.

**User resolution** Link the user accounts from different software repositories belonging to the same developer using the information available (username, email, first and last name, etc.).
Define company’s base method  For the end users (e.g. project managers, developers) it is important to have a friendly and easy way to define current base method that is prescribed inside the company.

Reconstruct project’s method  Using the information stored in software repositories the tool can reconstruct an actual method that was followed in a project. It performs the reconstruction in an incremental way – issue by issue. In case of a deviation from the base method (e.g. no method element from the base method is compliant) or ambiguity (e.g. several method elements from a base method are compliant) it requires the involvement of project members.

Process analysis  tool supports different analyses about how the development process has been performed and how project has been conducted. The number of different analyses depends on the data that are available.

Parts of the dissertation have been published and presented on several international conferences [25, 26] and in SCI journal [27].

1.3  Dissertation Overview

The dissertation is organized in eight chapters. In Chapter 2, we position our research within the related work that consist of the following main research areas: method engineering, mining software repositories, software process mining, and software process discovery. In Chapter 3 we explain how the research was performed (research approach), giving also a brief information on the participating companies. In Chapter 4 we analyzed software repositories and present what kind of information they store, and how data among different software repositories can be linked. In Chapter 5 we present the semi-automatic approach to reconstruct the software development method followed on a particular project with the minimal involvement of developers. We also introduce the metamodel that underpin the reconstructed methods. In Chapter 6 we present the different post-mortem or on-the-fly analysis of the development process using the reconstructed information and additional information available in software repositories. Chapter 7 covers the evaluation of the proposed approach and its results. Finally, in Chapter 8 we conclude the dissertation and present some directions for the future work.
Related Work
There are several works that can be considered related to our research. These can be grouped into five categories, according to the research fields they come from:

- Method Engineering
- Mining Software Repositories
- Software Process Mining
- Software Process Discovery
- Other Related Research Fields

Note that there is some overlapping among these fields, in terms of approaches and techniques that they use. Different names that they carry are more a result of the fact that they come from different research groups and time periods. Below we present an overview of each of these research field and its relation to our research.

2.1 Method Engineering

Method engineering, as defined by Brinkkemper, is an engineering discipline to design, construct and adapt methods, techniques and tools for the development of information systems [28]. In general, the idea lies in the conceptualization, and construction of new software development methods and tools (or in the adaptation of existing ones), so that they best fit requirements of a certain company. In this way a company can adapt development method to their needs, rather than buying one of “off the shelf” methods and using it unchanged. The research on method engineering has a long tradition. A good introduction to the field can be found in [29].

2.1.1 Situational Method Engineering

Based on the method engineering principles, a specific direction has emerged, called situational method engineering. An excellent review on the past research can be found in the work by Henderson-Sellers and Ralyte [9]. As the name implies, situational method engineering deals with developing new methods or tailoring existing ones on-the-fly, i.e. to meet specific project situations. This is very important since even inside a company we often have to deal with different kinds of projects, and tailoring of a method based
on the company and project characteristics contributes to the higher quality of the development process and its product, i.e. the developed software [18].

In situational method engineering methods are constructed from the method elements, such as fragments, chunks, components, etc. All these method elements are available in a repository of all method elements - often called method base [28, 30–34]. To construct a new method, method engineer in cooperation with the company members defines and selects the method elements from the method base that are required and compliant with the company characteristics and requirements. The construction can be done in a bottom-up fashion by selecting the individual method elements and constructing them in the development method or by top-down approach in which case we start with a top level architecture and recursively refining the details to identify method elements. In both cases, all the information is gathered by interviewing project managers.

To construct a method, several approaches have been proposed, such as:

**Assembly-based approach** assembles a situational method from the method elements that are already stored in a repository (method base). Method elements are selected based on the project characteristics [33].

**Paradigm-based approach** comparing to assembly-based approach, it supports creation of new method elements in conformance with a metamodel and stores them in a method base. This is important to support situations when pre-existing method elements are not sufficient [35].

**Extension-based approach** uses generic patterns with aim at proposing a means for constructing situation specific methods [36]. Patterns encapsulate knowledge about processes that can be reused and applied in different settings. They guide method engineer in the construction of methods.

**Ad hoc approach** is a combination of the previous three approaches (assembly-based, paradigm-based and extension-based) [37].

**Deontic matrix approach** uses deontic matrices to construct a method. A deontic matrix is a two-dimensional matrix that represents a relationship between pairs of method elements. Each value in the matrix present an assessment of the likelihood of the occurrence of these two method elements using five levels of possibility (mandatory, recommended, optional, discouraged, forbidden) [18].
UML Activity Diagrams approach uses activity diagrams to represent the method elements and relations among them. The method elements are selected by analyzing the application context. [39]

The configuration based approach focuses on the construction of a situational method from scratch, using configuration packages. Configuration package is a predefined method configuration, designed to fit selected development characteristics [40, 41].

All these approaches assume that (1) it is possible for project members to explicitly specify the required situational method upfront and successfully communicate requirements to the method engineer, and (2) requirements do not change over the lifetime of the project.

Once we have a method that is tailored to the characteristics of a company or further to a characteristics of a project, the work of situational method engineering is mainly done. However, there is another important aspect of development methods - how they are executed and followed on the actual projects. Often methods are not executed in the way as they are prescribed [13, 42, 43]. Thus, the observation and reconstruction of actual methods followed on projects, plays an important role in the identification of method elements that are either redundant or require alterations to be used in future. Furthermore, it can help to detect new method elements that capture new ways of development practice which might be a consequence of new circumstances not seen before.

With our research, we mainly focus on that last part. We use the data created by project members and stored in corresponding software repositories to reconstruct the development methods that were actually followed on these projects, and to detect deviations from the prescribed methods. Furthermore, using the data from software repositories allow us to monitor, control, guide and analyze a development process as it is performed.

### 2.2 Mining Software Repositories

Mining software repositories refers to the investigation and mining of data stored within software repositories such as revision control system, bug/issue tracking system, document management system, wikis, review tools, etc. [44–49] to get valuable insights about the development practice and software itself. The size of the available data related
Related Work

to software projects is increasing since more and more tools are used by developers during their work to support the development process. Kagdi et al. [44] and Jung et al. [50] provide a survey of approaches and studies that exist within the mining software repositories community. Mining software repositories is a broad research area and their major goals are manifold: (1) supporting software maintenance, (2) software process improvement, (3) empirical validation of new ideas in software engineering fields, (4) predicting defects or detecting inconsistencies [50].

Several approaches have been proposed that discover and exploit relationships in software repositories to extract and establish links between project members and the various artifacts created during development process, in order to help the project members during their work.

In [51, 52], Ćubranić et al. introduced the tool, named Hipikat, to provide developers with efficient and effective access to the repository of information (the collection of artifacts created during development and stored in software repositories), called project memory. Project members can use it to gain knowledge about the past experience that they can use to address the current problems and needs. This is especially important for novice developers that are not yet familiar with the ways of working inside the company. Hipikat integrates multiple information sources to extract the information about the following four types of artifacts: change tasks (issues/tasks from the issue tracking system), file versions (file versions committed to the revision control system), messages (messages from the newsgroups and mailing lists), and documents (design documents posted to the project’s website). Besides that, it also extracts the relationships (links) between artifacts and the information about the artifact’s author. Once ready, the project memory is used to determine relevant artifacts in response to a query. The query results are provided as a recommendation to a developer to support his/her work.

Similar research has been done by Begel et al. [53]. They collected information from different software repositories such as: Microsoft Visual Studio Team Foundation Server as a revision control system and issue tracking system, Active Directory as employee database, Outlook and Exchange to obtain emails from public mailing lists, etc. They stored crawled information to a graph (called Codebook) of typed nodes, which represent repository objects (such as people, changesets (commits), work items (issues/tasks), files, and source code), and typed edges, which label the relationships (links) of the nodes to one another (such as commits, bug assignments, caller/callee, use/def, textual allusions). Project members can search the Codebook, using the keywords, to obtain the
nodes whose metadata best matches the keywords. On top of the Codebook they built two applications, Hoozizat and Deep Intellisense. First is concerned with finding the people who own and are responsible for a feature, application programming interface (API), product or service, while the purpose of the second is to aid code investigation. It displays a reverse chronologically sorted list of events (code changes, linked work items, linked messages) related to that source code.

With our approach we also exploit data stored in various software repositories and links among them. However, compared to these approaches our main objective is to reconstruct the development method that underpins the development process, which would, comparing to these approaches, provide additional knowledge and information to support work of project members. Below we present in more details two sub-fields of mining software repository that are most closely related to our research. These are topic analysis and software process recovery.

2.2.1 Topic Analysis

Topic Analysis is a sub-field of mining software repositories where researchers are mainly focusing on mining data in software repositories in order to recover and analyze the main topics that represent activities, tasks, component, etc. that were active (in progress) at a particular stage of the development process. Basically this means that they try to extract the information about the activities/tasks that developers have been working on, which gives us a high level overview of what has been going on in the project.

To extract topics most approaches employ various statistical topic models, such as Latent Dirichlet Allocation [54] and Latent Semantic Indexing [55] that provide a means to automatically search, index, cluster and structure unstructured and unlabeled textual data (commit log comments, source code, documentation, mailing lists, etc.) [56–60]. Topic models accomplish these tasks by discovering a set of topics within the textual data, where a topic is a collection of co-occurring words. When extracting topics, the existing approaches use data from only one software repository, in most cases revision control system.

Although results show that this can be done efficiently, the topics do not convey much information on the underlying development method, except maybe the main activities, if they can be inferred from the topics.
2.2.2 Software Process Recovery

Software Process Recovery is a sub-field of mining software repositories - it uses the data from software repositories and mining techniques to reason about the software development process. The main goal of the software process recovery is to recover a development process by semi-automatically analyzing data stored in software repositories, and thus require a minimal involvement of developers. Several works on this have been done by Hindle et al. [61–66]

In [62], Hindle et al. proposed an approach for software process recovery that uses data from different software repositories to recover Unified Process Views, which illustrate how the relative emphasis on different disciplines changes over the course of the project. An example of a Unified process views is presented in Figure 2.1. Since the recovered views are based on the actual data generated during the project performance, we can assume that they present accurate information about the division of effort per discipline throughout the project.

![Figure 2.1](https://example.com/figure21.png)

**Figure 2.1** Unified process diagram: this is often used to explain the division of labour within the Unified Process [67].

The proposed approach consists of seven steps:

**Source acquisition** is the discovering and mirroring of relevant data from different software repositories. In their case studies they included the data from revision control system, bug tracking system and mailing lists.

**Extraction** attempts to abstract the data collected in the acquisition phase. The raw data from software repositories have to be transformed to the format that is usable in
the next steps. Extractors that they found useful include CREX [68], CVSAnalY [69], SoftChange [70], and MLStats [69].

**Unsupervised analysis** analyzes the extracted data in an unsupervised way, without the help of the end user. Here they employed STBD revisions [71], summary statistics, natural language processing, word-bag analysis, and topic analysis.

**Annotation** step is used to enhance the results of unsupervised analysis. This can be achieved, for example, by modifying the stop words, since the lexicon of the project is often unique [72]. This can influence the topics extracted as well as the classification of word-bags. This step also serves as a method to prepare trainings sets for supervised learners.

**Supervised analysis** performs analysis that require involvement of a human - mainly to prepare a training set of labeled commits. They use topic labeling and maintenance classification.

**Signal mapping and reporting** takes previous analyses and presents them in Recovered Unified Process Views.

The software process recovery as addressed by Hindle et al., is closely related to our research. In both cases the motivation is to enable reconstruction of a method elements followed on the project from data stored in software repositories with minimal involvement of developers. However in our research we go one step further and focus on the reconstruction of a development method in more details. Besides disciplines, we are also interested in activities, artifacts and user roles that construct a development method followed on a project.

### 2.3 Software Process Mining

Software Process Mining is closely related to the process mining research area. In process mining a lot of work has been done on the analysis of business processes from event logs that are commonly available in today’s information systems. More information about the process mining can be found in the process mining manifesto [73]. In software process mining researchers apply process mining techniques to derive a process map and
Related Work

identify inefficiencies, imperfections, and enhance existing process capabilities. An important step before we can apply process mining techniques, is the preparation of the event logs. In case of software process mining, the event logs are prepared using the data stored in various software repositories [74, 75].

In [76], Rubin et al. introduced the Process mining framework for software processes. They developed an approach that can incrementally discover the process model by its refinement every time there is new information available in the software repository. To prepare an event log, which was used for process mining, they used the data from the revision control system. They defined five event types (design, code, test, review, verification), which they extracted from revision control system by classifying files to event types based on the filepath, filename and extension. Furthermore, for the event timestamp they used commit timestamp, and for the event performer they set a person that performed a commit. Using this event log they performed analyses from the workflow, company, resource perspective.

Similar research has been done by Poncin et al. [77]. To prepare the event log they used the data stored in software repositories for six capstone projects performed by students. Students were expected to follow a well-known V software development model and adhere to the guidelines of the European Space Agency to produce the prescribed documents. Based on this information they prepared an event log using the FRASR tool [78]. They built process model for six student projects and compared how theirs were compliant with the V development model that they should follow.

Lemos et al. [79] applied process mining techniques to analyze a software development process with a goal to detect inconsistencies between a process model and its corresponding execution log. As an event log they used the data provided by a large Brazilian company which logged information about nine different development activities during a longer period of time. The event log included information of more than 2000 projects. Their results show that developers do not follow the prescribed process as it might be expected.

Several researchers applied process mining techniques to analyze processes on a more details granularity - on the level of code review, issue/bug life cycle, etc. For example, Poncin et al. [78] used data from Bugzilla for open source project GCC, the GNU Compiler Collection. They took the data for 1 year and it contained 42,373 bug reports. Using the information about the bug states (e.g. new, assigned, resolved, reopened, closed) they prepared an event log that they used to discover the process model of the bug re-
Samalikova et al. [80] applied process mining techniques to discover and analyze the ‘actual’ change control board (CCB) process model on the data from a large industrial company in The Netherlands. To prepare the event log required for process mining they used snapshots of the CCB database (copies of the status database on a weekly basis) that are stored in the software configuration management system. Each snapshot captures the evolution of the change request, i.e., the changes of the change requests status, in time. They compared the ‘actual’ CCB model with the ‘official’ one to detect inconsistencies, which were discussed with the development teams.

Gupta et al. [75, 81] extended their work by including several software repositories in the analysis of bug resolution process. They linked data from issue/bug tracking system, revision control system and peer code review system and used them for the preparation of the event log that consisted of several different activities such as bug reporting, bug fixing, patch submission, source code commit, etc. From the extracted event log they discovered runtime process model for the bug resolution and conducted process performance and efficiency analysis. They identified bottlenecks and detected anti-patterns. Furthermore, they performed organizational analysis and discovered metrics such as joint activities, subcontracting and handover of work.

In software process mining, researchers are mainly focusing on the reconstruction of software processes on the level of disciplines or on the level of issue life cycle, but do not consider reconstruction of a development method that underpins the development process. The main reason might be that they are mainly interested in the process workflow analysis, as this is the case in process mining. As part of our research we are interested in the reconstruction of a development method that was followed.

### 2.4 Software Process Discovery

In the field of software process discovery [82], the main objective is to automatically derive a formal model of a process from the data that was collected during the execution of a process. In [83], Cook has presented techniques for detecting and characterizing differences between a formal model of a process and its actual execution. Some of the metrics used were string distances between a process model and the actual process data, workflow modelling, and Petri-nets. This is referred as process validation. In order to
recover the actual process on the project, they rely on the execution logs that can be obtained by tracking all the actions that developers and other stakeholders do on the project.

Several works in this field used data stored in software repositories as an evidence of an executed process. In [84–86], Kindler et al. present the algorithms and models that constitute the so-called incremental workflow mining approach, which exploits the user interaction with a revision control system for the semi-automatic derivation of a descriptive process model that faithfully reflects the real process. Their approach consists of three steps. In the first step, they use data from a revision control system and do activity mining to obtain the set of activities and the models of process instances. In this step, they presume that they know the structure of the log, so they do not deal with the reconstruction of the method elements (they call this document types, where each document should be classified to one of the following artifacts: design, code, test results, review) and process instances. In the second step, they take the set of discovered activities to derive the overall process model represented with Petri nets. In the last step, this internal model is transformed into an UML Activity Diagram, which can be shown to the user.

In [87], they improve their work by proposing an approach to define document types. They presume that they know the process instances. For all process instances, they apply all the combinations of document types and then observe which of the execution logs are valid based on the informative model, which contains execution dependencies among the document types/activities. In cases when it is not possible to extract unambiguously the set of types, in spite of checking all possible permutations, the involvement of users is needed. Overall, this approach makes many presumptions, such as that each commit contains only one document or at least one document type and that they are able to tell which commits contain documents of the same type.

Similar work has also been done by Duan and Shen [88]. In this work they suggested a framework to semi-automatically restore the control-flow of the development process from the data stored in the repositories. Their approach consists of three stages. In the first stage they generate an event log from the software repository – they only use data from revision control systems. In the next stage they map the data from repositories to the predefined set of activities (construct, build, readme, test, config). Mapping is based on the predefined keywords (e.g.: if there is a word test in file path or name, this will be mapped to the test activity). In the last step they discovered control-flow of the project and represented it with a Petri net.
Process discovery research area is very closely related to the field of software process mining since they share the same goal – discovery of software processes. Several approaches have been proposed that use data from software repositories to recover as-is processes. Since they are mainly interested in the workflow, they do not focus on the reconstruction of the development method elements to the same level of details as we are in our research. They mainly observe a process as a set of a few high-level activities.

2.5 Other Related Research Fields

There are also other research areas that are less directly related to our work but still worth to mention. This is the research field known as the Software process improvement, and the phenomenon known as the Organizational Patterns.

With the term organizational patterns, we denote structures of relationship, usually in a professional organization, that help the organization achieve its goals. Patterns are collected and organized into pattern languages, which are published as a foundation for process improvement and organizational design. A good explanation of organizational patterns can be found in [89, 90]. Organizational patterns can also be used to capture software development methods [89, 91, 92]. Mainly organizational patterns are still captured and documented manually, but some of the researchers are trying to use data from software repositories to detect bad practices (anti-patterns) [93, 94].

Another research area where our work could contribute is the software process improvement in which process models play a central role in the process improvement cycle. A good review can be found here [95]. Before the improvement and optimization of the development process, it is important that we are able to monitor, control and recover the actual development process that was conducted to produce a product. Software development companies have recognized that the failure to effectively direct the software process is one of the main causes of unsuccessful projects [96]. With our approach, companies will better view into the actual development methods and processes followed on the project. We further anticipate that by the employment of our approach the software companies will be able to raise the maturity of their software development processes (CMMI) to levels 3 (documented), 4 (managed) or even 5 (optimized) [97].
Related Work

2.6 Our Approach Versus Existing Approaches

As presented in the previous sections, there are several research areas that can be considered directly or indirectly related to our work. Most notably, Method Engineering (ME), Mining Software Repositories (MSP), Software Process Mining (SPM) and Software Process Discovery (SPD). Table 2.1 summarizes the main topics these research areas address. The topics in green represent the ones that are followed also in our research.

More detailed comparison is provided in Table 2.2. We selected representative works/publications for each of the related areas and compared them with our research over several dimensions and characteristics. What we can conclude is the following: our approach provides more detailed level of method reconstruction than existing approaches as it focuses on the method from more granular level. Furthermore, to the best of our knowledge, it is the only approach that supports compliancy check against base methods, i.e. the methods that represent real ways of working in organizations, on such detailed level. In this way, base methods are continuously updated to reflect how projects are performed and to document good/bad practices.

<table>
<thead>
<tr>
<th>Table 2.1</th>
<th>Main topics addressed by research areas that are related to our research (Method Engineering (ME), Mining Software Repositories (MSP), Software Process Mining (SPM) and Software Process Discovery (SPD)). The topics in green represent the ones that are followed also in our research.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ME</td>
</tr>
<tr>
<td>Software development method construction &amp; design</td>
<td>Yes</td>
</tr>
<tr>
<td>Software development method tailoring</td>
<td>Yes</td>
</tr>
<tr>
<td>Software development method/process reconstruction</td>
<td>No</td>
</tr>
<tr>
<td>Software development method/process compliancy check</td>
<td>No</td>
</tr>
<tr>
<td>Investigation and mining of data stored in software repositories</td>
<td>No</td>
</tr>
<tr>
<td>Issue/bug/change request process reconstruction &amp; analysis</td>
<td>No</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>MSR</th>
<th>SPM</th>
<th>SPM</th>
<th>OUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>27</td>
<td>82</td>
<td>86</td>
<td>64</td>
</tr>
</tbody>
</table>

Table 2.2

Comparison of our approach with the existing approaches. We have limited our comparison to the most closely related publications (cf. Section 4).
Research Approach
This section gives a description of the research approach.

### 3.1 Research Questions

Our research was motivated by the following research questions:

- **RQ1**: What kind of supporting tools (software repositories) are typically used in software development, what kind of information they enable to capture and how much information developers actually store to software repositories?

- **RQ2**: Does the information stored in software repositories suffice, in respect to the state of the art, for more detailed reconstruction of the software development method followed on a particular project compared to existing approaches?

- **RQ3**: Can we use the reconstructed information to gain information about the project performance?

### 3.2 Data Collection Procedure

In order to answer these research questions one of the main challenges was to obtain representative data. We put a lot of effort in the cooperation with local (Slovenian) companies in order to get access to their software repositories and view into their development practice. This was especially challenging since companies usually treat this as confidential information and are not willing to share these data. For this purpose we do not make any direct relation in this thesis between the data analyzed and the involved companies.

To explore and answer the research questions we collected data from the following five software companies whose business is software development:

- **Marand d.o.o.** is a company with around 100 employees that develops innovative and easy to use healthcare IT products.

- **Optilab d.o.o.** is a small company of about 30 employees. They develop complex information systems for clients from the financial sector, utilities and healthcare.
**Research Approach**

**Ekipa d.o.o.** is a company with over 200 employees. They focus on development of entertaining mobile apps and games.

**Comtrade d.o.o.** A large company with over 500 employees. They develop IT solutions for different industries, including government, financial institutions, healthcare, telecommunication providers.

**Adacta d.o.o.** A company with over 350 professionals that provides support to 400 regional and international clients. They are specialized in developing and implementing business IT solutions and business consulting.

The companies shared their software repositories with us (limited to selected projects only) and provided their personnel (project managers) for qualitative analysis.

For the evaluation step, an additional company joined the research. Its data (software repository) and personnel were used to validate the research findings.

### 3.3 Research Approach

To answer the research questions, the following approach was used:

- **Step 1:** analysis of software repository content: the purpose of this step was to find out what kind of supporting tools the participating companies and open source projects use within software development and, more importantly, what kind of attributes they capture in software repositories. For further research, we assumed that attributes which we found in all repositories (from the involved companies and open source projects), are generic and could be thus found also in any other software repository (i.e. from any other software development company).

- **Step 2:** development of the approach for semi-automatic reconstruction of development method elements from software repositories: the purpose of this step was to develop algorithms (and tools) that will allow us to reconstruct the development method elements from software repositories. The objective was to reconstruct the development method elements that represent valuable information for
project managers and other project team members. Using semi-structured interviews with project managers from the participating companies, we identified the main development method elements of their interest. For the artifacts and activities we then developed algorithms for their reconstruction, while the disciplines and user roles are inferred from the base method.

- Step 3: evaluation: the findings from the first step and the approach developed in the second step were evaluated by involving another company in the research. Firstly, we checked whether our assumption about generic attributes holds in their case and then employed our approach to reconstruct the selected development method elements from the repository on a recently finished project. Finally, we discussed the accuracy and usefulness of the reconstructed software development elements with their project manager.
Analysis of Software Repositories
As described in the research approach (see step I), we first need to identify the kinds of attributes that are typically stored in software repositories (irrespective of project characteristics, technology and tools). Moreover, we need to analyze the relationships between the key elements of software repositories (commits, issues and users) to see if these are clear enough to infer other elements of software development methods (e.g. activities, deliverables, etc.) that we are interested in. In an ideal case, we expect that there is a one to one relationship between issues, commits and users, i.e. each commit in a revision control system is linked to exactly one issue in an issue tracking system and both are linked to a single user only. Furthermore, we expect that user accounts of different software repository tools that pertain to a same person can be consolidated.

To analyze software repositories, as a first step we asked participating companies to provide us access to their development environments or just give us snapshots of their software repositories. This was not easy to achieve due to the privacy and security issues, but eventually we got enough data to get a good picture of what they use and collect. As we expected, the companies were quite similar in terms of the type of tasks for which they were using computerized support (e.g. revision control, issue/bug tracking, document management, etc.) but differed to a certain extent in the actual tools they were using. Among the tools that we found, the most common tools used were:

- Jira, Bugzilla or DevTrack for issue/bug tracking (ITS)
- Subversion or Git for revision control (RCS)
- Sharepoint or LogicalDOC as a document management system (DMS)

Additionally, some companies used tools for other tasks, such as for managing code reviews (e.g. Crucible, Reiveld) or for managing team collaboration (e.g. Slack, Confluence, Skype). But since these tasks did not have computerized support in all companies and in some cases data cannot be obtained due to the privacy issues, we did not analyze them further.

As a next step, to make our conclusions more generalizable, we also gathered data from three open source projects: MongoDB, Spring Framework, and Hibernate ORM. All three projects use Jira as issue tracking system and Git as a revision control system.
4.1 Information Stored in Software Repositories

Software repositories store rich and valuable information about the development process and underlying development methods. In this section we analyze information stored in software repositories used by companies and open source projects that we analyzed.

Revision control system stores information about the project revisions. Each revision has information about changed (added, modified, deleted) files, along with who made the change, why they made it, and when they made it. This information can be exacted from all main revision control systems. In Table 4.1 we compare and relate data from Subversion and Git, which are the most common revision control systems.

Table 4.1
Data stored in the Revision Control Systems

<table>
<thead>
<tr>
<th></th>
<th>Subversion</th>
<th>Git</th>
</tr>
</thead>
<tbody>
<tr>
<td>Revision Id</td>
<td>Revision number</td>
<td>Sha</td>
</tr>
<tr>
<td>Message</td>
<td>Message</td>
<td>Message</td>
</tr>
<tr>
<td>User</td>
<td>User</td>
<td>Author</td>
</tr>
<tr>
<td>Timestamp</td>
<td>Date</td>
<td>Author date</td>
</tr>
<tr>
<td>Commiter</td>
<td>–</td>
<td>Commiter</td>
</tr>
<tr>
<td>Commiter timestamp</td>
<td>–</td>
<td>Commiter date</td>
</tr>
<tr>
<td>Changes</td>
<td>Changed types (change type, file)</td>
<td></td>
</tr>
</tbody>
</table>

Issue/bug tracking systems store information about issues (tasks) that were performed on a project. For each issue we can extract detailed information, such as who reported the issue and when, issue type (e.g. bug, feature, task), priority, description, title, estimated time, labels, attachments, relations to other issues, etc. Furthermore, each issue is assigned to a developer who is responsible for its resolution. When working on an issue, a developer changes its status (open → in progress, in progress → resolved) based on a predefined workflow. Developers can also report on the time that they spent working on the issue (worklogs) and give comments on it. The comparison of the main attributes that can be extracted from the issue tracking systems (Jira, Bugzilla), are presented in Table 4.2. All the examined issue tracking systems allow to add new attributes, specific to the needs of a project or a company.
Table 4.2
Data stored in the Issue Tracking Systems

<table>
<thead>
<tr>
<th>Jira</th>
<th>Bugzilla</th>
</tr>
</thead>
<tbody>
<tr>
<td>Issue Id</td>
<td>Key</td>
</tr>
<tr>
<td>Issue type</td>
<td>Type</td>
</tr>
<tr>
<td>Title</td>
<td>Summary</td>
</tr>
<tr>
<td>Status</td>
<td>Status</td>
</tr>
<tr>
<td>Assignee</td>
<td>Assignee</td>
</tr>
<tr>
<td>Reporter</td>
<td>Reporter</td>
</tr>
<tr>
<td>Created</td>
<td>Create date</td>
</tr>
<tr>
<td>Description</td>
<td>Description</td>
</tr>
<tr>
<td>Priority</td>
<td>Priority</td>
</tr>
<tr>
<td>Components</td>
<td>Components</td>
</tr>
<tr>
<td>Estimate</td>
<td>Original estimate</td>
</tr>
<tr>
<td>Attachments</td>
<td>Attachments</td>
</tr>
<tr>
<td>Labels</td>
<td>Labels</td>
</tr>
<tr>
<td>Links</td>
<td>Links</td>
</tr>
<tr>
<td>Worklogs</td>
<td>Worklogs</td>
</tr>
<tr>
<td>Resolution</td>
<td>Resolution</td>
</tr>
<tr>
<td>Comments</td>
<td>Comments</td>
</tr>
</tbody>
</table>

Document management systems store, as the name implies, a set of documents. In our case we treat a document management system the same as a revision control system. Basically, they share the same purpose except to document management systems (e.g. SharePoint), which specializes in storing and working with documents, while the revision control systems (e.g. Git) are more specialized in working with the source code. Some companies use revision control system to store both, source code and documents.
In case of a document management system, each time a document is changed a new revision is created. Each revision has information about the changes made (added, modified, deleted documents), along with who made the change (user), why they made it (message), and when they made it (timestamp).

For each company and tool, we examined what kind of data they actually store in their databases or logs. In Figure 4.1, the set of attributes that we were able to find in software repositories of all five participating companies as well as in software repositories of all three open source projects is listed.

![Figure 4.1](image-url)
4.2 Links Between Software Repositories

The next step was to check how well the data from different tools comprising software repositories can be linked together. Remember that a software repository is not a standalone physical database but rather represents a logical view on several databases and logs from different systems. In our study, we found that important logical links exist between issues and commits, which both carry important information and are kept track of in software development. While an issue represents a problem or associated tasks that need to be carried out in order to solve a specific problem, a commit refers to changes on specific files that are a result of solving the problem or task and are put back into the repository. Issues and commits are, however, managed in different systems and thus not necessarily linked. The link can be established if the commit message carries enough information so that we can identify which issue it is linked with. Figure 4.2 (red border) shows an example of a link between an issue (Jira) and a commit (Github). The link can be extracted from a commit message, which contains information about the issue key. For unlinked commits and issues, techniques such as Frlink can be used [98]. Let us also note that linking commits with issues is a good development practice that has been practiced in open source community for a long time [26] and should be enforced by the companies that want to raise the quality of their development processes.

4.2.1 User Resolution

Another challenge for establishing links between data collected through various systems into a software repository, is to link user accounts created in these systems that refer to the same user. Figure 4.2 (blue border) shows an example where the same person was assigned to an issue in Jira and has performed a commit in the Github. However, for each tool different user account that belongs to the same person was used. This is almost always the case, as software repositories usually comprise tools of different vendors and the single-sign-on option is not available. Since we want to link the data and keep track of which project member in the company has done what, it is important that we are able to link the user accounts from different software repositories belonging to the same project member (Figure 4.3).

For this purpose, various existing entity resolution and identity merge algorithms can be used, which compare all the user data that are available (username, email, first and last name, etc.). In our case, we use the one published by Goeminne and Mens [99].
4.3 Real Software Repositories Data Analysis

In this section we will analyze software repositories from real projects to see what information is actually stored and what information can be extracted. In the analyses we used data from three open source projects and two projects provided by the participating companies. A short description of each project is provided below:
MongoDB was started by the software company 10gen (now MongoDB Inc.) in October 2007. The company shifted to an open source development model in 2009. On Core Server project they use Jira since 2009, and created 15392 issues till the end of 2014 (created date). As a revision control system they use Git(hub), and till the end of 2014 it stored 28374 commits (author date). As a review tool they use Rietveld. The first two are publicly accessible, but the review tool is only available to the employees of MongoDB.

Spring Framework is open source application framework and inversion of control container for the Java platform. Since the end of 2003 they use Jira as issue tracking system, and have till the end of 2014 created 12467 issues. Since 2008 they use Git(hub) as a revision control system and till the end of 2014 it stored 9696 commits.

Hibernate ORM is object/Relational Mapping (ORM) framework. Since the end of 2003 they use Jira as issue tracking system, and have till the end of 2014 created
9419 issues. Since 2007 they use Git(hub) as a revision control system and till the end of 2014 it stored 5673 commits.

**Company I** provided us data from a large project, which they started developing in 2007 and has been deployed to over 15 companies. As an issue tracking system, since 2008, they use Jira, which had by the end of 2014 (created date) 13389 issues, and as a revision control system they use Subversion, which stored 18571 commits at the end of 2014. They use agile approach when managing projects (Scrum).

**Company II** provided us data from a medium sized project. They started the project in 2008. They have been using Jira since middle of 2010, and have created 5148 issues by the end of 2014. As a revision control system they use Subversion, which stored 13735 commits at the end of 2014. They use agile approach when managing projects (Scrum).

### 4.3.1 Import of Data

To perform the analysis, we first had to import all the data from different software repositories. Since in our case we used data from Jira, Subversion and Github, we developed a tool to support import from all these repositories. After importing the data, information about issues was extracted from the commit messages. For this purpose we used regular expressions. Next, we linked user accounts from different tools that belong to the same user. We linked users by comparing their names, usernames and emails.

### 4.3.2 Analysis and Results

First we analyzed how many commits can be linked to issues, by extracting information on the issues’ ids from the corresponding commit messages. It is important to mention that we did not give any special attention to the fact that developers enter information about issue id manually, so in many cases a developer can make a typing mistake and issue id might not be extracted. The results are represented in Figure 4.4. An interesting finding is that the number of commits that can be linked to corresponding issues, is rising along with the projects’ duration. This is true for open source projects as well as for commercial projects. Surprisingly, in some cases open source projects get even better results than commercial projects although we expected that it would be the other way around. In 2014, 84.0% (on average) of commits were linked to an issue.
Next we analyzed how many commits linked to an issue are linked to exactly one issue. This is important since we want to link each commit to exactly one issue and in this way provide more accurate reconstruction. The results are presented in Figure 4.5. As we can see, our expectations was confirmed. On average 96.3% of commits that we were able to link with issues were linked with exactly one issue.
Next we analyzed how many of issues can be linked to at least one commit. In the analysis we excluded issues that were not resolved (issue status = closed or resolved; issue resolution = fixed). The results are shown in Figure 4.6.
As we can see, there were 67.7% (on average) issues that we were able to link with a commit. Surprisingly, the dependency between issues and commits was better on open source projects than on the selected commercial projects. However, on the other side, on the commercial projects there are many more issues that are linked to other activities and not just to program code, which seems to be the case in open source projects. Based on the results we can conclude that on the commercial projects developers are not so strict and consistent in updating issue statuses. It is also important to mention that we did not filter out issues, which were of type question, meeting, epic, etc. The distributions of issue types per project and issue resolutions per project are represented in tables 4.3 and 4.4 respectively.
Table 4.3
Issue types and their proportion on the projects

<table>
<thead>
<tr>
<th>Issue type</th>
<th>Mongo</th>
<th>Spring</th>
<th>Hibernate</th>
<th>Com. I</th>
<th>Com. II</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bug</td>
<td>63.6</td>
<td>43.7</td>
<td>63.9</td>
<td>35.0</td>
<td>10.3</td>
</tr>
<tr>
<td>New Feature</td>
<td>4.8</td>
<td>8.3</td>
<td>5.6</td>
<td>4.1</td>
<td>–</td>
</tr>
<tr>
<td>Task</td>
<td>4.0</td>
<td>7.5</td>
<td>7.9</td>
<td>41.4</td>
<td>7.3</td>
</tr>
<tr>
<td>Improvement</td>
<td>20.4</td>
<td>37.4</td>
<td>15.5</td>
<td>5.8</td>
<td>–</td>
</tr>
<tr>
<td>Sub-Task</td>
<td>3.0</td>
<td>1.8</td>
<td>3.6</td>
<td>10.7</td>
<td>68.5</td>
</tr>
<tr>
<td>Question</td>
<td>4.2</td>
<td>–</td>
<td>–</td>
<td>0.5</td>
<td>–</td>
</tr>
<tr>
<td>Story</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>13.9</td>
</tr>
<tr>
<td>Refactoring</td>
<td>–</td>
<td>0.6</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Pruning</td>
<td>–</td>
<td>0.1</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Backport</td>
<td>–</td>
<td>0.6</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Patch</td>
<td>–</td>
<td>–</td>
<td>2.8</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Deprecation</td>
<td>–</td>
<td>–</td>
<td>0.4</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Remove Feature</td>
<td>–</td>
<td>–</td>
<td>0.1</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Technical Task</td>
<td>–</td>
<td>–</td>
<td>0.1</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Meeting</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>0.2</td>
<td>–</td>
</tr>
<tr>
<td>Sub-Bug</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>1.0</td>
<td>–</td>
</tr>
<tr>
<td>Sub-Improvement</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>0.2</td>
<td>–</td>
</tr>
<tr>
<td>Sub-New Feature</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>0.2</td>
<td>–</td>
</tr>
<tr>
<td>Service Request</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>0.8</td>
<td>–</td>
</tr>
</tbody>
</table>
In case of the MongoDB we manually checked 274 issues (from 2014), which we were not able to link to a corresponding commit. As it turned out, for additional 120 issues we were able to extract information about the corresponding commit from the issue’s comments. Thus 92.7% of all issues from 2014 were linked to a corresponding commit. Furthermore, we noticed that some of the unlinked issues were linked to a specific branch or to MongoDB enterprise components, which are not publicly available. In our case we only included commits from the master branch.

Finally, we checked how many issues that were labeled as closed and fixed and we were able to link to commits, were resolved by one and not many developers. In some cases,
several developers can contribute to resolution of a single issue. First and the most important thing was to relate the same developers in a revision control system. For example, in case of MongoDB, we identified 248 unique users out of 434 users all together, and in case of a Company I we got 119 unique users from 193 users all together. The next thing was to define how we know that each issue was resolved by one developer. For each issue we checked the users of all the commits linked to an issue. If there was only one user, we marked this issue as solved by one user. The results are presented in Figure 4.7. As we can see, in average there is 85.6% of issues that were resolved by one user. Again, this was found true for both, open source projects as well as for the commercial projects.

Figure 4.7

Issues (closed, resolved, fixed, have commit) that are resolved by only one developer.
4.4 Chapter Summary

In this chapter we identified the kinds of attributes that are typically stored in software repositories. Moreover, we analyzed the relationships between the key elements of software repositories to see if these are clear enough to infer other elements of software development methods that we are interested in. To gather the required information we cooperated with five Slovenian companies as well as looked at the data from the open source projects.

Information stored in software repositories  We analyzed the information stored in software repositories typically used by companies and open source projects that we examined. This contained revision control system (e.g., Git) + (optionally) document management system (e.g., Sharepoint) and issue tracking system (e.g., Jira). We analyzed what attributes are available and stored in different software repositories and the final list of attributes that we were able to find in all is presented in Figure 4.1.

Links between software repositories  In the next step we checked how well the data from different software repositories can be linked together. Mainly we were interested in how to link issues and commits and how to link user accounts created in these systems that belong to the same user.

Real software repositories data analysis  To evaluate how well we are able to link data from different repositories we have performed an evaluation of the real data. The results were promising and gave us motivation to proceed with our reconstruction approach.

In this chapter we answered to the RQ1 (What kind of supporting tools (software repositories) are typically used in software development, what kind of information they enable to capture and how much information developers actually store to software repositories?). To answer this research question we analyzed data from several commercial companies from Slovenia as well as several open source projects. As it turned out the most typical software repositories present on more or less all projects were revision control system (e.g., Git) + document management system (e.g., Sharepoint) and issue tracking system (e.g., Jira). We have analyzed what kind of data they enable to capture and what kind of data are actually stored during project performance. The final set of
attributes can be seen in Figure 4.1. Furthermore, we were interested in how well we can link the data from different software repositories. Mainly we were interested in how well we can link issues and commits. As it turned out the results were promising and gave us good support for the reconstruction approach.
The Reconstruction of a Software Development Method
The main objective of our research was to enable reconstruction of more detailed information about the development methods used on projects than existing approaches enable, and to do this without any substantial involvement of developers. The existing approaches [61, 62, 74, 78] focus on the reconstruction of disciplines (i.e. they are able to tell how much effort was spent on analysis, design etc. or how these disciplines were following one another) or they go into details on the development phase only (i.e. by focusing on the issue lifecycle). In contrast, our goal is to focus on the whole project, and not just the development phase and to reconstruct more than just the disciplines.

In further text we use two terms for two different kinds of methods: base method and project-specific method. Base methods are company specific, i.e. they describe how companies perform their projects. They contain all sorts of methods and method elements that can be used on projects. In a particular company, a base method acts as a template that can be configured based on characteristics of a particular project to form a specific method, suitable to that project. In the reconstruction process we thus deal with either base methods or project specific methods, depending on how much data we take into consideration.

In this section we first describe which method elements were chosen by the participating companies as being of their interest and thus to be reconstructed from software repositories. The steps of the reconstruction process are shown in Figure 5.1.

5.1 Metamodel of Software Development Method

Development methods can be described by a number of different concepts, including phases, disciplines, deliverables, activities, techniques, examples, roles, experiences, life cycles, tools, etc. A number of different metamodels exist that underpin existing development methods [100]. For the purpose of our research, we followed the Software & Systems Process Engineering Metamodel (SPEM) metamodel [101]. Its basics, including the separation of method and process concepts, were briefly described to the participating companies. Afterwards, the companies were asked to identify the main method meta elements that they would be interested in reconstructing from their software repositories; similarly as suggested in [94] and [93]. The final selection, which was influenced also by the data that is actually captured in software repositories, included the following method meta elements:
Disciplines: a discipline presents a set of activities that are closely related in the sense that they all contribute to the same overall goal (e.g. Analysis, Design, Implementation).

Activities: an activity presents a general unit of work assignable to a specific performer (Develop a use case, Design GUI, etc.). As a result of an activity, different deliv-
erables (artifacts) can be produced.

**User roles:** a user role is responsible for performing activities and producing the required artifacts (e.g. Developer, Analyst, Architect, etc.). Note that several project members can be assigned to a single user role.

**Artifacts:** an artifact is a result produced as part of performing a particular activity (e.g. Source code, Unit test).

The metamodel is represented in Figure 5.2. Each discipline consists of one or many activities while an activity belongs to exactly one discipline. An activity can be connected to none, one or many artifacts and an artifact with one or many activities. For each activity, there is exactly one user role assigned, but for a particular user role there might be several activities that the user role is responsible for. The recursive relationship on the meta element Activity designates that activities are dependent on each other, which is due to the fact that they need to be performed in a certain order.

![Figure 5.2](image)

*Figure 5.2*

*Software development method metamodel and its connection with software repository concepts.*
Figure 5.2 also indicates the relationships of the metamodel elements with the software repository concepts. The three most important concepts, originating from a software repository, are a file, an issue, and a user. The concept file designates a physical file that is stored in a revision control system or document management system, the concept issue represents an issue from an issue tracking system, and finally the concept user denotes user accounts from any of the software repository systems. The meaning of the relationships among metamodel elements and software repository concepts is as follows:

**Artifacts** are in a relationship with files (stored in software repository) that represent these artifacts. Each artifact can be related to none, one or several files while a file belongs to exactly one artifact.

**Activities** are in a relationship with issues. Each issue requires a certain amount of work to be resolved. This work might be represented as an activity. A good development practice is that each issue is connected to exactly one activity while activities might resolve several issues at once.

**User roles** are in a relationship with users. Users represent project team members with user accounts in a software repository. Several users can be assigned to a particular user role and vice versa, a particular user can play more than just a single user role in an observed project.

Our metamodel consists of four metaelements, which were selected based on the interest shown by the project managers inside the companies as well as by taking into account what kind of data we can actually obtain from the commonly used software repositories. The main reason why we limited our approach to these four metaelements was to make our approach more generalizable and not too dependent on the specific data stored by a particular company/project. The metamodel, can always be expanded with additional metaelements, which can be obtained automatically from the software repositories or added to the base method manually. For example, one of the popular modern software development techniques is Continuous Integration/Delivery/Deployment [102]. We could manually add this as a technique metaelement linked to the corresponding activities related to the implementation, deployment and test. Furthermore, we could include Continuous integration server (e.g. Jenkins, Bamboo) as a software repository and upgrade base method with new activities and artifacts (e.g. Deployment
5.2 Construction of a Base Method

One of the concepts that plays an important role in our approach is the so called base method. With the base method, we denote the set of method elements that are typically used in a particular company when performing development projects. Taking into account the metamodel described in Section 5.1, the base method of a company includes its typical disciplines, activities, user roles, artifacts and relations among them. The intensity of individual activities and the sequence of their performance is however not described with the base method as this depends on each particular project settings and its characteristics.

The construction of a base method is a preliminary step before the reconstruction of the project specific development method. Since base methods are company specific, this needs to be done individually for each company. Base methods can be constructed in different ways. One way is to acquire the required information from the company, i.e. from their project managers or documentation, and then construct the company’s base method manually. This is typically the case for the light and agile methods (like the Scrum [19] or Kanban [20], for example), since these methods only describe a lightweight project management framework and leave the specification of the development practices to the company and development teams. Alternatively, we can start with a base method that corresponds to some commercial software development methodology, such as for example RUP [67] or similar. The base method is then updated every time a new project-specific method is reconstructed and eventually becomes a good representation of the company’s way of developing software.

Figure 5.3 represents an example of a base method constructed by analyzing the company’s documentation. We did that together with one of the company’s project managers. Based on the acquired information, we were able to construct the company’s base method consisting of 5 disciplines, 15 activities, 23 artifacts, 8 user roles, and the relations among them. We also captured 14 rules (e.g. Financial Calculation - If the project is small then Financial Calculation is not required; If the project is medium or large then
Financial Calculation is required). More details about the construction are presented in the Section

Figure 5.3
The company’s base method draft manually constructed out of the company’s documentation together with one of the company’s project managers. It comprises 5 disciplines, 15 activities, 23 artifacts, 8 user roles, and 14 rules. Examples of a rule: (a) Financial Calculation - If the project is small then Financial Calculation is not required; If the project is medium or large then Financial Calculation is required, or (b) If the project is small then Benefits Review Plan is not required; If the project is medium or large then Benefits Review Plan is optional.
5.3 Reconstruction of Software Development Method Elements

Once the base method is defined, we use data from software repositories to reconstruct activities, artifacts, user roles, and disciplines. In this section we describe how this can be done. In the guidelines that follow, we take into account the findings on the data that software repositories include and how well this data is linked (see Chapter 4).

The reconstruction of the method elements can be done incrementally as issues are processed independently of one another. This is important as it allows us to monitor, control and reconstruct a development method as project is performed. For the reconstruction we only process the issues that are relevant for the development process. We exclude issues that have a specific resolution status and type. These have to be defined separately for each company. For example, we exclude all the issues that are duplicates of another issue, issues that have been canceled, that cannot be reproduced and issues of a type epic, story, etc. As an issue of a type Epic, for example, in many companies presents a high-level issue, which is later split into several sub-issues. Furthermore, we can link a specific issue type directly to the activity, so that these issues are not included in the reconstruction process (e.g., we link all the issues of a type bug to the activity Resolve bugs).

To reconstruct a project’s specific method, the following three steps are carried out: (1) we gather all files from commits that are linked to the issue and classify them into artifacts, (2) based on the artifacts, we define the activity, and (3) based on the retrieved activities, we identify corresponding roles and disciplines. Below we present each step in more details.

5.3.1 Reconstructing Artifacts

Artifacts are reconstructed from files that were committed to the repository. This is done immediately after an issue is being resolved. We check the repository and retrieve all files that were committed as a consequence of resolving this issue. Then we infer from the files (by checking their names, file types and if necessary also the file content) which artifacts from the base method they represent. Techniques that can be used for matching files to artifacts are numerous. Machine learning algorithms are useful when we have data to learn from, i.e. software repositories from past projects. In this case, we create a classifier which we then use for matching. If this is not available, we acquire additional information from the company employees so that matching can be done. The algorithm used
in such cases is described in Section 5.3.4. The schematic presentation of the approach is presented in Figure 5.4.

Figure 5.4
Schematics of reconstruction of artifacts.

5.3.2 Reconstructing Activities

After we reconstruct the artifacts as a result of resolving an issue, we go further and check which activities are connected to these particular artifacts. We do this by checking the base method where these relationships are defined. An example is presented in Figure 5.5 (elements with the red border).

In most cases activities are connected to one artifact only, thus the reconstruction of the correct activity is not a problem. The involvement of developers is only required in rare cases a) when these relationships are not one to one (several activities might be responsible for the creation of one particular artifact) or b) when we have no artifact which we could use to infer the corresponding activity. This happens when we deal with an issue that did not commit any file to the repository. In this case we need to involve a developer to select the correct activity and mark the issue as a bad practice, or to cre-
ate/update an activity in the base method. In case the developer decides to create a new one, he/she also needs to provide the information on the user role and discipline.

In cases when issues are without any physical artifacts stored in the revision control system or document management system, we can use the algorithm described in Section 5.3.4.

5.3.3 Reconstructing User Roles and Disciplines

Similarly, we also reconstruct user roles and disciplines. Since we already know the activity name, we simply check the base method to retrieve also the names of the associated user roles and disciplines. Furthermore, for each retrieved user role we also retrieve the users (user accounts) that were assigned to this particular activity. An example is presented in Figure 5.5 (elements with the blue border).

5.3.4 Algorithm for Reconstructing Artifacts and Activities

For matching files to artifacts and issues to activities we use the algorithm that is described in Table 5.2.

For the algorithm to work, the first step is to go through the base method and capture keywords that best describe each artifact. In addition, we capture file types that represent

Figure 5.5
We use the information about the artifacts created as part of an issue to recover the correct activity corresponding to a given issue (elements with the red border). Once we have information about the activity we look up in the base method to find out what is the user role that is responsible for this activity and to which discipline it belongs (elements with the blue border).
the format in which a specific artifact is usually created. The example of the keywords and file types for the artifact *Functional specification* is presented in Table 5.1. Next, we capture keywords for the activities without artifacts.

**Table 5.1**

| Keywords requirement, functional requirement, non-functional requirement, usability, scalability |
| File types doc, docx, rtf, txt, pdf |

Such manual acquisition of this information is only required if we do not have any data to learn from. If this data is available, i.e. we have access to software repositories of finished projects, then techniques, such as Bag of Words \[103\], TF-IDF \[104\] or similar can be used to automatically acquire this information.

The algorithm to match files to artifacts and issues to activities is as follows: for each resolved issue \(I\), we find all connected commits \(C\). For each such commit \(C\), we classify each committed file \(F\) to an artifact \(A\) from the base method \(BM\). We try to do that based on the file types. If several artifacts (artifact list \(AL\)) contain the same file type, we calculate individual artifacts’ weights. An artifact weight \(w\) for an artifact \(A\) tells what is the likelihood that the file \(F\) represents the artifact \(A\) from the artifact list \(AL\). The higher the weight, the higher the likelihood. The weight \(w\) is calculated as a sum of TF-IDF values. The TF-IDF metric \(\text{tfidf} = \text{tf}(K,A) \times \text{idf}(K,AL)\) is calculated as a product between frequency of the keyword \(K\) in the file \(F\) \(\text{tf}(K,F) = f_{KF}\) and the logarithm of the ratio between the number of all artifacts \(A\) in the artifact list \(AL\) and the number of these artifacts from the artifact list in which the keyword \(K\) appears \(\text{idf}(K,AL) = \log(|AL| + |K \in AL|))\). Once the committed files have been successfully classified to an artifact, we check in the base method which is the activity that is responsible for the delivery of this artifact.

If the issue under analysis cannot be connected to any commit and we cannot identify the corresponding activity over connected artifacts we reconstruct the activity directly from the base method by employing a very similar approach (see lines 14-24 in Algorithm 5.2). Instead of searching for artifact keywords in committed files, we search for activity keywords in the issue title and description. These are two attributes that we can find in all issue tracking systems. For clarity reasons (to avoid duplicate lines), this part is not
shown in the algorithm.

The algorithm is represented below. It uses three data structures:

**Matrix WeightKA:** a two-dimensional matrix that tells for each keyword $K$ and artifact $A$ what is the likelihood that $K$ represents $A$. The likelihood of $K$ representing $A$ is calculated using TF-IDF.

**List Artifacts:** a list of artifacts from the BM that contain a specific file type in their file types set.

**List Activities:** a list of activities from the BM that are in BM linked to the artifacts reconstructed during the classification of files linked to a specific issue.

Table 5.2
Algorithm to reconstruct Artifacts and Activities

<table>
<thead>
<tr>
<th>Algorithm to reconstruct Artifacts and Activities</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>INPUT:</strong></td>
</tr>
<tr>
<td><code>List ResolvedIssues</code> //ordered by resolved time ASC;</td>
</tr>
<tr>
<td><code>BaseMethod BM</code></td>
</tr>
<tr>
<td><strong>OUTPUT:</strong></td>
</tr>
<tr>
<td>// files are classified to artifacts</td>
</tr>
<tr>
<td>// issues are classified to activities</td>
</tr>
<tr>
<td>// relations between activities and artifacts</td>
</tr>
<tr>
<td>1. <strong>Matrix WeightKA</strong>;</td>
</tr>
<tr>
<td>2. <strong>Set IssueArtifacts</strong>;</td>
</tr>
<tr>
<td>3. for each issue I in ResolvedIssues</td>
</tr>
<tr>
<td>4. for each commit C in ConnectedCommits</td>
</tr>
<tr>
<td>5. for each file F in C.files</td>
</tr>
<tr>
<td>6. if F.type == IgnoreFileType then continue;</td>
</tr>
<tr>
<td>7. <code>List Artifacts = artifactsByFileType(F.type, BM)</code></td>
</tr>
<tr>
<td>8. if Artifacts.size == 0 then</td>
</tr>
<tr>
<td>9. // new type → ask project member, update BM</td>
</tr>
<tr>
<td>10. else if Artifacts.size == 1 then</td>
</tr>
<tr>
<td>11. <code>Classify(F, Artifacts[0])</code></td>
</tr>
</tbody>
</table>
The Reconstruction of a Software Development Method

5.4 Limitations and prerequisites

There are some limitations and prerequisites of our approach, which are presented in more details below:

- In order to make our approach more generalizable, we intentionally limited our reconstruction on the data that can be obtained from the main software repositories present on most major projects (for more details see Chapter 4). This was also the reason that we limited our reconstruction to the four main metaelements. Metamodel can always be extended with new metaelements, which can be recon-
structured from other software repositories or added manually.

- One important precondition for our approach to work is that we are able to extract links between issues and commits. Since this is nowadays a common practice for open source as well as for commercial projects, we believe that this is not a huge limitation. Furthermore, there exist approaches that support automatic reconstruction of links between issues and commits, but in that case our reconstruction will be limited with the accuracy of the reconstructed links.

- With our approach we are able to reconstruct only those development method elements that can be reconstructed from the data available in software repositories. This means that with our approach we are not able to reconstruct an artifact that was created without leaving any trail in a revision control or a document management system. The same goes for activities. If there is no issue related to a specific activity in an issue tracking system, we have no information as to what actually happened. In this case this method element will be either marked as missing or not found. Since more or less all activities are nowadays supported and tracked by computers, we believe that this is not a big limitation.

5.5 Chapter Summary

In this chapter we present the approach that enables the reconstruction of software development method using the data available in software repositories and requires minimal involvement of developers. Comparing to the existing approaches, the proposed approach supports the reconstruction on the higher level of details, which beside disciplines includes also artifacts, activities, and user roles.

Metamodel of software development method When defining the metamodel, we followed the SPEM metamodel [101]. The final selection of meta elements was done based on the information available in software repositories and the interest of participating companies. The metamodel includes the following meta elements: discipline, activity, artifact, and user role.

Construction of a base method The construction of a base method is a preliminary step before the reconstruction of a project specific method and needs to be done only
once per company. With the base method we denote the set of elements that are typically used in a particular company when performing development projects. The base method can be constructed by using the existing documentation, existing off-the-shelf methods or by gathering the information from project members through the interviews.

**Reconstruction of method elements** Before the reconstruction of method elements we first have to import and link the data from different software repositories. Next, we use the prepared data from software repositories and try to map them to the elements from the base method. The reconstruction is done in an incremental way, issue by issue. Once issue is resolved, we check its commits and try to classify the changed files to artifacts. Based on the artifacts, if there are any, we try to reconstruct information about the activity. Each time, when we are not able to reconstruct artifacts or activity with sufficient confidence, we involve project members to provide new knowledge and supplement the base method. Discipline and user role are implicitly reconstructed from the base method.

**Limitations and prerequisites** There are several limitations and prerequisites of our approach. The most important precondition of the approach is that we are able to link commits and issues. Since this is nowadays a common practice this should not be a huge limitation. Furthermore, with our approach we are only able to reconstruct those development method elements that leave trail in software repositories.

In this chapter we answered to the **RQ2 (Does the information stored in software repositories suffice, in respect to the state of the art, for more detailed reconstruction of the software development method followed on a particular project compared to existing approaches?)** by showing that the information stored in software repositories suffice for more detailed reconstruction of the software development method followed on a particular project compared to existing approaches. We proposed an approach that besides disciplines also supports the reconstruction of activities, artifacts and user roles. Based on the findings from the RQ1 the approach is limited to only exploiting data from software repositories that are typically used in software development (revision control system + document management system and issue tracking system). With this limitation we wanted to make the approach more general and useable in practice. Furthermore
we showed how the approach supports detection of deviations from the base method. Whenever we are not able to reconstruct the method elements from the available data, we mark this as a possible deviation and ask project members to provide new knowledge.
Software Development Process Analysis
The reconstructed development method elements in combination with other information available from the software repositories (e.g., worklogs) provide a basis for different project analyses. We can perform post-mortem analysis (as described in the next section), or an on-the-fly analysis that is possible due to iterative reconstruction of the development process, i.e., one issue at a time. This allows us to monitor the development process in real time and to take action when the process deviates from the plan. Below we present in more details some of the possible project analyses, and explain how we can monitor, control and guide the development process, and provide real time reconstruction and analysis. All figures in this section are created using the data provided by the participating company. The same data appears also in the evaluation phase (presented in the previous section).

### 6.1 Post-mortem Analysis

Post-mortem analysis is when we apply our approach on a finished project. There are several interesting things to observe about the project performance.

#### 6.1.1 Time Spent per Discipline/Activity/User role (daily/aggregated)

Using the information about the worklogs allows us to observe on a daily basis how much time (effort) was spent per discipline, activity, or user role. This tells us, for instance, for which part of the project the most time was spent. Furthermore, we also have a possibility to analyze the total time spent on a particular discipline, activity, or user role.

Once we have data from several projects, we can compare this information and extract if there is any correlation between successful and less successful projects and how a project of a specific type has performed on average. For example, is it important how much time developers have invested in the activities from the discipline analysis & design and how much in the activities from the discipline test? If they did not invest much time in the analysis & design in the beginning of the project, then later they might have to come back to that.

In Figure 6.1, we present how the relative emphasis of each discipline has been changing throughout the project. At the beginning of the project, the most intense was the discipline Analysis & Design, since the company and the customer had to agree what functional requirements should be supported by the new application. After the cus-
Customer confirmed the functional specification, wireframes, design and project initiation document, the project moved to the discipline *Implementation*. At the end of the implementation phase, the discipline *Test* gained emphasis with the main goal to validate the solution. As part of this activity, the developers also performed stress and security tests. A considerable amount of testing was also performed by the customer, who did not report the time spent (this tells that the discipline testing actually has a bigger share of the project effort). As part of the *Deployment* discipline, the developers’ activities were mainly related to the preparation of documentation and to the system administration. The number of locations to which the system had to be deployed was low and there were no specific activities that had to be done for the installation. This explains why the discipline *Deployment* is not so intense at the end of the project as one would generally expect. The supporting discipline *Project Management* was running throughout the whole project.

Figure 6.1
*Intensity of particular disciplines.*

Figure 6.2 shows how much time was daily spent per particular user role. In the be-
ginning of the project, most of the time was spent by the team member having the role of Analyst. Once the wireframes were confirmed, the focus went on to the UI Designer and Front-end Developer roles. They had to prepare a design and slice it to HTML and CSS code, which is needed by developers. At the same time team members having the role of Developer spent some time on preparing the development environment. After that a considerable amount of work was performed by the developers. Throughout the
whole project, two stakeholders were also actively involved, their role was to clarify requirements, observe progress, and provide feedback when needed. Their effort is however not logged by the company system.

Figure 6.3 presents how much time was daily spent per particular activity. Similarly as in case of the disciplines and user roles, we can observe how the relative emphasis of each activity is changing through time. Here we again analyze how the time has been spent per activity and spot out special patterns. At the beginning of the project, most of the activities related to the discipline Analysis & Design were performed. Later on, most of the work was done as part of the activity Develop. As expected, the activities related to Project Management were present throughout the whole project.

6.1.2 Actual vs. Estimated Time

Before acquiring a project, companies typically prepare an offer for which they also estimate the time that will be needed for a particular activity on the project in order to estimate the total costs of the project. In case the methodology Scrum is employed, developers also estimate the time needed to perform a task that is assigned to them. So, in many cases we could benefit from having information about how in the past the estimated time and the actual time differed. The comparison of the actual and estimated time allows us to detect for which disciplines/activities/user roles developers spent more time than expected. This information is very important to project managers and can help them to make better estimations on new projects.

In our case, the observed company does not rely on the planning poker to estimate the time needed to perform each task, which is one of the Scrum principles. However, in order to prepare an offer and to estimate the project costs, they do an internal financial calculation as part of which they also estimate the time needed for a particular task on the project. All this information is documented in an Excel file, which is classified as the Financial Calculation artifact from the base method. We used this information to gather information about the estimated time for each activity and consequently also for each discipline and the user role.

The comparison of the estimated and actual time is presented on the left hand side of Figures 6.1, 6.2, 6.3. It shows what portion of time was planned for a particular discipline/activity/user role and what portion was actually spent. With this comparison, project manager can identify project activities (and roles) that required more time than was expected. In our case, much more time was spent on the activities and user roles
related to the discipline Test than it was planned. The main reason was that they spent more time resolving bugs than they expected. On the other hand, they spent less time on the activities related to the discipline Implementation. If they had done more testing during the development, less bugs would have occurred.

6.1.3 Project Timeline

It is a rule in the observed company that at the beginning of each project they also prepare a project plan, which includes the timeline of a project – often presented with a gantt chart. From the gantt chart we gathered the information about the timespan of a particular task and when it was planned to be resolved. We used this information to visualize how the expected timeline deviated from the actual one. The information from the gantt chart is integrated into the actual timeline of a project and is presented with light blue rectangles in Figures 6.1 and 6.3. The first bigger deviation happened in the phase of negotiating wireframes. According to the plan, this was to be finished on 14th of March, however it took 13 days longer. Next, we see that a larger deviation occurred in the phase of development, which finished on 4th of August and not on 2nd of June as expected. There is also a deviation related to the activities from the disciplines Test, Deployment, and Project Management.

6.2 On-the-fly Analysis

The main idea of our approach is to consider each issue as an activity in the whole development process. This allows us to use our approach during the project performance to monitor, control and guide the development process. Furthermore, this also gives us the possibility to reconstruct the development method elements incrementally in real time, and also perform real time project performance analysis. All of the analyses are performed on a daily scale. One reason for this is that more detailed and accurate data about the time spent are often not available. When performing a reconstruction and analysis, we only include issues that are already resolved. So, as long as an issue is not resolved it is not included in the reconstruction. Since most of the issues are only open (active) for a few days, this does not present any real issue for the real time analysis.
Figure 6.3
Analysis on the level of activities.
6.2.1 Monitor, Control, and Guide

With the base method in place we have a possibility to monitor, control, and guide the development process. We observe “developers’ communication with the software repositories” and take actions when necessary. For example, each time a developer performs a commit, we check if the commit contains information about the issue id, and if this issue already exists in the issue tracking system. If not, we warn the developer that important information might be missing and ask her/him to provide the necessary details. Of course, maybe the developer did not provide information about the issue id on purpose because he/she only moved some files to another location, which is not related to an issue. What we can also do in real time is to classify all the files from a commit to artifacts and then check in the base method if these artifacts belong to only one activity. If not, we again warn the developer about possible bad practice or ask him to update the base method. At the same time we also check if the issue status is “in progress”, and if there are already any worklogs from this developer. When a developer resolves an issue, we immediately check to which activity we can relate this issue and if there is none that suits, we ask the developer for an explanation. As a result new knowledge is provided or we mark this as a bad practice. With all this we are able to document new ways of working and supplement the base method, provide help to developers and make sure that they do not deviate from the prescribed ways of working unintentionally, which is of particular importance for novice developers.

6.2.2 Real-time Process Reconstruction & Analysis

Constant monitoring of the work performed by developers allows us also to reconstruct the development process iteratively, in real time. To achieve this we observe the activities in software repositories. For example, every time a developer resolves an issue we recover process elements following the steps explained in Section 5.3. Since each issue presents a small step of the whole process, we can monitor and control that each activity (issue) is compliant with the prescribed method, and that the time (effort) spent for a particular activity and its time span are compliant with the project plan. This gives project managers a possibility to track the progress of the project in real time and take necessary actions to guide the project if it deviates from the course, or update the planning.
6.3 Chapter Summary

The reconstructed development method elements in combination with other information available from the software repositories (e.g., worklogs) provide a basis for different project analyses. These analyses can be done when the project is finished or during its performance.

Post-mortem analysis Once the project is finished we can use the reconstructed method in combination with the information about the worklogs to conduct a project retrospective. We can observe on a daily basis how much time (effort) was spent per discipline, activity, or user role. Furthermore, we also have a possibility to analyze the total time spent per discipline, activity, or user role and compare how this is compliant with the plan. All the deviations can be discussed and used as an input for further projects.

On the fly analysis Since the proposed approach supports the incremental reconstruction of method elements, we can perform all the analyses also while the project is being performed. Each time a new issue is resolved we can re-run the analyses and observe the current state of the project. With this the project manager has a better overview of the project performance and can detect any deviations from the plan as they happen. Furthermore, by observing “developers’ communication with the software repositories” we can monitor, control and guide their work. As soon as we detect any deviations from the base method we can warn a project member about the bad practice or obtain new knowledge to supplement the base method.

In this chapter we affirmatively answered to the RQ3 (Can we use the reconstructed information to gain information about the project performance?). The reconstructed information can be used in a combination with other information available in software repositories to perform different analyses about the project performance that can support and guide the work of project members.
To make the evaluation unbiased, we invited an additional software company to join the project. In this way, we did not know what tools this company uses to facilitate development, neither what information it stores in its software repository. When selecting the company and its project, we gave special attention to selecting the representative case. The company and its project were selected with special care so that it would be as typical as possible (in terms of the company size, project duration, budget, and team size). Here we followed the findings from our previous study available in [105] as well as findings presented in [106]. By its size the company falls under small and medium enterprises which are the most widespread type of software development enterprise in Slovenia. Regarding the project, it shares the characteristics with a typical project in Slovenia. Its duration was 6-12 months, the budget was under 100,000 €, and the team size was 17 (15 employees + 2 stakeholders).

The evaluation comprised the following steps:

**Step 1 - Analysis of the company’s software repository:** The aim of the first step was to find out whether our assumptions about a) typical attributes that could be found in a software repository and b) linkage between repository data (issues, commits, user accounts) hold for this particular company.

**Step 2 - Construction of the company’s base method draft:** In the second step, the goal was to create a draft of the company’s base method.

**Step 3 - Reconstruction of the development method elements:** The step three was dedicated to the evaluation as to what extent specific development method meta elements can be reconstructed from the company’s software.

**Step 4 - Analysis of the project performance:** The purpose of the fourth step was to evaluate how useful the reconstruction approach can be if used for controlling the project performance.

In the following, we report on the evaluation findings.

### 7.1 Profile of the Company

The company that we analyzed in the evaluation, develops e-business solutions for telco, insurance, and health industries and employs about 50 people. They develop software
by following a combination of agile and traditional approaches. The decision to involve this particular company into our research was based on the following reasons:

- The company was willing to provide all the necessary information about the project that seemed appropriate for the evaluation.
- The project team members were allowed and willing to commit required time for the purpose of the evaluation.
- The company already had a prescribed and documented method in place – i.e. guidelines for software development. This was useful as we could use it as a starting point for the construction of the company’s base method.

7.2 Analysis of the Software Repository Content

For the selected project, we imported the data from three different tools that the company was using during the project. These were Jira (used as issue tracking system), Subversion (used as revision control system) and LogicalDoc (used as document management system). In LogicalDoc a commit is perceived as a new version of a file (check-in). The first step was to retrieve issues, commits and users (user accounts). For a summary report on the data collected see Table 7.1.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td># of issues retrieved (ITS)</td>
<td>186-13(^1)</td>
</tr>
<tr>
<td># of commits retrieved (RCS + DMS)</td>
<td>379 + 166</td>
</tr>
<tr>
<td># of all files</td>
<td>3578</td>
</tr>
<tr>
<td># of users (employees + stakeholders)</td>
<td>15 + 2</td>
</tr>
</tbody>
</table>

\(^1\) In case of Jira, 13 issues were excluded as they were duplicates of other issues, could not be resolved, or were of the following type *meta task*.

Next, we tried to link commits from Subversion and LogicalDoc to Jira Issues. At first, we did that by extracting issue IDs from commit messages using regular expres-
sions, such as "\bJiraProjectKey\b\{[^w\d-]+\d+\}". A similar approach was also used by others [107, 108]. In this way, we were able to link roughly 70% of all commits with corresponding issues and about 60% of issues with corresponding commits. These results alone were already promising, as we linked the majority of the commits with issues and vice versa. To improve these results, we could have used the approach as suggested in [98], but we rather decided for a manual check via developers so that we also learned how consistent they are in using supporting tools. Together with their help, we were able to link additional 139 commits and 54 issues. 34 (6.2%) commits and 18 (10.4%) issues were left unlinked.

By analyzing unlinked commits, we found 9 of them were made to restructure and move the repository to a new location. The additional 12 were related to specific changes, such as upgrade library, add user as developer in pom.xml, fix typo, import files, etc. The last 13 unlinked commits that were left were all found to be connected with project management activities and the creation/modification of various related documents.

Similarly, by analyzing unlinked issues we found that they mainly presented system administration activities, such as increase RAM in test environment, update from java 6 to java 7, install SSL certificate, as well as activities that did not result in the creation of any artifact (e.g. setup development environment).

Regarding the resolution of users via user accounts, we had no problems, since the users were using the same usernames for all the systems. On this project, 17 different people participated, out of which 15 were employees and 2 stakeholders. For details on the results of linking the data see Table 7.2.

Table 7.2
The percentage of issues linked to commits and vice versa.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>% of commits linked to issues (regex)</td>
<td>68.3</td>
</tr>
<tr>
<td>% of issues linked to commits (regex)</td>
<td>59.5</td>
</tr>
<tr>
<td>% of commits linked to issues (regex + manually)</td>
<td>93.8</td>
</tr>
<tr>
<td>% of issues linked to commits (regex + manually)</td>
<td>89.6</td>
</tr>
</tbody>
</table>

At the end of this step, we also checked how well the development method meta model corresponds to the company and its expectations from the development method reconstruction. The company’s CIO was satisfied with the selected development method
meta elements.

7.3 Construction of the Base Method Draft

When we asked the company to tell us how they usually develop software (i.e. do they have any predefined steps, deliverables, techniques, user roles etc. that project team members need to follow) they gave us a documentation in which they defined basics of their development method. This included the description of project disciplines and corresponding activities. For each activity, the documentation also provided a description of the activity goals and associated artifacts. Each activity was further linked with user roles responsible for its performance. The described development method also differentiated among different types of projects. Based on the project size, these were divided into three groups: small, medium and large. All this information was written in a series of word files and available to all employees. We constructed the base method by analyzing the provided documentation. We did that together with one of the company’s project managers. The base method draft is depicted in Figure 5.3. In the next sections, we describe how this base method served us to reconstruct the development method elements that were used on the observed project.

7.4 Reconstruction of the Project Software Development Method Elements

The most important part of the evaluation was to check how well we can reconstruct development method elements of an observed project by analyzing the data from the corresponding software repository.

In order to find “golden rule”, i.e. to be able to measure how accurate the reconstruction is, we asked the person who acted as the manager of the observed project, to help us manually reconstruct the development method elements that were used on the project. For each issue, the manager identified connected commits and their files and based on that concluded what artifacts they represent. If an issue was found that had no connected commits (this happens when during the resolution of an issue no files are created), the project manager was asked to tell what activity this issue was about. Similarly, for the commits and related files that were not identified over issues, the project manager was
asked to classify them into the artifacts they represent. From activities, we then inferred disciplines and user roles.

In the next step, we used this information as the baseline against which we compared the results obtained with our algorithm. To measure the quality of the reconstruction, we used the precision and recall measures, which are known from information retrieval and pattern recognition. The results are shown in tables below. To fairly judge the quality of the algorithm, we also compared manually and automatically classified file versions into artifacts and issues into activities. The reason for this is that some artifacts are created gradually, through many versions, and are thus connected to many issues. Consequently, it would not be enough to limit the comparison on the artifacts only, as these might be reconstructed well only for some of the commits.

Table 7.3
Precision and recall of the automatically classified file versions to artifacts and issues to activities.

<table>
<thead>
<tr>
<th></th>
<th>Manually classified</th>
<th>Automatically classified</th>
<th>Precision</th>
<th>Recall</th>
<th>F-measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>File version</td>
<td>5945</td>
<td>5908</td>
<td>0.997</td>
<td>0.991</td>
<td>0.994</td>
</tr>
<tr>
<td>Issues</td>
<td>173</td>
<td>155</td>
<td>0.98</td>
<td>0.88</td>
<td>0.93</td>
</tr>
</tbody>
</table>

Table 7.4
Precision and recall of the automatically reconstructed development method elements compared to the manually retrieved development method elements.

<table>
<thead>
<tr>
<th>Method element</th>
<th>Manually classified</th>
<th>Automatically classified</th>
<th>Precision</th>
<th>Recall</th>
<th>F-measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Artifact</td>
<td>15</td>
<td>12</td>
<td>1.00</td>
<td>0.8</td>
<td>0.89</td>
</tr>
<tr>
<td>Activity</td>
<td>12</td>
<td>10</td>
<td>1.00</td>
<td>0.83</td>
<td>0.91</td>
</tr>
<tr>
<td>Discipline</td>
<td>5</td>
<td>5</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>User role</td>
<td>8</td>
<td>7</td>
<td>1.00</td>
<td>0.88</td>
<td>0.94</td>
</tr>
</tbody>
</table>

To achieve good results with our reconstruction algorithm, it is crucial that artifacts are reconstructed with as high precision and recall as possible, as the reconstruction of other method elements depends on this.

As you can see from the results, the classification of file versions to artifacts yielded
very good results in terms of precision (0.997) and recall (0.991) (Table 7.3, row 1). The reason for this is that artifacts are quite different in terms of their names, content and formats in which they are created. Thus we were able to accurately differentiate among them with a high confidence. The results also show that file versions did not much influence much the classification accuracy. This is an important finding as it supports iterative reconstruction of development method elements, i.e. step by step through the project performance.

The files that were misclassified or were left unclassified, were not many and in most cases due to one of the following reasons: (a) the file was of an unknown type, i.e. not defined in the base method – in our case these were mainly fonts of file type woff, eot, tff... (b) the file content could not be parsed – these were pdf files that contained images/scans and at the same time could not be classified based on keywords in their filenames and file paths and (c) files that included keywords which are more typical for some other artifact – in our case this happened for files that represented meeting minutes (e.g. on one particular meeting they were discussing a lot on the functional specification, so this word occurred many times in the meeting minutes, and thus the file was classified as an artifact “functional specification” rather than “meeting minutes”).

Good results in terms of precision (0.98) and recall (0.88) were obtained also for the classification of issues to activities (Table 7.3, row 2). In most cases, we were able to correctly identify activities that corresponded to issues by classifying files these issues created.

Table 7.4 shows results of reconstructing the development method elements. It compares manually retrieved development method elements with the automatic reconstruction. As you can see, all the automatic reconstructions were correct (100% precision) which is not surprising as the classification of files got such a high accuracy. The recall for activities and artifacts were however not that perfect at first sight (0.8, 0.83, respectively). Several activities and artifacts were missing in the automatic reconstruction. The explanation that we got from the project members revealed that the missing elements were all newly introduced and thus could not be found in the base method. Let us emphasize however that these results are based on the fully automatic reconstruction, i.e. without any involvement of the development team. In other words, the results, presented in the tables above, could be improved if the developers were asked for additional information in cases when classification or reconstruction could not be done.
7.5 Checking Project Performance

The algorithm for reconstructing development method elements can be used also during project performance. In this case, we reconstruct development method elements one by one, every time an issue is resolved, and check for their compliancy with the base method. There are several benefits of doing this during project performance. If the reconstructed development method elements are not compliant with the base method, the project manager is notified about that and can react by asking responsible project team members for clarification. In case no suitable argumentation is given, a bad practice was obviously detected and can also be prevented. However, if those responsible for the deviation can argue why they declined from what was expected, the base method can be supplemented by capturing new knowledge in terms of new development method artifacts or rules that bind project characteristics with some specific development method element. Additionally, if the project is being checked during its execution by reconstructing development method elements after each resolved issue, the information about activities and disciplines can be visualized on a timeline diagram, which gives an interesting insight into the current state of the project – this is only possible if company store information about the time planned and spent on the level of issues. It can even help to detect situations that might represent risk for the project. For example, if the majority of issues that are being resolved are still connected to activities and disciplines that should already be finished then it could be that we are at risk that the project will be late.

For the purpose of the evaluation, we simulated the project realization by passing the issues to our approach, as they were appearing chronologically (ordered ascending by the resolved date-time attribute) during the analyzed project. For each issue we reconstructed the development method elements. The Figure 7.1 shows the reconstructed development method. What is worth mentioning is that we improved the base method by three new development method elements (two activities and one user role), which were detected during the reconstruction and present new knowledge about development practice. This happened when the algorithm was not able to classify a file into any of existing artifacts or an issue into any of existing activities and we thus asked the project manager for explanation. He explained that these development method elements are important but we obviously failed to capture them when we were creating the base method draft. Final assessment given by the project manager was that he would like to have our approach and iSPRToolset implemented and available for future projects he will be working on.
7.6 Discussion

The approach presented in this thesis has some limitations that were in more details presented in Section 5.4. Firstly, it does not reconstruct all possible development method elements but only some of them, i.e. disciplines, activities, artifacts and user roles, to be
exact. Furthermore, it depends on the quality of the data captured in software repositories.

The approach is semi-automatic, since it requires some effort (although not substantial) from the project team members in order to work optimally. The involvement of project members is needed when we introduce the approach inside the company to construct a draft of a base method. The amount of time required for this depends on whether the company already has a documented method or if we can use any off-the-shelf methods. In the worst case scenario the base method has to be constructed by interviewing the project members. However, it is important to note that this is done only once per company. In the case of the company that participated in the evaluation it took two days. Once we have the base method in place, we only need to involve project members when there is a deviation from the base method. At that point the involvement is required in order to capture new knowledge, which is important in order to keep the base method up-to-date to reflect the actual ways of working on the projects. We believe that after a few projects have been completed, most of the development method elements will be captured and we expect that only marginal involvement of project members will be required on further projects. Furthermore, we expect that the level of involvement depends on the characteristics of a project such as size of a project, number of project members, complexity of a project, new technologies and approaches, etc. So the larger the project is, the more involvement on the side of project members can be expected. In the case of the project used in the evaluation two hours of the project manager’s time were required during the reconstruction.

Comparing the required effort on side of project members to reconstruct the development method with existing approaches is quite difficult since the existing approaches only reconstruct the method on the level of disciplines and are mainly interested in the workflow. However, all the approaches require some manual effort before the reconstruction in order to prepare the event log in case of software process mining and discovery or to prepare a training set that can be used for further classification in case of software process recovery. This manual effort could be compared with the effort that our approach requires in order to prepare the draft of the base method. Later on, our approach requires some additional effort in order to capture new knowledge in case of detected deviations. With this we keep the base method up-to-date to reflect the actual ways of working. This is, however, not supported by the existing approaches.

On the other hand, with our approach we make one step further towards automatic
reconstruction of software development methods from the data available in software repositories. Comparing to the existing approaches it supports reconstruction of development method to a higher level of details. Besides disciplines, which is the level the most of the existing approaches were focusing on, it also supports reconstruction of activities, user roles and artifacts. Furthermore, it helps the company in capturing and maintaining its base method. Next, it helps the company to conduct development projects in a way that they are performed consistently and in line with what the company prescribes within the base method. Moreover, it helps to detect deviations from the base method as project is performed. In the case of the project used in the evaluation we were able to detect the following deviations:

- System Administration (activity) was missing in the base method.
- Setup Development Environment (activity) was missing in the base method.
- System Admin (user role) was missing in the base method.
- Contract (artifact) should be created, but was not reconstructed from the data.
- Tender (artifact) should be created, but was not reconstructed from the data.
- Milestone Report (artifact) should be created, but was not reconstructed from the data.

Another limitation in our approach is that it does not support the reconstruction of a workflow. Based on our observations, this is not trivial since several activities might be present several times throughout the project with different emphases and might overlap with other activities. Even if the project is following a waterfall lifecycle, we might be able to reconstruct the workflow on the level of disciplines, but on the level of activities this still remains a challenge. The problems related to workflow reconstruction can clearly be seen on the analyses of project performance (see Chapter 6), which show how the relative emphasis of disciplines and activities is changing throughout the project. From these analyses one can conclude which activity started/finished before another or which activity required most effort at a specific point in time, but it is hard to define the exact workflow.

So far, at least to our knowledge, none of the existing approaches supports the reconstruction of a workflow on the level of activities. Current approaches mainly focus on
the reconstruction of the workflow on the level of an issue, a review process or disciplines.

To further validate the usability of the approach, semi-formal interviews were conducted with seven project managers of the participating companies. We asked them the following questions:

■ How do you perceive the suggested approach in terms of its complexity? Could it be introduced in your company? Do you think it would be accepted by your employees?

■ What do you expect the main benefits would be of using such approach in your company?

■ What would you suggest in order to make the approach more useful?

The feedback that we received was generally positive. They all agreed the approach is simple enough to be adopted in their companies. Since it does not require any substantial effort from developers or changes that developers would need to introduce in their everyday practice, the acceptance of the approach is also expected to be high.

As the main benefit, they emphasized the following possibilities offered by the approach: a) to do retrospective on finished projects, b) to observe project performance on the fly and identify steps that do not fit their regular practice (i.e. decline from their base method), c) to keep base method up-to-date, and d) to give more emphasis on methodological aspects of their development activities (as a side effect). Finally, as a suggestion for improvement they were all consistent that it would be very useful if we were able to reconstruct also workflow information, i.e. how exactly activities and their smaller counterparts (tasks) were performed during an observed project.

7.7 Threats to Validity

There are multiple threats to validity that this research faces. We will address them in the context of construct validity, internal validity, external validity and reliability.

With respect to construct validity, we had to address the fact that we rely upon data that are created and annotated by project members and are stored in software repositories. To improve construct validity we validated the data and results with project members and constructed the reconstruction approach based on the insights that we got from
Evaluation

From an internal validity point of view we do not face any threats, since our main goal was to show that using data from software repositories we can reconstruct a development process and method followed on the project. In our evaluation we analyzed an already completed project, hence the data should not be biased.

An external validity issue we face is that we evaluated our approach only on one case study, hence it is hard to justify how generalizable our results are. However, the approach to reconstruct method elements and perform different analyses is straightforward: if all the required data are available, it is reasonable to assume that reconstruction can also be done on other development projects. Among different companies, the main difference, when using the proposed approach, is in the base method, which is specific to the company and should be defined based on the company’s development practice. Another threat we face is that the data are not of such good quality as required. For example, commits and issues might not be linked to the extent required. In our case study we were able to link 93.8% of all commits with an issue. However, this might not be achievable on other projects since it is up to the development culture and rules inside a particular company.

In terms of reliability the accuracy of the annotated data can be a concern as it can produce biased results. In case of the reconstruction this would give spurious results, but reconstruction would still be successful. So this threat is more related to the accuracy of the reconstructed method. To mitigate this threat, the reconstructed method was validated with project members.

7.8 Chapter Summary

In this chapter we present the results of the evaluation. Our approach was evaluated on the data provided by an additional company that is following a combination of agile and traditional approaches and employs 50 people.

Analysis of software repository content As a first step we analyzed what kind of soft-
ware repositories they use. As expected they used revision control system (Subversion), document management system (LogicalDoc), and issue tracking system (Jira). Next we checked how efficiently can we link the data between software repositories. We were able to link 93.8% commits to issues and 89.6% issues to commits.

**Construction of the base method draft**  As a next step we have constructed the draft of the base method for what we have used the documentation available inside the company. We validated the base method with the project manager. The construction of the base method in our case took two days.

**Reconstruction of the method elements**  The most important part of the evaluation was to check how well we can reconstruct the method elements. The results were really good. We were able to automatically classify 88% of all issues to corresponding activity with F-measure 93%, and 99.1% of all file versions to corresponding artifacts with F-measure 99.4%.

**Checking project performance**  When evaluating the reconstruction approach we reconstructed the development method elements in an incremental manner, one by one. In this way we were able to detect potential deviations from the base method and obtain new knowledge. In the case of the project used in the evaluation we were able to detect the following deviations:

- System Administration (activity) was missing in the base method.
- Setup Development Environment (activity) was missing in the base method.
- System Admin (user role) was missing in the base method.
- Contract (artifact) should be created, but was not reconstructed from the data.
- Tender (artifact) should be created, but was not reconstructed from the data.
- Milestone Report (artifact) should be created, but was not reconstructed from the data.
Software development is a complex and creative task, whose sophisticated results are increasingly influencing our daily lives in various ways. Due to the nature of this work, it is important that each company has a method in place to manage, control, and guide the work of software developers and project managers. Otherwise, confusion may ensue, leading to project failures, low quality of the developed software and higher maintenance costs.

To manage software projects, companies often use different supporting tools, such as issue tracking systems, revision control systems, document management systems, code review tools and others. Their main goal is to support the work of developers. Each tool per se contains a lot of information and valuable knowledge on how a project has been performed in practice. However, to obtain an even better overview of the development process as a whole, the information from these tools can be linked. Linked data can then be used to reconstruct what really happens behind those projects and eventually to learn, among others, why some projects go well and others do not.

In this dissertation, we described how the data from different software repositories (issue tracking system, revision control system, document management system) can be used to reconstruct valuable information on the project performance with only a little involvement of the developers. The aim of our work was to demonstrate that using and linking the data from tools comprising software repositories, allows us to reconstruct the development method in more details than existing approaches do. Furthermore, the aim was to show that it is possible to capture the actual ways of working in a company, in a form of a base method, which can be constantly kept up-to-date without any significant involvement of developers.

To identify the information that can be retrieved from software repositories we cooperated with five companies, which shared their data with us. Based on the findings we developed an approach to reconstruct development practice. We evaluated the approach on a real software repository shared by an additional company. The results show that software repository information suffice for the reconstruction of various aspects of development process, i.e. disciplines, activities, roles, and artifacts.

In dissertation we have also shown how the reconstructed data together with other data available in software repositories can be used to perform different post-mortem or on-the-fly analysis. These analyses are important to monitor, control and guide work of developers, and they provide an overview of the current state of a project at any time. This is especially beneficial for project managers to take actions as soon as possible to
prevent project deviations (e.g. running late or spending more time that planned).

8.1 Principal Scientific Contributions

The principal scientific contribution of this dissertation is the novel approach to semi-automatically reconstruct development method that was followed on a specific project using the data that are stored in software repositories (issue tracking system, revision control system and document management system). The approach supports the reconstruction of the method in an incremental manner, issue by issue, which plays an important role in further project performance analysis. Because of the incremental reconstruction we can at each point in time observe the current status of a project and as soon as possible detect any deviations. Compared to the existing approaches that are mainly focused on the reconstruction of the process workflow on a high-level, level of disciplines or main activities, or they focus on the reconstruction of specific elements of the development process (e.g. lifecycle of an issue), we go one step further with our approach. We reconstruct development method to a higher level of details that include disciplines, activities, artifacts and user roles. We evaluated the approach on the data provided by a Slovenian company and the results are promising.

8.2 Future Work

As part of our future work we plan to gather data from other software repositories and include it into the process of reconstruction. With this we expect to rise the reconstruction accuracy and level of details reconstructed. We also plan to use the approach on other software projects to see how it performs in real-time manner (monitor, control, guide).

As part of the future work we also plan to investigate how we can use data stored in software repositories to reconstruct the workflow on the level of activities. This was pointed out by a representative of the participating companies as one of the possible improvements of the current approach.
iSPRToolset
As a part of the research described in this thesis, we developed a computerized tool called the intelligent software process reconstruction toolset (iSPRToolset) that facilitates the application of our approach in a company. The main reason why we decided to develop a new tool was to support several specific functionalities that were not available and could not be supported in the existing tools. For example, we had to support import and linking of data from several different software repositories. The existing tools, at that time, were limited to importing the data from only a specific set of software repositories, mainly for their own research purposes. Next, we wanted to support the construction and visualization of the company’s base method in a simple and friendly way so that it could be presented and discussed with the project members. Furthermore, we wanted to support manual as well as automatic classification of the data from software repositories to the method elements. This enabled us to have all the data in one place for further project performance analysis. Probably we could use, for each step separately, a different tool, but we wanted to present this to the company as something that is friendly and easy to use and what does not require too much effort. This was especially important in the phase of the evaluation to simplify the presentation and verification of the results, and to enable us to evaluate the usability of our approach in practice. The tool supports the following tasks:

**Import and link data from various software repositories**  The tool supports the import of the data from Git, Jira, Subversion, Github, Trello, LogicalDoc and DevTrack. Furthermore, it supports linkage between software repositories using regular expressions as presented in Section 7.2.

**User resolution**  Link the user accounts from different software repositories belonging to the same developer using the information available (username, email, first and last name, etc.).

**Define company’s base method**  For the end users (e.g. project managers, developers) it is important to have a friendly and easy way to define current base method that is prescribed inside the company.

**Reconstruct project’s method**  Using the information stored in software repositories the tool can reconstruct an actual method that was followed on a project. It performs the reconstruction in an incremental way – issue by issue. In case of a deviation
from the base method (e.g. no method element from the base method is compliant) or ambiguity (e.g. several method elements from a base method are compliant) it requires the involvement of project members. For more information about the reconstruction see Section 5.3.

**Process analysis** tool supports different analyses about how the development process has been performed and how project has been conducted. The number of different analyses depends on the data that are available.

In the next section are shown the examples of the tool graphical user interface on the open source project MongoDB. For more details check [http://ispr.jmlabs.eu](http://ispr.jmlabs.eu).

**Tool functionalities** Software repositories store a vast amount of data, so it is important to give users a friendly way to observe and browse through the data. The tool supports two options (1) the data can be visualized and browsed on a timeline, or (2) the data can be browsed through a table.

![Timeline](image)

**Figure A.1**
Data from software repositories visualized on a timeline.

The example of the timeline is shown in Figure A.1. On the timeline a user can easily
see the duration of each issue and how issues evolve through time. In this way it is also very easy to spot the issues that have a very long duration. Furthermore, if an issue has a red border this indicate that this issue has a link to the data from other software repositories. This allows to easily spot out the issues without artifacts. With a click on the element on the timeline, detailed information is shown. For example, if a user clicks on the issue it will get a detailed overview of all the data that are available. Data are already organized in a way to serve the important information such as (1) basic information about the issue (type, priority, status, description, etc.), (2) workflow of an issue, (3) related data from other software repositories, mainly revision control system, (4) reconstructed method elements and (5) presentation of the main events on a timeline. An example of a detail view is presented in Figure A.2.

In table view a user can observe the data from all software repositories at once or for each repository separately. Each row in a table presents one element from a software repository (e.g. in case of issue tracking system each row presents a different issue; in case of revision control system each row presents a different commit). User has an option to order or filter data by columns that are available. Furthermore, with a click on a row a user gets more detailed information about an issue or a commit – the same as in case of a timeline view. Examples of a table view are shown in Figure A.3 and A.4.
Figure A.2
Detail view of a data related to a selected issue (SERVER-15994).
Figure A.3

Table view of a data available in Jira. Each row presents a different issue.
An important step in the reconstruction is also the construction of a base method. The tool supports a simple addition and modification of method elements and links among them as shown in Figure A.5.

When method elements are reconstructed we can analyze for a specific issue what data are available and how this data were used to reconstruct the method elements. An example of an issue view (for the issue SERVER-17117) is presented in Figure A.6. This view gives us information about the issue, linked commits, artifacts that were produced in commits, and files that are related to this artifacts. Furthermore, there is information
Figure A.5
Tool supports construction of a base method in a friendly way.

about a user, an activity, a user role and a discipline.
Figure A.6
Detail view of a specific issue (SERVER-17117).

The tool also supports different analyses. The most interesting ones are those showing the time spent by user role, activity or discipline. Some of these analyses have already been introduced in the Section 6.1. In Figure A.7 there is presented the basic analysis of the data available in software repositories. We have information about the number of commits, number of issues, issue types, issue priorities, etc. In Figure A.8 is presented a more detailed analysis on the level of artifacts. Here we can select the artifacts that we are interested in and observe how the number of files related to the selected artifacts has been changing through time.
Figure A.7
Basic analysis of data available in software repositories.
Analysis on the level of artifacts. The chart presents the number of changed files that are linked to a specific artifact (in this case source code) through time.
BIBLIOGRAPHY


References


Delno avtomatizirana rekonstrukcija in dokumentiranje metod razvoja programske opreme

Marko Janković

doktorska disertacija
predana
Fakulteti za računalništvo in informatiko
kot del izpolnjevanja pogojev za pridobitev naziva doktor znanosti
s področja računalništva in informatike

rezirjeni povzetek

Ljubljana, 2020
Motivacija Razvoj programske opreme zahteva sistematičen in discipliniran pristop, ki zagotavlja kakovost procesa in njegovih rezultatov, tj. programske opreme [1]. Slednje je bilo prepoznano že v začetnem obdobju razvoja programske opreme in je pričelo do oblikovanja številnih metod razvoja programske opreme, ki omogočajo nadzorovan razvoj programske opreme. Z leti se je izkazalo, da ni idealne metode razvoja, ki bi ustrezala vsem vrstam projektov. Kako primerna je določena metoda razvoja za posebne organizacije, je odvisno od številnih dejavnikov in sicer vse od projektnih in organizacijskih lastnosti do značilnosti razvojne skupine. Te ugotovitve so bile potrjene s strani številnih raziskovalcev [2–8].

Eno izmed raziskovalnih področij, ki so se pojavila kot posledica omenjenih problemov, je inženirstvo metod (angl. method engineering). Raziskovalci na tem področju so vložili veliko truda v iskanje ustreznih rešitev. Ena izmed njih je inženirstvo situacijskih metod (angl. situational method engineering), ki je proces konstruiranja metod razvoja, ki so posebej prilagojene potrebam projekta [9]. Kljub temu empirične študije kažejo, da se način, kako razvoj poteka v praksi, bistveno razlikuje od tega, kar je predpisano v metodah [16, 18]. Slednje pomeni, da so v večini podjetj dejanski postopki razvoja programske opreme slabodokumentirani, če sploh [18]. To precej ovira učinkovitost razvoja programske opreme, saj se lahko podobne napake ponavljajo vedno znova, najboljše prakse niso zajete, ne obstaja dokumentacija iz katere bi se lahko učili novi zaposleni, ni mogoče izvesti analize uspešnosti projekta, itd.

Eden od možnih odgovorov na prej omenjene izzive je analiza podatkov, ki nastajajo med razvojem programske opreme in so shranjeni v programskih repozitorijih (angl. software repositories). Zavedamo se, da je preveč ambiciozno pričakovati, da bo vse znanje in izkušnje možno zajeti zgolj z analizo razpoložljivih programskih repozitorijev. Pričakujemo pa, da bi morale informacije, ki so na voljo v takšnih repozitorijih, zadostovati za dobro razumevanje, kako je bil določen razvojni proces izveden in kako poteka razvoj programske opreme v določenem podjetju.

Podjetja vidijo dodano vrednost v možnosti dokumentiranja in spremljanja dejanskega dela na razvojnih projektih in preverjanje skladnosti s predpisanimi metodami [16, 24]. Delati to ročno je zelo časovno potratno, zato je potreba po pristopu, ki bo zahteval minimalno vključenost in napor na strani razvijalcev.

Sorodna dela Številna raziskovalna področja že uporabljajo podatke iz programskih repozitorijev, da bi dobili vpogled v proces razvoja programske opreme. Spodaj so
Razširjeni povzetek

predstavljena štiri glavna področja, povezana z našim raziskovanjem.

Inženirstvo metod (angl. method engineering) se ukvarja z načrtovanjem, konstruiranjem in prilagajanjem metod, tehnik in orodij za razvoj informacijskih sistemov [28]. Glavni cilj je, da se metodo razvoja prilagodi glede na lastnosti in potrebe organizacije. Dober uvod v področje najdete v [29]. Ker se je skozi čas izkazalo, da se metode razvoja razlikujejo tudi med projekti znotraj ene organizacije, se je oblikovala smer inženirstvo situacijskih metod (angl. situational method engineering), ki se ukvarja z razvojem novih metod ali prilagajanjem obstoječih glede na specifične zahteve projekta. Odličen pregled preteklih raziskav lahko najdete v [5].

Rudarjenje programskih repozitorijev (angl. mining software repositories) se nanaša na preiskovanje in rudarjenje po podatkih, ki jih lahko pridobimo iz programskih repozitorijev. Pregled obstoječih pristopov in študij lahko najdete v [44] in [50]. Našemu raziskovanju je najbližje podpodročje rekonstrukcije procesa razvoja (angl. software process recovery), ki na podlagi podatkov iz programskih repozitorijev ter z uporabo tehnik rudarjenja, poskuša rekonstruirati proces razvoja programske opreme [61, 62].

Rudarjenje programskih procesov (angl. software process mining) je tesno povezano s področjem rudarjenja procesov [73]. Raziskovalci, ki se ukvarjajo z rudarjenjem programskih procesov uporabljajo tehnike rudarjenja procesov, da bi rekonstruirali model procesa in ugotovili neutičnostnosti, pomanjkljivosti in izboljšali obstoječe stanje. Pomemben korak, preden lahko uporabimo tehnike rudarjenja, je priprava dnevnika dogodkov. V primeru rudarjenja programskih procesov so dnevniki dogodkov pripravljeni iz podatkov, ki so shranjeni v programskih repozitorijih [74, 75].

Odkrivanje programskih procesov (angl. software process discovery) se ukvarja z odkrivanjem formalnega modela procesa razvoja na podlagi podatkov, ki so bili zbrani tekom izvajanja [82]. Številni raziskovalci so predlagali različne pristope, ki rekonstruirajo proces razvoja na podlagi podatkov iz programskih repozitorijev [84–88].

Večina obstoječih pristopov rekonstruira proces razvoja na visoki ravni, kjer jih zanimajo predvsem discipline ali ključne aktivnosti. Ne ukvarjajo se z rekonstrukcijo bolj podrobnih informacij kot so, kateri artefakti so bili ustvarjeni kot del aktivnosti, katere uporabniške vloge so bile odgovorne in kako je to skladno z metodo razvoja, ki je predpisana znotraj organizacije.

**Raziskovalni pristop**

V sklopu našega raziskovanja smo želeli odgovoriti na sledeča raziskovalna vprašanja:

1. Inženirstvo metod se ukvarja z načrtovanjem, konstruiranjem in prilagajanjem metod, tehnik in orodij za razvoj informacijskih sistemov. Glavni cilj je prilagoditi metodo razvoja glede na lastnosti in potrebe organizacije. Dober uvod v področje najdete v [29]. Ker se je skozi čas izkazalo, da se metode razvoja razlikujejo tudi med projekti znotraj ene organizacije, se je oblikovala smer inženirstvo situacijskih metod, ki se ukvarja z razvojem novih metod ali prilagajanjem obstoječih glede na specifične zahteve projekta. Odličen pregled preteklih raziskav lahko najdete v [5].

2. Rudarjenje programskih repozitorijev se nanaša na preiskovanje in rudarjenje po podatkih, ki jih lahko pridobimo iz programskih repozitorijev. Pregled obstoječih pristopov in študij lahko najdete v [44] in [50].

3. Rudarjenje programskih procesov je tesno povezano s področjem rudarjenja procesov [73]. Raziskovalci, ki se ukvarjajo z rudarjenjem programskih procesov uporabljajo tehnike rudarjenja procesov, da bi rekonstruirali model procesa in ugotovili neutičnostnosti, pomanjkljivosti in izboljšali obstoječe stanje. Pomemben korak, preden lahko uporabimo tehnike rudarjenja, je priprava dnevnika dogodkov. V primeru rudarjenja programskih procesov so dnevniki dogodkov pripravljeni iz podatkov, ki so shranjeni v programskih repozitorijih [74, 75].

4. Odkrivanje programskih procesov se ukvarja z odkrivanjem formalnega modela procesa razvoja na podlagi podatkov, ki so bili zbrani tekom izvajanja [82]. Številni raziskovalci so predlagali različne pristope, ki rekonstruirajo proces razvoja na podlagi podatkov iz programskih repozitorijev [84–88].
RQ1: Kakšna podporna orodja (programski repozitoriji) se običajno uporabljajo pri razvoju programske opreme, kakšne informacije lahko zajemajo in koliko informacij razvijalci dejansko vnesejo v programske repozitorije?

RQ2: Ali informacije, shranjene v programskih repozitorijih, zadoščajo za podrobnejšo rekonstrukcijo razvojne prakse kot obstoječi pristopi?

RQ3: Ali lahko uporabimo rekonstruirane podatke za vpogled v potek projekta?

Glavni iziziv pri odgovoru na raziskovalna vprašanja je bil pridobiti reprezentativne podatke. Veliko truda smo vložili v sodelovanje z lokalnimi (slovenskimi) podjetji, da bi dobili dostop do njihovih programskih repozitorijev in vpogleda v njihovo razvojo prakso. To je bilo še posebej zahtevno, saj podjetja to običajno obravnavajo kot zaupne informacije in teh podatkov niso pripravljena deliti. V sklopu našega raziskovanja smo sodelovali s šestimi različnimi podjetji.

Naš raziskovalni pristop je bil sestavljen iz naslednjih korakov: (1) analiza vsebine programskih repozitorijev, katere namen je bil ugotoviti katera podporna orodja sodelujejo podjetja in odprtokodni projekti uporabljajo pri razvoju programske opreme, (2) razvoj algoritmov (in orodja) za rekonstrukcijo elementov metode razvoja, ki nam bodo omogočili rekonstrukcijo elementov razvojne metode iz podatkov, ki so na voljo v programskih repozitorijih, (3) evalvacija pristopa na podatkih novega podjetja, ki ni bilo vključeno v prejšnje korake. Najprej smo preverili ugotovitve o generičnih atributih in nato evaluirali še pristop za rekonstrukcijo metode razvoja. Rezultate in ugotovitve smo prediskutirali skupaj z vodjo projekta.

**Analiza programskih repozitorijev** V prvem koraku smo identificirali atributne, ki so običajno shranjeni v različnih programskih repozitorijih, ter analizirali razmerja med ključnimi konstrukti programskih repozitorijev, kot so potrditev (angl. commit), zahtevek (angl. issue) in uporabnik. Pri slednjem nas je predvsem zanimalo ali lahko na osnovi teh podatkov sklepamo o elementih metode razvoja (npr. aktivnosti, artefakti, itd.). V idealnem primeru pričakujemo, da so povezave med potrditvami, zahtevki in uporabniki ena na ena. To pomeni, da vsaka potrditev v sistemu za nadzor verzij povezana z natanko enim zahtevkom v sistemu za sledenje zahtevkem, oba pa sta povezana z natanko enim uporabnikom.

Za podatke, ki smo jih potrebovali za potrebe analize smo prosili sodelujoča podjetja, ki so nam omogočila dostop do njihovih razvojnih okoliš in programskih repozitorijev.
Kot smo pričakovali, so si bila podjetja precej podobna glede na vrsto orodji, ki so jih uporabljali za podporo razvojnega procesa. Najpogostejša orodja so bila:

- Jira, Bugzilla ali DevTrack za sledenje zahtevkov/napak
- Subversion ali Git za nadzor verzij
- Sharepoint ali LogicalDOC kot sistem za upravljanje dokumentov

Nekatera podjetja so uporabljala tudi druga podporna orodja, ki so omogočala upravljanje pregledov kode (npr. Crucible, Reitveld) ali upravljanje skupnega sodelovanja (npr. Slack, Confluence, Skype). Ker ta orodja niso bila prisotna v vseh podjetjih in ker za nekatera orodja podatkov ni mogoče pridobiti zaradi politike zasebnosti, jih nismo dodatno analizirali. Da bi bile naše ugotovitve čimbolj splošne, smo v analizo vključili tudi podatke iz treh odprtokodnih projektov: MongoDB, Spring Framework in Hibernate ORM. Vsi trije projekti uporabljajo Jira kot sistem za sledenje zahtevkom in Git kot sistem za nadzor verzij.

**Podatki v programskih repozitorijih**  
Sistem za nadzor verzij hrani podatke o verzijah izvorne kode projektov. Vsaka verzija vsebuje informacije o spremenjenih (dodanih, posodobljenih, izbrisanih) datotekah, podatek o osebi, ki je naredila spremembo, zakaj jo je naredila in kdaj. Te informacije je mogoče pridobiti iz vseh razširjenih sistemov za nadzor verzij. Primerjava med Subversion in Git je prikazana v tabeli 4.1.

Sistemi za sledenje zahtevkom/napakam hranijo informacije o zahtevkih, ki so bili izvedeni na projektu. Za vsak zahtevek lahko pridobimo podrobne informacije kot so: kdo je ustvaril zahtevek in kdaj, vrsta in prioritet zahtevka, opis, naslov, oznake, priloge, povezava z drugimi zahtevki, itd. Vsak zahtevek je dodeljen razvijalcu, ki je odgovoren za njegovo reševanje. Primerjava glavnih atributov, ki jih je mogoče izveleči iz sistemov za sledenje zahtevkom (Jira, Bugzilla), je predstavljena v tabeli 4.2.

Sistemi za upravljanje dokumentov hranijo verzije dokumentov. V našem primeru sistem za upravljanje z dokumenti obravnavamo kot sistem za nadzor verzij. V bistvu imata enak namen razen, da so sistemi za upravljanje dokumentov (npr. SharePoint) specializirani za hranjenje dokumentov, medtem ko so sistemi za nadzor verzij (npr. Git) bolj specializirani za hranjenje izvorne kode. Nekatera podjetja uporabljajo sistem za nadzor verzij za shranjevanje izvorne kode in dokumentov. Vsaka verzija vsebuje informacije o izvedenih spremembah (dodanih, posodobljenih, izbrisanih dokumentih), skupaj s tem,
kdo je naredil spremembo (uporabnik), zakaj jo je naredili (sporočilo) in kdaj je bila narejena (časovni žig).

Za vsako podjetje in orodje smo preverili, kakšne podatke dejansko hranijo v svojih podatkovnih bazah ali dnevnikih. Na sliki 4.1 je naveden nabor atributov, ki smo jih našli v repozitorijskih programske opreme vseh petih sodelujočih podjetij, kot tudi v repozitorijskih programske opreme vseh treh odprtokodnih projektov.

**Povezave med programskimi repozitoriji** V naslednjem koraku smo preučili, kako dobro je mogoče povezati podatke shranjene v programskih repozitorijih. Slednje je pomembno, saj programski repozitorij ni samostojna fizična baza podatkov, temveč predstavlja logični pogled na več podatkovnih baz in dnevnikov različnih sistemov. V sklopu našega raziskovanja nas je zanimala predvsem povezava med potrditvami in zahtevki, kot je to z rdečo obrobo prikazano na sliki 4.2.

Naslednji izziv je povezovanje uporabniških računov, ustvarjenih v različnih sistemih, ki se nanašajo na istega uporabnika. Slika 4.2 (modra obroba) prikazuje primer, ko je bila ista oseba dodeljena zahtevek v Jiri in je izvedla potrditev v Githubu. Kljub temu, da gre za isto osebo je za prijavo v posamezno orodje uporabljen drug uporabniški račun. Slednje je pogosto, saj programski repozitoriji običajno vsebujejo orodja različnih ponudnikov (slika 4.3).

**Analiza realnih podatkov programskih repozitorijev** Pri analizi realnih programskih repozitorijev smo uporabili podatke iz treh odprtokodnih projektov in dveh projektov sodelujočih podjetij.

Analizirali smo, koliko potrditev je mogoče povezati z zahtevki, tako da iz sporočil potrditev pridobimo informacijo o ključu zahtevka. Rezultati so prikazani na sliki 4.4. Zanimiva ugotovitev je, da se število potrditev, ki jih je mogoče povezati z ustreznimi zahtevki, povečuje skupaj s trajanjem projektov. To velja tako za odprtokodne projekte kot tudi za komercialne projekte.

V drugem koraku smo analizirali koliko potrditev povezanih z zahtevkom je povezanih z natanko enim zahtevkom. Slednje je pomembno, saj želimo vsako potrditev povezati z natanko enim zahtevkom in tako zagotoviti nančrnjejšo rekonstrukcijo. Rezultati so prikazani na sliki 4.5. Kot lahko vidimo, so bila naša pričakovanja potrjena. Več kot 95% (v povprečju) potrditev, ki smo jih lahko povezali z zahtevkom, je bilo povezanih z natanko eno zahtevkom.
Razširjeni povzetek

Nazadnje smo preverili koliko zahtevkov, ki so bili končani in smo jih lahko povezali s potrditvami, je rešil natanko en razvijalec. Za vsak zahtevek smo preverili vse povezane potrditve in preverili, če je vse potrditve naredil isti uporabnik. Rezultati so predstavljeni na sliki 4.7. Kot lahko vidimo, je v povprečju velik odstotek zahtevkov, ki jih je rešil en razvijalec. Tudi to velja tako za odprtokodne kot tudi za komercialne projekte.

Rekonstrukcija metode razvoja programske opreme

Glavni cilj našega raziskovanja je bil omogočiti rekonstrukcijo metode razvoja, ki ji razvijalci sledijo na izbranem projektu, z minimalnim vključevanjem razvijalcev.


Metamodel metode razvoja

Obstajaveč različnih metamodelov, ki podpirajo obstoječe razvojne metode [100]. Za namen našega raziskovanja smo sledili metamodelu Software & Systems Process Engineering Metamodel (SPEM) [101]. Njegove osnove, vključno z ločevanjem metod in procesnih konceptov, so bile na kratko opisane sodelujočim podjetjem. V sodelovanju s podjetji, ki so pripravljali podatke v programskih repozitorijih, smo oblikovali ožji nabor meta elementov (discipline, aktivnosti, artefakti, uporabniške vloge), ki bi jih radi rekonstruirali. Končni meta model s povezavami med meta elementi je prikazan na sliki 5.2.

Konstruiranje osnovne metode razvoja

Konstruiranje osnovne metode je predhodni korak, ki je potreben za rekonstrukcijo specifične metode projekta. Ker so osnovne metode specifične za posamezno organizacijo, je to potrebno narediti za vsako organizacijo posebej. Osnovne metode se lahko oblikujejo na različne načine. Eden od načinov je oblikovanje metode na osnovi informacije, ki jih pridobimo od organizacije. Alternativno lahko začnemo z osnovno metodo, ki ustreza komercialni metodi razvoja programske opreme, kot je na primer RUP [67] ali podobno. Osnovna metoda se nato posodobi vsakič, ko se rekonstruirja nova specifična metoda projekta in sčasoma postane dobra
predstavitev postopkov razvoja programske opreme znotraj organizacije.

Slika 5.3 predstavlja primer osnovne metode, izdelane na podlagi intervjujev z vodjami projektov podjetja, ki je sodelovalo pri našem raziskovanju.

Rekonstrukcija elementov metode razvoja

Rekonstrukcija elementov metode se lahko izvaja postopoma, saj se zahtevki obdelujejo neodvisno drug od drugega.

V okviru rekonstrukcije obravnavamo le zahtevke, ki so relevantni za sam potek razvoja. Za rekonstrukcijo specifične metode projekta se izvedejo naslednji trije koraki: (1) zberemo vse datoteke iz potrditev, ki so povezane z zahtevkom in jih razvrstimo v artefakte, (2) na podlagi artefaktov definiramo aktivnost, in (3) na podlagi aktivnosti identificiramo ustrezne uporabniške vloge in discipline. Spodaj je podrobneje predstavljen vsak korak.

Artefakti se rekonstruirajo iz datotek, ki so bile spremenjene v repozitoriju in so povezane z opravljenim zahtevkom. Pri razvrščanju datotek v artefakte uporabimo podatke o nazivu datoteke, končnici in vsebini. Tehnike, ki jih lahko uporabimo za razvrščanje datotek v artefakte je veliko. Algoritmi strojnega učenja so koristni kadar imamo podatke, ki jih lahko uporabimo za učenje. V primeru, da ti podatki niso na voljo, lahko uporabimo algoritem, ki je opisan v razdelku 5.3.4. Shematski prikaz rekonstrukcije artefaktov je predstavljen na sliki 5.4.

Ko smo rekonstruirali artefakte, ki so nastali kot rezultat reševanja zahtevka, rekonstruiramo aktivnosti. To naredimo z iskanjem ustreznih povezav med artefakti in aktivnostmi v osnovni metodi. Primer je predstavljen na sliki 5.5 (elementi z rdečo obrobo). V večini primerov so aktivnosti povezane samo z enim artefaktom, zato rekonstrukcija aktivnosti ni problem. Vključevanje razvijalcev je potrebno le v redkih primerih: a) kadar te povezave niso ena na ena (več aktivnosti je lahko odgovornih za ustvarjanje določenega artefakta), b) če nimamo artefakta, ki bi ga lahko uporabili za sklepanje o aktivnosti. To se zgodi, ko obravnavamo zahtevek v okviru katerega ni bila ustvarjena nobena datoteka. V tem primeru moramo vključiti razvijalca, ki izbere ustrezno aktivnost in označiti zahtevek kot slabo prakso, ali pa ustvari/posodobi aktivnost v osnovni metodi. V primeru, da se razvijalec odloči, da bo ustvaril novo aktivnost, mora zagotoviti informacije o uporabniški vlogi in disciplini.

Podobno rekonstruiramo tudi uporabniške vloge in discipline. Ker žele poznamo aktivnost, preprosto preverimo v osnovni metodi, da pridobimo tudi povezane uporabniške vloge in discipline. Poleg tega za vsako uporabniško vlogo pridobimo tudi uporabniške
Analiza procesa razvoja programske opreme

Rekonstruirani elementi metode razvoja v kombinaciji z drugimi informacijami, ki so na voljo v programskih repozitorijih, zagotavljajo osnovo za različne analize. Analizo lahko izvajamo po koncu projekta ali pa tekom izvajanja projekta. Slednje nam omogoča, da sprotno spremljamo razvojni proces in ukrepmo, takoj ko potek projekta odstopa od načrta. Vsi rezultati v tem razdelku so bili ustvarjeni z uporabo podatkov, ki nam jih je zagotovilo sodelujoče podjetje. Isti podatki so bili uporabljeni tudi v fazi evaluacije.

Analiza po koncu projekta

Predlagani pristop za rekonstrukcijo metode razvoja lahko uporabimo nad podatki že končanega projekta, kar nam omogoči zanimive analize o uspešnosti projekta. Z uporabo podatkov o porabljenem času (angl. worklogs), ki jih vnašajo uporabniki za posamezne zahteve, lahko opazujemo koliko časa (truda) je bilo tekom projekta vloženega za posamezno disciplino, aktivnost ali uporabniško vlogo. Iz slednjega lahko razberemo, kateri del projekta je zahteval največ časa.

Kadar imamo na voljo tudi podatke o projektmem planu, kjer so običajno ocenjeni predvideni časi potrebni za izvedbo posameznih aktivnosti, lahko to informacijo uporabimo za primerjavo planiranega in dejansko porabljenega časa. Slednje nam omogoča vpogled v to, katere aktivnosti so zahtevale več časa, kot smo predvidevali. V nekaterih podjetijih projektni plan vsebuje tudi podatek o planiranem časovnem okvirju projekta - pogosto predstavljen z Gantovim diagramom. Slednje nam omogoča, da v analizi vključimo predvidena časovna obdobja za planirane aktivnosti in posledično opazujemo, katere aktivnosti so najbolj zamujale ter prispevale k daljšemu trajanju projekta.

Rezultati analiz so predstavljeni na slikah 6.1 (na nivoju disciplin), 6.3 (na nivoju aktivnosti), 6.2 (na nivoju uporabniških vlog).

Analiza tekom projekta

Glavni cilj analize uspešnosti projekta tekom izvajanja je, da omogočimo spremljanje, nadzor in usmerjanje razvojnega procesa. Opazujemo komunikacijo razvijalcev s programskimi repozitorijami in po potrebi ukreparamo. Na primer, vsakič, ko razvijalec zaključi posamezen zahtevek, se izvede rekonstrukcija elementov metode razvoja, nato pa se preveri, ali je slednje skladno s predpisano metodo razvoja. V primeru odstopanja se razvijalca opozori.

**Evalvacija** Da bi bila evalvacija nepristranska, smo k sodelovanju povabili novo podjetje (profil podjetja je predstavljen v razdelku 7.1). Posledično nismo vedeli katera orodja podjetje uporablja za razvoj in katere informacije hranijo v svojih programskih repozitorijih. Evalvacija je vključevala naslednje korake: analizo programskih repozitorijev podjetja, konstrukcijo osnovne metode podjetja, rekonstrukcijo elementov metode razvoja in analizo uspešnosti projekta. V nadaljevanju je vsak korak predstavljen bolj podrobno.


Pri konstrukciji osnovne metode podjetja smo izhajali iz njihove dokumentacije, kjer imajo popisane metode razvoja. To je vključevalo opis projektnih disciplin in pripadajoče aktivnosti. Za vsako aktivnost je bil v dokumentaciji podan tudi opis ciljev aktivnosti in seznam povezanih artefaktov. Vsaka aktivnost je bila povezana z uporabniki vlogami, ki so odgovorne za njeno izvedbo. Osnovno metodo smo, skupaj z vodjo projekta, izdelali z analizo priložene dokumentacije. Osnutek osnovne metode je prikazan na slici 5.3.

Najpomembnejši del evalvacije je bila ocena natančnosti rekonstrukcije elementov metode razvoja. Da bi lahko ocenili, kako natančna je rekonstrukcija, smo prošli osebo, ki je delovala kot vodja opazovanega projekta, da nam pomaga ročno rekonstruirati elemente metode razvoja, ki so bili prisotni na projektu. V naslednjem koraku smo ročno označili podatke uporabili kot izhodišče s katerim smo primerjali rezultate dobijene z našim algoritmom. Za merjenje kakovosti rekonstrukcije smo izračunali natančnost (angl. precision) in priključ (angl. recall) razvrstitve ter povprečno mero priključa in natančnosti F. Rezultati so prikazani v tabelah 7.3 in 7.4. Za doseganje dobrih rezultatov pri našem rekonstrukcijskem algoritmu je ključnega pomena, da se artefakti rekonstruirajo z čim
večjo natančnostjo in priklicem, saj je od tega odvisna rekonstrukcija drugih elementov metode. Kot lahko vidimo iz rezultatov, je razvrstitev različic datotek v artefakte prinesla zelo dobre rezultate (tabela 7.3, vrstica 1). Razlog za to je, da so artefakti precej različni glede na njihova imena, vsebine in formate, v katerih so ustvarjeni. Dobri rezultati, v smislu natančnosti in priklica, so bili dobljeni tudi za razvrstitev zahtevkov v aktivnosti (tabela 7.3, vrstica 2). V večini primerov smo lahko z razvrščanjem datotek, ki so bile ustvarjene v okviru zahtevka, pravilno identificirali aktivnost.

Za namen evaluacije analize uspešnosti projekta smo simulirali realizacijo projekta tako, da smo zahtevke, urejene kronološko glede na čas razrešitve, postopno vključevali v naš pristop. Za vsak zahtevek smo rekonstruirali elemente metode razvoja. Slika 7.1 prikazuje rekonstruirano metodo razvoja. Osnovno metodo smo izboljšali s tremi novimi elementi (dva artefakta in eno uporabniško vlogo), ki smo jih odkrili tekom rekonstrukcije. To se je zgodilo, ko algoritmem ni mogel razvrstiti datoteke v nobenega od obstoječih artefaktov ali zahtevka v katero od obstoječih aktivnosti, zato smo vodjo projekta prošli za pojasnilo. Pojasnil je, da so ti elementi metode razvoja pomembni, vendar jih očitno nismo zajeli, ko smo konstruirali osnovno metodo. Končna ocena vodje projekta je bila, da pristop prinaša uporabno vrednost pri spremljanju in nadzoru razvoja programske opreme in bi ga želel uporabiti tudi na prihodnjih projektih.

**Zaključek** V pričujoči disertaciji smo predstavili, kako se lahko podatki iz različnih programskih repozitorijev uporabijo za rekonstrukcijo dragocenih informacij o poteku in uspešnosti projekta, z minimalno vključenostjo razvijalcev. Namen našega dela je pokazati, da lahko z uporabo in povezovanjem podatkov iz programskih repozitorijev omogočimo rekonstrukcijo metode razvoja bolj podrobno, kot obstoječi pristopi.