Grip on software: understanding development progress of SCRUM sprints and backlogs

Helwerda, L.S.

Citation
Helwerda, L. S. (2024, September 13). *Grip on software: understanding development progress of SCRUM sprints and backlogs*. *SIKS Dissertation Series*. Retrieved from https://hdl.handle.net/1887/4092508

Version:	Publisher's Version
License:	Licence agreement concerning inclusion of doctoral thesis in the Institutional Repository of the University of Leiden
Downloaded from:	https://hdl.handle.net/1887/4092508

Note: To cite this publication please use the final published version (if applicable).

Grip on Software: Understanding development progress of SCRUM sprints and backlogs

Proefschrift

ter verkrijging van
de graad van doctor aan de Universiteit Leiden,
op gezag van rector magnificus prof.dr.ir. H. Bijl,
volgens besluit van het college voor promoties
te verdedigen op vrijdag 13 september 2024
klokke 11:30 uur
door

Leon Sebastiaan Helwerda

geboren te Voorburg, Nederland
in 1992

Promotores:
Prof.dr.ir. F.J. Verbeek
Dr. W.A. Kosters

Promotiecommissie:
Prof.dr. M.M. Bonsangue
Prof.dr. H.C.M. Kleijn
Prof.dr. S. Manegold
Prof.dr. M.R.V. Chaudron (Eindhoven University of Technology)
Dr. C. Soomlek (Khon Kaen University)
Dr. F. Niessink (Stichting ICTU)

Copyright © 2024 Leon Helwerda
Cover art by Marian Helwerda, based on photograph by Leon Helwerda
Printed by: NBD Biblion

Funded by Stichting ICTU as part of a collaboration between Leiden University and ICTU

SIKS Dissertation Series No. 2024-28
The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

Contents

1 Introduction **1**
- 1.1 Preface . . . 3
- 1.2 Context . . . 3
  - 1.2.1 Software development, Agile and SCRUM . . . 4
  - 1.2.2 Machine learning, pattern recognition and predictive analytics . . . 7
  - 1.2.3 Case studies of workflows . . . 9
- 1.3 Design scope . . . 11
- 1.4 Problem statement . . . 14
- 1.5 Pipeline components . . . 17
  - 1.5.1 Instances . . . 18
  - 1.5.2 Non-functional requirements . . . 19
- 1.6 Structure of this thesis . . . 20

2 Data pipeline **23**
- 2.1 Introduction . . . 25
  - 2.1.1 Ecosystem . . . 26
  - 2.1.2 Structure . . . 27
- 2.2 Design . . . 27
  - 2.2.1 Distributed data systems . . . 27
  - 2.2.2 Agent-based communication . . . 28
  - 2.2.3 Organizational approaches . . . 28
- 2.3 Method . . . 29
  - 2.3.1 Data acquisition . . . 30
  - 2.3.2 Further pipeline steps . . . 33
- 2.4 Technical considerations . . . 36
  - 2.4.1 Generalizability . . . 36
  - 2.4.2 Continuous integration . . . 37
  - 2.4.3 Documentation . . . 37
  - 2.4.4 Novelty . . . 38
- 2.5 Results . . . 39
- 2.6 Discussion . . . 40

3 Database construction **41**
- 3.1 Introduction . . . 43
- 3.2 Relevant work . . . 45

3.3 Method . . . 45
3.3.1 Data model . . . 47
3.3.2 Linking data sources . . . 57
3.4 Architecture . . . 59
3.5 Experiments . . . 63
3.5.1 Setup . . . 63
3.5.2 Results . . . 64
3.6 Discussion . . . 66

4 Pattern recognition methods **69**
4.1 Proposition . . . 71
4.2 Background . . . 72
4.2.1 Framework . . . 73
4.2.2 Story points and adaptations . . . 74
4.3 Related work . . . 75
4.4 Approach . . . 76
4.4.1 Feature extraction . . . 77
4.4.2 Data set . . . 79
4.4.3 Models . . . 80
4.5 Analysis strategy . . . 83
4.6 Results . . . 84
4.6.1 Sprint classification and estimation . . . 84
4.6.2 Backlog size estimation . . . 85
4.7 Conclusions . . . 89
4.7.1 Threats to validity . . . 90
4.7.2 Proposed additions . . . 91

5 Information visualization **93**
5.1 Preamble . . . 95
5.2 Purpose . . . 96
5.3 Relevant concepts . . . 97
5.4 Dashboard framework . . . 99
5.5 Visualizations for analytical decision support . . . 102
5.5.1 Sprint report . . . 102
5.5.2 Prediction results . . . 111
5.5.3 Timeline . . . 114
5.5.4 Leaderboard . . . 118
5.6 Visualizations for ecosystem management . . . 121
5.6.1 Collaboration graph . . . 121
5.6.2 Process flow . . . 125
5.6.3 Heat map . . . 128
5.6.4 Platform status . . . 131
5.7 Novel backlog visualizations . . . 134
5.7.1 Product backlog burndown chart . . . 134
5.7.2 Product backlog progression chart . . . 135
5.7.3 Product backlog relationship chart . . . 137

5.8 Evaluation . . . 138
5.8.1 Assessment . . . 138
5.8.2 Adoption . . . 141
5.8.3 Conclusion . . . 142

6 Discussion **145**
6.1 Retrospective . . . 147
6.1.1 Technical overview . . . 147
6.1.2 Main contributions . . . 149
6.2 Overall conclusion . . . 151
6.2.1 Problem statement . . . 151
6.2.2 Research questions . . . 152
6.3 Future work . . . 159
6.3.1 Further research . . . 159
6.3.2 Generalizability . . . 161

Glossary **163**

Bibliography **169**

Appendices **183**

A Code repositories of the Grip on Software pipeline **185**

B Queries used in database performance experiments **189**

Summary **203**

Samenvatting **207**

Curriculum Vitae **211**

Acknowledgments **213**

SIKS Dissertation Series **215**

Chapter 1

Introduction

The Grip on Software research and its relation to the SCRUM *software development method*

Abstract of Chapter 1

In this chapter, we present the Grip on Software research project as well as the contextual situation of the project within specific fields of study: software engineering, quality control and development methodologies. We describe methods and terminology that exist in this field, in particular for the SCRUM framework. Furthermore, we provide an overview of topics in machine learning and pattern recognition that are applicable to our study. We then introduce the problem statement and hypotheses have resulted in our research questions.

The following problem statement is central to our research: "How can extraction and analysis of measurable events during a SCRUM software development process as well as other qualities of the product and team be used to significantly improve the predictability of practices employed at software development organizations?"

We discuss the social aspects and importance of this research for our stakeholders, including developers, project owners and quality control engineers. This surmounts to a motivation on the actual utility of the results. We finally explain the toolset that we use for our research as well as the scientific factors within it: reusable code components, pipeline architecture, database modeling, machine learning, estimation models and information visualization.

1.1 Preface

Software development is a complex process. It is important that software products become stable and maintainable assets. To that end, methodologies have been designed to support development. We examine and extend one of these software development frameworks—SCRUM—using machine learning methods in the hope to better understand what happens during the process, to learn from these events, to avoid problematic situations and to improve the process as a whole.

A combination of software development and machine learning might evidently seem like a harmonious solution, given that lots of data is available for such algorithms. Similarly, we may easily receive feedback from a large group of involved people. However, it is still important to properly introduce novel concepts and techniques to an existing working environment.

This thesis presents the concepts, goals, challenges, design, methods, implementation, results and observations of the Grip on Software research study, which delves into the discovery of unused or underused data sources in a software development environment that are valuable when it comes machine learning. After acquisition, the data needs to be modeled and represented in a structured manner, such that extraction of representative attributes is possible. This paves the way to machine learning and information visualization as our research output. Consequently, interested parties are able to look into the results, find problems in the software development process specific to their team and adjust ahead of future challenges.

We consider problems regarding workload scheduling, particularly in modern development frameworks; these are often faced with fast-paced changes. Given that end users become more involved in the process of requesting and tuning desired functionality in the software, development platforms—including quality control and other tracking systems—need to function optimally in order to support a team in delivering the work that they commit themselves to. Typical planning applications only provide an overview of the current situation and limited reporting options. These prevalent deficiencies hinder key figures with prominent roles in the process from understanding what has happened recently. We augment existing reports with more depth in time and breadth of qualitative characteristics.

In this introduction, we first focus on the contextual description in Section 1.2, with initial literature review and specifics on the SCRUM software development method, in particular on how it is applied at two governmental organizations—our case studies—ICTU and Wigo4it. This leads to a description of goals and design scope in Section 1.3. Subsequently, we introduce the research questions of Grip on Software in Section 1.4. Next, our discussion concentrates on the implementation aspect, with an introduction of the component-based design of the data acquisition pipeline in Section 1.5. The remaining chapters of the thesis are finally outlined in Section 1.6.

1.2 Context

In order to properly understand the intricacies of an environment in which a research study takes place, it is relevant to review existing literature and lay down the groundwork upon which the analysis advances toward novel observations and results. For the Grip on Software research project, it is a prerequisite to review the prior work regarding software development methods, in particular SCRUM, which we introduce in Section 1.2.1. We look into the origins of this methodology and how it is formulated. We also focus on what typically happens when SCRUM is brought into practice. The essential terminology defined by the framework is also part of this elaboration.

We also consider the concepts from machine learning that are relevant for our eventual analytical goals in Section 1.2.2. State-of-the-art and established algorithms are discussed, while exploring the relationship between various terms and fields. We include many approaches that have a statistical foundation. We consider the main challenges and objectives of these types of algorithms. This enables us to apply suitable models in our subsequent study.

This contextual description is intended to introduce the main definitions and semantics regarding the ecosystems at software development organizations encountered during our research, which we further illustrate through case study descriptions in Section 1.2.3. Potential practical approaches and systematical formalizations are briefly discussed here, specifically when it comes to software development and machine learning. These are applied throughout our research, but in particular concerning Chapter 4. There, we provide a recap of the SCRUM framework and expand upon our literature study, after which we introduce our contributions to feature selection and machine learning models that solve classification and estimation problems in software development processes. In other chapters, we explore literature relevant to the topics described there, i.e., data acquisition pipelines in Chapter 2, database management as central topic of Chapter 3 and Chapter 5 for information visualization.

1.2.1 Software development, Agile and SCRUM

As a complex process, people approach software development in different ways and through various methods. Usually, a development process starts off with determining the *requirements* of the system and the software that is developed to run on that system. This requirement specification clarifies what the *product* in its entirety should do. Several more steps follow to analyze the requirements into diagrams and an initial design of the software and its *architecture*, consisting of a high-level structural overview. These structures become concrete by programming *code* that performs the necessary actions. The code of a component in the architecture integrates with other subsystems. Then, *tests* are employed to find potential problems at various levels of inspection, e.g., with *coverage* of enough lines of code. In this phase, conformance checks compared to the specified requirements are paramount. Once the product satisfies these tests, the product is made operational through *deployment*—such as installation or publication—and thus made available to the end user. Thereafter, frequent *maintenance* of the system is necessary to keep the product functional in a changing *ecosystem* in which the software is placed.

When these steps are taken one after another, the process is classically considered to be a *waterfall model* where each phase depends on the results of the previous stage. The output of each development step is then fixed, such that it is available as a reference work to evaluate the correctness of later steps, e.g., the verification of requirements during a test and validation of the desired functionality. Thus the process works in one direction when strictly adhering to the conceptual waterfall model [1]. This is commonly seen as a formalistic, contract-based top-down system.

The rigidity of this process becomes cumbersome when users or other involved people formulate new desires and requirements during later steps, which are then difficult to include in the process. Even if the requested changes are postponed until a later *milestone* or version release, it takes a lot of effort—thus more time—for the new idea to be fully developed. This is problematic if the idea is related to other, well-developed *features* that already meet requirements in providing desired functionality and therefore should not break. Similarly, the more important the change is to the product, the more difficult it is to implement in the existing code. Furthermore,

testing takes place very late in the process, which introduces risks of hidden bugs that surface after a long time, with no opportunity in the model to act upon this situation. Consequently, when strictly following a waterfall model, developers spend many hours looking back at specifications and code written long ago—potentially by others—with no clarity regarding original intentions. This brings along cognitive burden on top of the common workload [2].

Faced with these practical shortcomings, software engineers have considered alternative processes for development. Recent software development frameworks or methodologies apply a frequent, iterative cycle of development. Another important factor of modern software development is the increased attention to individuals and teams, less on intermediate *artifacts* that specify requirements in complete detail. Team interactions include collaborating on code, holding informal conversations to inform each other on progress and meeting with end users or representatives to discuss newly-developed features rather than formal documents.

These proposals accumulate into a short list of values, known as the Manifesto for *Agile* Software Development [3]:

- **Individuals and interactions** over processes and tools;
- **Working software** over comprehensive documentation;
- **Customer collaboration** over contract negotiation; and
- **Responding to change** over following a plan.

The manifesto continues to explain that there is still value in the latter topics, but the activities mentioned first in each item are considered more valuable.

Several novel software development methods are designed with these principles in mind. Generally, they focus on describing the core of their approach—given that not all situations, in the spirit of the Agile Manifesto, should be documented fully—often explicitly allowing the method to be extended or adjusted accordingly. Agile methodologies are typically flexible, with no preference for specific software that aids planning [4], while not restricting additional systems that support the specified values and goals [5]. One of such Agile software development methods is the SCRUM framework [6], its name being a metaphor derived from the team sport of rugby. We given an overview of the basic SCRUM framework. Many adaptations exist that focus on different team sizes, such as scaled Scrum, Scrum of Scrums and Scrumban. These variants often play a role in multiple teams working on the same project, for example in our case studies in Section 1.2.3.

In SCRUM, definitions are given for roles, events and artifacts. A small development team may consist of, e.g., developers, architects, analysts and planners, but any person can take on one or more of these roles, improving their skills in practical situations. One person with a fixed role, operating between the team and the organizations involved during the development, is the Product Owner (PO). The progress toward a valuable product falls under the responsibility of the PO, who oversees several artifacts. The team also has a Scrum Master (SM)—a role which may alternate between developers or be appointed to a coach—who leads the team in implementing the SCRUM framework and removing *impediments* that hinder the progress.

SCRUM defines a number of events that take place as part of a cyclic, fixed-length time span known as a *sprint* [7]. Each new sprint typically starts after another finishes, but before a sprint, the team and PO hold meetings to improve and select the planned work for the sprint,

respectively during a *refinement* and *sprint planning*. During a sprint, the developers briefly meet every working day at a fixed moment in a *Daily Scrum*, where they customarily discuss what they have been—and are—working on as well as any impediments that they encounter.

A sprint ends with two further meetings, namely the *review* and *retrospective*. The review is meant to allow the team to present, demonstrate and discuss the results of the sprint with representatives of the end user. Through collaboration, further work toward a finished product is inventorized. The retrospective is meant for the team members themselves to improve the effectiveness of the process. They consider how the previous sprint progressed and discuss how to keep their focus on the important elements. Additionally, changes may be made to the way the team uses SCRUM. The general workflow of a SCRUM sprint, including the events surrounding it, is shown in Figure 1.1.

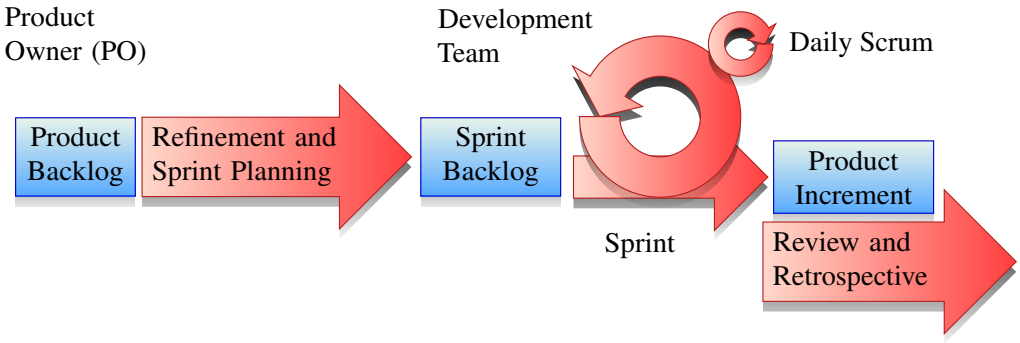


Figure 1.1: Workflow of a sprint in the SCRUM framework, showing events (red arrows), artifacts (blue) and the broad scope of some of the roles.

The team and PO self-manage the amount of work through several artifacts that list potential improvements to the product [8 app. 2]. One of the tasks of the PO is to collect desired features mentioned by the users or their representatives and place them on the *product backlog* (PB). The work items on this backlog are often called *user stories* that describe what the feature request entails in a simple format. Refinements bring forth further details and breakdowns into smaller stories, related to each other through links or *epic* tasks. The team regularly gives user stories a relative sizing, e.g., *story points*, indicating the complexity of development, based on available information and compared to other stories [9]. Once a user story is prepared enough to be *ready*—with the team agreeing that the work is not too difficult at the current time—it is added to a *sprint backlog* (SB), where work of an upcoming sprint is tracked.

The members of the development team commit themselves to collectively work on the selected features that make up the *sprint goal* (SG) objective. This is a subset of the future work on the product backlog, which has its own, encompassing *product goal* (PG). To find if work fits thematically and in terms of workload, diagrams like *burndown charts* help track the progression. The team also has a commitment to a unified, measurable way to determine if a task is complete. This *Definition of Done* (DoD) also describes quality measurements that changes made to the product should meet, such as unit tests, associated coverage of the lines of code, complexity and code style. The user stories that are completed according to this definition make up an *increment* of the product. Increments are provided to the users, perhaps as milestone versions. In Table 1.1, an overview of the artifacts, roles and commitment objectives defined by SCRUM is given.

ARTIFACT	ITEMS	RESPONSIBLE ROLE	COMMITMENT
Product backlog (PB)	Stories	Product Owner	Product goal (PG)
Sprint backlog (SB)	Refined stories	Development team	Sprint goal (SG)
Product increment	Features	Development team	Definition of Done (DoD)

Table 1.1: Overview of artifacts defined in the SCRUM framework, what they consist of, who is responsible for the contents and how the progress is agreed to be measured.

1.2.2 Machine learning, pattern recognition and predictive analytics

The study of algorithms that automatically analyze large amounts of data to perform tasks is commonly known as *machine learning* (ML). In this field, many different types of problems are under consideration, Sometimes, solutions are provided that find new application in generalized settings. Machine learning typically considers problems where entries—or *samples*—in a *data set* describing some kind of object, situation or event are in need of a *label* that describes the object in a numerical or categorical manner. A solution is to perform this labeling operation through a predictive method that provides a statistically likely answer. These algorithms, also referred to as *models*, are widely considered to be form of *artificial intelligence* (AI), where a computer is able to demonstrate skills similar to cognitive beings—such as humans—when it comes to understanding information and acting upon it [10]. This broad field is constantly changing, with novel concepts, algorithms and applications rapidly being developed.

Machine learning depends on the availability of a comprehensive data set, where there is a sizable quantity of entries as well as plenty of qualitative aspects of each record through relevant attributes. The data set should reflect a broad distribution of possible samples, encompassing uncommon instances and avoiding bias due to overrepresented cases.

In order to obtain an adequate data set, machine learning is combined with data acquisition methods to obtain enough raw data to start with. Usually, this data needs to be structured and filtered in order to select appropriate attributes for the eventual data set. Sometimes, *data mining* helps with extracting new properties from the data through the use of similar learning algorithms used in machine learning [11], but not necessarily with the goal of labeling in mind. This use of preprocessing algorithms is also found in *dimensionality reduction* [12], where attributes that do not contribute to the meaning of the sample—through some inherent metrics—are filtered out or combined into more relevant *features* that describe measurable observations about a specific sample in a data set. Additional *feature selection* helps with making the data set more refined by working on a relevant subset of the features [13].

The primary machine learning problems when it comes to making predictions are *classification*, *estimation* and *clustering* problems, where the goal is to find a label for an unlabeled sample that is either selected from a limited set of classes, from a numeric range or by grouping similar samples together, respectively. A *supervised learning* algorithm is able to use a (random) portion of labeled samples from the data set to learn patterns and understand relations between features in order to generate better labels in the future, in a process known as *training*. The other portions of the data set are then used to *test* a trained model with a similar distribution or as *validation* that the model was well-tuned. In contrast to these processes relying on already-known target labels, *unsupervised learning* makes use of samples without labels, which is applicable to clustering problems, for example.

A well-known group of ML algorithms typically associated with classification problems in particular are neural networks (NNs). These models are architecturally composed of layers, where the sample is fed as an input to a layer, which performs mathematical operations based upon weights and potentially other variables stored for that layer, producing an output of possibly different dimensionality. Multiple layers lead to larger networks, with the final layer providing a classification or estimation [14]. During training, the output is used to adjust the weight factors through a learning algorithm such as *backpropagation* [15], hopefully improving the accuracy of the labeling. With more layers and specialized architectures of each layer [16], such NN models are contemporarily known as deep neural networks (DNNs).

Aside from determining labels through classification and estimation algorithms, there is an interest in finding similar situations in the data set. Especially when it comes to repetitive, temporal data, regular occurrences of situations provide relevant insight in the underlying meaning of a final classification. Research in these regularities and peculiarities is known as *pattern recognition* (PR), which supports the labeling task by providing strong, new features in the data set. Pattern recognition on its own also helps with further analysis of the data set and extraction of clusters or related information [17]. Separately, *anomaly detection* also improves understanding of why rare situations take place, how to describe them in the data set and how to approach them in the further machine learning process [18].

Another method of advancing knowledge about patterns and relations is *regression analysis*, where data points are estimated using a function that closely fits most of the observations [19]. If the regression function is a line, then we specifically deal with linear regression and a *trend* becomes visible, but analysis with higher polynomials and other functions exist as well. One can also include data points in probabilistic distributions and generate new points based on probability density functions, such as with the Monte Carlo method (MC), using random number generation to draw potential new samples during many simulations [20].

While some of these statistical approaches are more remote from the concept of machine learning, they still play an important role in fundamental research as well as the construction of larger models. One challenge in machine learning is making results *explainable*. This means that we do not just obtain a classification or estimation, but are also able to trace back how model generates the label, e.g., which inputs end up being most relevant in the internal computation. Enhancements to a model should allow for more means of explaining the output instead of hindering this. Similarly, models constructed from existing algorithms in order to form an *ensemble model* that combined multiple outputs—for example through a majority vote—should still expose how the inner methods operate [21]. Recombining algorithms at a meta-level may help with specific objectives, such as in the analogy-based effort estimation algorithm (ABE), which identifies which attributes are most prominent in predicting the workload of developing code [22]. In Table 1.2, we provide an overview of the mentioned algorithms and the categories of problems that they are usually applied to.

In summary, machine learning attempts to achieve multiple predefined objectives for a specific application, where accuracy, bias reduction, relevance of features and explainable models are important goals. Additional factors such as privacy and other sensitive information are relevant as well. Proper care should be taken in this regard when constructing, training and evaluating the algorithms. Together, ensemble models allow more problems to be solved while increasing the descriptiveness and retaining forecasting power. For business-oriented problems, such as the software development life cycle, this logical approach yields an innovative process known as *predictive analytics* [23].

ALGORITHM	CLASSIFICATION	ESTIMATION	TREND
Neural network (NN)	✓	✓	✗
Deep neural network (DNN)	✓	✓	✗
Linear regression (Lin)	✗	✓	✓
Monte Carlo simulation (MC)	✗	✓	✓
Analogy-based effort estimation (ABE)	✗	✓	✗

Table 1.2: Types of machine learning and statistical algorithms, with an indication of applicability to some problems. Through discretization, use of raw outputs or other enhancements, some algorithms are usable for more goals than indicated here.

1.2.3 Case studies of workflows

Software development and machine learning are both very widely researched fields, with a large scientific foundation in understanding their functioning. When it comes to specific frameworks, methods and models, it is essential to find out how well one can apply these in a practical setting. The SCRUM development method allows teams to make precise adjustments in order to improve interactions and results. Furthermore, organizations use different ecosystems for SCRUM. This includes tracking backlogs and sprint progress, means for collaboration—such as version control systems with code review functionality—quality control and build platforms. Differences in selected systems mean that it is not straightforward to perform data acquisition and pattern recognition. Careful consideration is required in order to allow for generalization. We need to first discover how approaches of teams and organizations affect the way the data is stored, before considering what this entails for our predictive research.

We introduce two software development organizations from the Netherlands: Stichting ICTU and Wigo4it. Both organizations develop software applications exclusively for other government agencies, but are officially separate, non-profit entities. ICTU was established by the Dutch government as a foundation to be an implementing body (Dutch: *ICT-Uitvoeringsorganisatie*), while Wigo4it is operated by the four largest municipalities of the country.

These two organizations implement their own interpretations of the SCRUM development methodology. Additionally, they have their own work culture and regulations, which may lead to differences in teams, for example how much they need to collaborate and the number of external independent contractors among the developers. We observe how some of the development teams work during a sprint, in particular the approach to meetings and the methods used to track their progress. We interviewed some key roles in teams and support staff, as a means to survey what they consider to be critical processes, meetings, technology and data points. This helps us expand upon the fundamental concepts of software development and machine learning. We construct a plan to collect relevant data points while retaining a generalized view. We discuss the most important aspects of the two case studies in this section.

ICTU

In order to gain initial insight in how the development teams at ICTU* apply SCRUM during their work and how the systems support this process, we observed a specific project's meetings. At the

*https://ictu.nl/

start of our research, we followed two teams working on the same project in a scaled manner, with the first team focusing mostly on the user interface and the second team handling mainly architecture and data interfaces; both are not limited to these expertises. The teams use Jira [I] to track user stories and develop code which a developer updates in a Git repository [II] stored on GitLab [III], reviewed by peers before being made available on a test branch. Automated tests take place at all stages of development through Jenkins [IV], with results showing up in SonarQube [V] and a quality monitoring tool developed by ICTU [VI]. A version is released to the client for further acceptance testing and potential deployment to a production environment.

The schedule of a sprint for the two teams is mostly according to the SCRUM framework, with refinements sometimes preceded by additional pre-refinement meetings to further work out stories on the backlog for the project (PB). In order to facilitate sharing knowledge between teams, some meetings are held together, such as an overall sprint planning followed by a per-team planning to determine the team's backlog for that sprint (SB). The review is also held together, in order to demonstrate all the new features. The retrospective follows a similar format as the planning, with the two teams dividing after a collective meeting, but also discussing their findings across teams. The Product Owner (PO) takes the leading role in most meetings, although during planning some team members can act as a "story owner" to select fitting stories for their team. Also, the per-team Daily Scrum is operated by a team member at a large screen. In particular, the retrospective is coached by the Scrum Master (SM), who moves between teams when they are split up.

Additional meetings take place to work out specific details, such as sparring groups and technical meetings, which sometimes involve other experts in the organization, such as when it comes to automation, design and quality control. ICTU also organizes cross-team discussions on particular topics known as *guilds*. Most guilds are available for everyone, though usually aimed at specific roles.

When it comes to the systems in use, the teams usually try to spend only the necessary time to optimize them, keeping the focus on the product itself rather than meta-development. One problem they face is the performance provided by the development platform used by ICTU, back then known as BigBoat [VII]. Certain impediments caused by this limitation are escalated to those responsible for finding a solution, such as a support team or the software delivery manager (SDM), a role specific for ICTU, who ensures that the product can be released at various milestones.

Aside from the normal user story format, additional metadata is stored in Jira, such as story points, priority—often also achieved by ranking the stories relative to each other—and status, including readiness, i.e., how far into the refinement process the future task is. We find that sometimes administrative changes to the status of a story are made during a meeting after the story was already worked on, such as marking a story as done during a Daily Scrum the day after, which also leads to the change being registered by a different team member.

Eventually, data from the two teams of this overview did not become a part of our data set, which avoids bias on their specific work method or an early change of behavior due to the development team being observed as part of an experiment [24]. We however found that other teams at ICTU also have to deal with limitations of their development platforms. We validate that they make changes to progress trackers, in particular marking stories as done, during meetings, by cross-referencing these moments with teams' reservations. Despite these potential differences between actual events and registered data, we consider it possible to develop a data acquisition pipeline for the goal of classification and estimation of elements of the process. By focusing on lower time resolutions for stories, sprints and backlogs, threats to validity are avoided. We initially construct this pipeline for ICTU, while keeping generalization to more organizations in mind.

Wigo4it

At Wigo4it[†], we initially find that the application of the SCRUM framework has many parallels to how it is used at ICTU, with both organizations developing products for national and local government agencies. Differences do exist at detailed levels, such as the systems in use that enable software development. For Wigo4it, all teams make use of Azure DevOps [VIII], a platform that was also formerly known as Team Foundation Server (TFS) or Visual Studio Team Services (VSTS). This system allows tracking user stories in the form of work items, with similar metadata fields for priorities, effort values using story points, and state of the work item, among others. TFS also provides collaboration on code changes and building increments for tests on a central Jenkins instance. Quality control results are stored in SonarQube much like ICTU does, but without additional reporting tools.

A major difference between ICTU and Wigo4it is the number of clients and associated projects that the teams work on. Whereas most teams work on different projects at ICTU, Wigo4it develops a central system for providing social welfare assistance to beneficiaries, with changes requested by municipal governments included into the same product. Individual teams work on components of the system or perform testing and deployment on a task-based level. Still, SCRUM is central to the teams' working method, including the roles and events belonging to the core framework.

Wigo4it also regularly holds guilds, with interested parties meeting to attend presentations on specific topics. This fosters innovation and enables developers and other roles to consider more of the larger design of the product. Aside from the cross-organization guilds, most of the formal inter-team coordination is handled via intermediaries.

We include the data from the teams at Wigo4it in our research, alongside ICTU. This approach provides a helpful perspective on potential differences in working methods, including team compositions and development ecosystems, allowing us to augment our data set with more instances. Specifically, this broadens our scope from just a single organization, which is taken in consideration at all stages of research process. Ultimately, this diversification enables more cross-validation in our approach.

1.3 Design scope

From our initial findings during the case studies as well as earlier proposals, we observe that modern software development produces a rich set of data. This data should be utilized in order to support and improve the trajectory of the software development process toward a finished product. Our research is motivated by contributing novel techniques that support the improvement of the quality of software development processes.

Software development processes make use of various systems to monitor progress and keep an overview of software issues to resolve. We reveal the patterns in the considerable amount of data produced during the development process, which in turn improves understanding of this process. Because the data is dispersed across the systems and the focus of the development team rarely deviates from the main tasks, this data is often left unconnected and undiscovered. By combining sources and considering events across longer time spans, we find emerging results that, when identified and put into practice, help teams and key people in an organization make informed decisions more swiftly.

[†] https://www.wigo4it.nl/

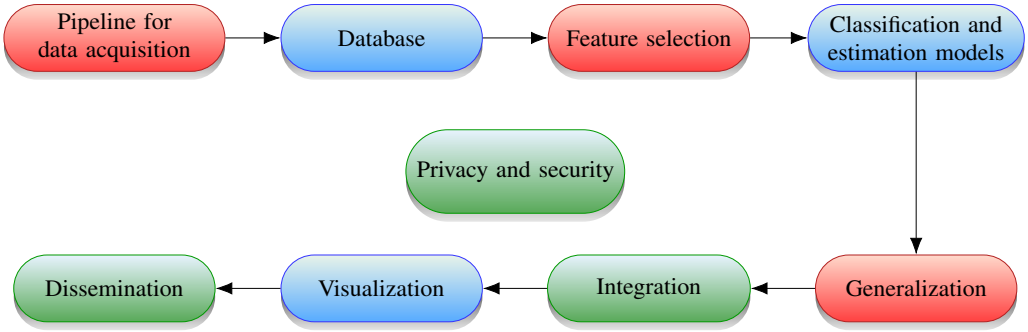


Figure 1.2: Aims and requirements as steps that are part of the design scope for the Grip on Software research, showing data mutation processes (red), technical artifacts (blue), external considerations (green) and some dependency relations.

Based on this scope, we further define the methods and objectives of the Grip on Software research project. Each of these objectives builds upon earlier ones in order to formulate the entirety of our approach. In Figure 1.2, we display the aims of the project and their relationship to each other. This includes data acquisition, database construction and feature selection to feed machine learning models. These components and processes are followed by steps toward generalization, integration, visualization and finally dissemination. As a general requirement, we consider privacy and security of the data during all phases of the process. We further specify these goals by describing the following requirements for each of the subtopics:

1. **Pipeline aimed at acquiring a data set for ML**: Machine learning—based on various algorithms like neural networks and probabilistic simulation—has shown strengths in various fields when it comes to finding relevant patterns and understanding diverse data sets. As the number of SCRUM projects that join our research increases and their completed sprints accumulate, we let our data set steadily grow over time. This dynamic approach allows for the models to be trained further and become more generalized. An integrated pipeline facilitates data acquisition and feature extraction, supporting frequent and rapid model training and validation.

2. **Database**: A pipeline must also contain a consolidated database with data from multiple originating systems. This database enables us to make links between different types of entities that we preserve. The augmented data allows us to fulfill a need of automatically and easily finding emergent patterns from large amounts of related information. For example, after collecting code quality measurements, we are able to relate them to code changes that took place before them; these code changes implement specific user stories which the developer marks as done. All these events occur during a short time span in the same sprint, but would otherwise not be linked to each other.

3. **Feature selection**: An essential part of machine learning with large data sets is to properly select features that best describe the characteristics of the artifacts, events and processes behind them. We evaluate how the features contribute toward a certain classification or estimation. Along the way, we resolve potential errors in the data, such as when developers indicate that a certain task is currently unfeasible by awarding it with an abnormal number

of story points, as described in Section 1.2.1. Subsequently, we produce a feature set that meets criteria regarding reduction of dimensionality, accuracy and understandability for our research and feedback purposes, which are relevant in machine learning studies as introduced in Section 1.2.2.

4. **Classification and estimation models**: We aim to train multiple models with this data, targeting classification and estimation of sprint outcomes as well as estimation of backlog sizes over time. From our observations, providing solutions for both decision problems is most beneficial to our objective and thus a logical step in our approach. The models take into account that possible workload differs between teams and sprints due to a wide variety of factors, not limited to team size, focus of development activities and current planning status. The models use the data sets to learn from multiple teams while taking into account the current and recent sprints, specifically significant situations involving story backlogs. This leads to a contribution to several decision problems in the field of Agile software development planning.

5. **Generalization**: By including project data from multiple software development organizations to our data set, we are able to further cross-validate our approaches and models. As such, we expand the data acquisition to encompass aspects of more SCRUM development processes, leading to a heterogeneous and well-balanced data set. This also allows for generalization of the models, leading to applications in more situations. Furthermore, a larger data set means reduction of bias and overfitting due to practices that are limited to few teams. The introduction of independent elements from different teams and organizations in our data set helps bring forward the application of the research to a wider audience.

6. **Integration and tracking**: It is important to consider how to construct and integrate algorithms for a real-world use case. When it comes to a decision-making process part of SCRUM, developers desire qualitative and correct data to support arguments and discussions. Outcomes of algorithms should be reproducible and transparent. The factors used as inputs to a neural network—or in general the data set for training such models—should be easily understood. This means that each feature used in the data set should have a clear correspondence with the originating data and any additional transformations, query selections and other operations. As such, our pipeline needs to track metadata across different stages.

7. **Visualization**: To increase the impact of the results produced during the Grip on Software research, we want to provide the development teams access to intuitive representations of the data, the features and the outcomes. We develop information visualizations that display situations and events that are relevant to teams and projects. Each visualization focuses on a certain category of decision problems and provides systematic support underlying a SCRUM software development process. By performing proper data selection, we zoom into the events and situations from different viewpoints. The visualizations are not simply static renderings, but instead allow for interaction. The user further manipulates what is being displayed through filter controls, specialized zoom handling, dragging of elements, hoverable tooltips, timelapse animations and other motion-based actions. Through our visualization approach, we transform our detailed data set into extensive forms of information, that people use to gain knowledge when they are properly presented to them.

8. **Dissemination**: Next to visualization, we provide the results in other formats as well. Through integration with existing quality control systems, the predicted classifications and estimations are brought to a central point where developers are easily able to include it in their workflow. We link back to the visualization hub and provide details about the specific prediction at easily available places. By writing an application programming interface (API) with an open specification, we allow more programs to use the predictions without much work. Additionally, the features are made available as an open data set, thus enabling further external research and dissemination.

Throughout the domain-aware pipeline, additional requirements focus on **privacy** and **security** considerations. Sensitive data is collected under agreement with the organizations, clients and teams. We have a need to temporarily collect personally identifiable information in order to link the profiles of developers spread across systems. Seeing each team member as unique allows for some deeper understanding of the data. However, it is not our goal to perform analysis on a person-by-person basis. The data needs to be accessible only where it is necessary. We use one-way encryption techniques in order to turn details like names and email addresses into pseudonymous records early.

In the end, we intend to provide valuable feedback to the *stakeholders*. They are the most relevant parties who have an interest in the software development project. Stakeholders interact with each other through various decision-making processes and benefit the most from the progress. In our case, the primary stakeholders of the research are the members of the development team, including Scrum Masters, but also quality control engineers and project managers or similar roles. Meanwhile, the successful development and release of the software product benefits multiple distinct entities, namely the entire software development organization, the users and the client. The latter acts as the eventual owner of the product when the development organization fulfills a contractor role. Our research helps shape an improved digital information ecosystem where software development becomes more reliable, standardized and cost-efficient. These motivating factors are relevant in particular for governmental software projects, but also for software development in general.

1.4 Problem statement

To recap on the contextual description in Section 1.2.1, SCRUM is an Agile software development framework aimed at improving a software product according to the needs of the end user, by gradually implementing the most desired features in order to make increments. Additionally, by evaluating the product and the process frequently, adjustments can be made early on. In order to perform such reviews and retrospectives, development teams often make use of both objective metrics such as project progress tracking and code quality indicators, but also individual experiences, e.g., how difficult a story ended up being. This mix of expert judgment and in-depth evaluation helps improve the eventual product and customize the process.

Due to its nature as an iterative process, SCRUM has many repetitive events and actions. This produces loads of data which, when put to good use, may further improve the review and retrospective activities. In fact, by analyzing patterns from emerging properties and training algorithms to learn from a broad data set of teams working on software projects, we propose to take a step further in helping stakeholders make swift decisions during the development cycles.

The objective of the Grip on Software research project is to allow software development teams and other stakeholders involved in the process to receive the most relevant indicators from large amounts of data produced from their own development project. These indicators are then backed by analysis on a larger data set with different projects and organizations. Altogether, we aim to improve planning of story points during sprints and backlogs as a whole, even early on in the process.

The following problem statement is central to our research: "How can extraction and analysis of measurable events during a SCRUM software development process as well as other qualities of the product and team be used to significantly improve the predictability of practices employed at software development organizations?"

This question leads to several more questions that each address a different aspect of our research approach. The questions for each topic also have more specific sub-questions which expand further upon the steps and objectives for the subject matter. The main research question, the topical questions and their sub-questions are the foundation of the multi-faceted research that together form the Grip on Software project. We briefly describe the importance and aims of these questions in this section. The research questions are further addressed in their own chapters. We introduce the specific questions and supplemental points here:

RQ1 How can we reliably collect data regarding SCRUM software development practices and consolidate the resulting artifacts inside a central database that constantly grows and allows adaptable queries?

- **RQ1a** Based on relevant design principles from distributed data systems and agent-based networking, how do we design a data acquisition pipeline applied to multiple organizational ecosystems?
- **RQ1b** Which objectives related to inspection, curation and disclosure do we achieve with a data acquisition pipeline when taking our formulated requirements in mind?
- **RQ1c** How can we model the relationships between different data artifacts acquired from dynamic systems regarding SCRUM software development in order to properly deduce information about their state during different sprints?
- **RQ1d** Which technical challenges are relevant when deploying a database component as part of a pipeline for multiple organizational ecosystems?

Our first topical question relates to the technical aspect of the Grip on Software research. In order to acquire relevant data from SCRUM processes and to safely store and update the data with incoming updates, it is necessary to have one or more computing systems in order to accomplish these goals. Additionally, we formulate requirements that potential designs for such systems should adhere to. This question is important on its own, because it lays down the foundation for further research, which depends on the proper collection and storage of the data, that, in turn, forms the input of the further analysis.

There are two technical topics that we distinguish primarily in these specific questions: (i) the acquisition of data from the source systems at the organizations and (ii) the construction of a database with a query-based interface that supports selecting data. Our research into these topics

includes literature reviews in order to find relevant approaches. This leads to development of novel techniques as well as potential for reuse of existing systems that are applicable to these goals. Both topics have several phases in selection and development: designing principles and models, directing systems to work according to specified objectives, de-duplicating approaches for similar source data and deploying the compound system in several software ecosystems.

The goal of resolving these sub-questions is to construct a complete pipeline that works in multiple circumstances and according to functional and non-functional requirements. We validate the success of this approach through metrics from experiments that demonstrate the performance and applicability of the GROS pipeline when it comes to representative workloads of acquiring, storing and retrieving real-life data, in the context of complex models that describe events of a SCRUM process.

RQ2 How can we improve the predictability of SCRUM software development practices, specifically the progress of sprints and backlogs—based on analysis of data selected from the development process—and how do we validate our approach?

- **RQ2a** Which features can we select based on ongoing data from the software development process that are most indicative of the progress?
- **RQ2b** What kinds of learning algorithms can we introduce to this problem, which learn from these features and provide feedback on what kinds of decisions can be taken?
- **RQ2c** How do we predict the likelihood of timely and properly finishing a currently-running sprint or a longer-running period aimed at resolving a product backlog within a development project, even before it has started?
- **RQ2d** How do we validate that the predictions and recommendations are within our expectations and based on relevant, explainable factors?

Our second research question describes the aspects of machine learning and pattern recognition, which is a central theme of the Grip on Software research. Based on the database filled with data acquired through our pipeline, we perform data analysis related to the main research question, which pertains the predictability of SCRUM processes. Through our analysis, we formulate and select relevant features based on scoring algorithms.

These features then form a data set for novel machine learning algorithms. We base these algorithms on known-to-work classification models as well as reimplementing them on earlier estimation models specific for this purpose. We also implement regression models and generative algorithms that make use of different time scales that tie in with SCRUM. We determine several predictive tasks to be relevant; for SCRUM sprints, we perform classification based on planned story points as well as estimation of story points that could be finished during that sprint—and provide predictions for these tasks before the sprint has started—while for longer periods, we generate future sprints that simulate working on backlogs in order to predict a date at which the work is completed.

An important quality of the machine learning algorithms is to allow the predictions to remain explainable to the development team and other stakeholders, by including as many details and metrics on how the classification, estimation or generation came to be in the end result. We further perform studies into the workings of the algorithms, through the use of external accuracy measures and evaluating the effects of data set sizes and generated distributions on the quality of the predicted outcomes of the sprints and backlogs.

RQ3 How can we effectively introduce visual representations of results and recommendations from our analysis of data collected from the SCRUM development process to the involved parties?

- **RQ3a** Which concepts and goals are relevant when designing information visualizations for patterns analyzed from a software development process?
- **RQ3b** How do we integrate results from predictive models within existing development practices?
- **RQ3c** Which effects do the introduction of results from predictive models and other intermediate analyses have on the development process, validated across multiple ongoing projects?
- **RQ3d** What is the overall assessment of the proposed information visualizations, considering automated measurements for usability and the adoption according to interviews and surveys?

Our third and final question describes the presentational aspect of the research performed in the Grip on Software project. We wish to allow the producers of the data—namely, the development teams and the overall software development organization—to gain new insights in what the events and patterns describe and uncover. A visual format of relevant data points improves the understanding of the overall research among users. In particular, we construct several information visualizations that we integrate in a hub. This helps us bring selections of data to another level by enriching them with context to form information, rendering it to form a viewpoint that reflects reality and augmenting the view with interactive controls, so that people gain knowledge from the visualizations.

We incorporate results and characteristics of the predictive models within specialized and generalizable visualizations. We also build visualizations that help in other situations involved in the SCRUM process as well as software development ecosystem management. The complete system from data acquisition to visualization of predictive patterns enables a feedback loop of actionable results. We perform several interview-style discussions and usability surveys as part of our user tests, but also consider proactive requests from teams for extensibility. We finally include automated and survey-based usability metrics in our evaluation of the novel information visualizations, which help determine if the visualizations are understandable for the people using them, including developers with color blindness, for example.

1.5 Pipeline components

A core contribution of our research is the development and deployment of the Grip on Software data acquisition pipeline. We construct the pipeline in order to collect relevant data regarding events that take place during a SCRUM software development process. The data is consolidated in a database which is designed to model the entities and relationships, using new links between previously disconnected artifacts. Using novel query templating functions and integrations, we extract features for a data set that is used for training machine learning models in order to find patterns in the data. Finally, we make the data processed by the pipeline accessible through information visualizations, which allow users to gain a deeper understanding of SCRUM activities at various levels of detail.

The sources of the data used in the GROS pipeline already exist as part of the software development ecosystem at the organizations involved with this research. From our case studies in Section 1.2.3, we observe that the two software development organizations use a selection of systems with various interconnections between them. However, commonly these systems operate independently. In order to fulfill tracking and automation for a SCRUM team, separate systems are often available. A digital version of the backlog and planning board is realized by an issue tracker. The team collaborates on writing code and tracking releases using a certain version control system. For quality assurance, build platforms are able to generate metrics on how well a particular version of the software product behaves.

These are just generic descriptions of the ecosystem in place at an organization, which our pipeline potentially encounters. A novel research approach is to consider not just one system as a single viewpoint of SCRUM activities, but to include many aspects from multiple sources. Additionally, when we specialize toward one product, we take similar solutions into account for other software. By using distinct components that only need to interact with a limited environment surrounding each of them, we deal with differences between the interfaces and the data they provide at early stages. This splits up the workload and simplifies the progress toward a uniform data set.

In Figure 1.3, we show a schematic outline of the GROS pipeline. We identify the systems that provide data on SCRUM activities. Then, the data acquisition pipeline gathers updates into a collection in a simplified exchange format. This allows us to import the newly-obtained data into a database. Another component provides domain-specific data analysis and extraction. This component either generates a data set for prediction using machine learning or provides data to interactive information visualization, both as separate components. These final components present outputs that are usable by development team members and other involved stakeholders.

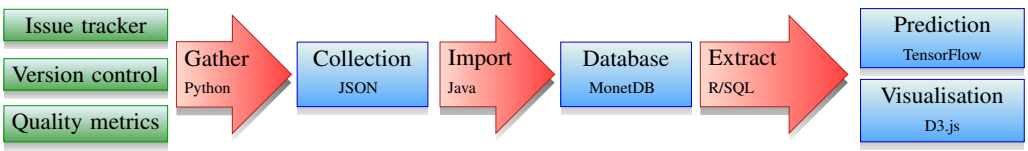


Figure 1.3: High-level overview of the full Grip on Software pipeline, with the primary SCRUM data sources (green), active components (red) and resulting artifacts (blue) highlighted.

1.5.1 Instances

We set up several instances of the pipeline during our research. We provide separate pipelines at each organization discussed in Section 1.2.3, with updates provided during deployments of new versions. At ICTU, we use the complete pipeline in order to collect data from a large number of software development projects, including some projects that are maintained by internal support teams. Many teams at ICTU have their own virtual network in which the development environment is situated, with one centralized issue tracker known as Jira [I]. We design and implement an *agent-based* data acquisition component, where each team has their own agent service for collecting data. Personal data is anonymized at the earliest stage. Further data processing takes place in an isolated network. The results from the predictions as well as the visualizations are made available to the teams again.

The other organization introduced in Section 1.2.3 is Wigo4it, where a score of teams work on a composite project. They all make use of an integrated development platform provided by Azure DevOps Server [VIII]. Here, we simply acquire the data at a single location.

Finally, an instance of the pipeline is maintained at Leiden University, where we assemble a combined data set. At regular intervals, the pipelines at the organizations upload data dumps to the central instance using secure exchange protocols and only after encrypting sensitive data. The pipeline at Leiden University is therefore deployed from the database component onward, which allows us to perform further analysis. This combined data set helps validate whether our approach would work at other organizations as well.

1.5.2 Non-functional requirements

The pipeline is designed to be reusable and extensible in order to fit with different software development ecosystems. Components are able to be arranged separately across internal and external networks. Additional configuration enables running on different virtualized platforms and selecting the systems to use as data sources. Detection of more data sources is possible while the system is active, simplifying the process of gathering relevant data.

Several programming languages, exchange formats and query template constructs are involved in the workings of the GROS pipeline. We use the strengths of each computing language in the places where they work best. By following common practices, code style guidelines and novel techniques for tracking data types and integrating functions, we support the maintainability of the components while promoting the use of state-of-the-art techniques.

These qualitative aspects of software are often called non-functional requirements. In Table 1.3, we indicate more aspects that we find important in the context of the GROS pipeline. We look into how we accomplish these requirements for the components, including the ones shown in Figure 1.3 as well as some subcomponents, such as the data acquisition agent, the secure export exchange and visualizations provided as plugins. We expand upon these components and their characteristics further in later chapters—in particular in Chapter 2—with the references to code repositories in Appendix A.

COMPONENT	TESTS	*Coverage*	PERFORMANCE	DOC	INTEGRITY	PRIVACY
Gather [a]	Unit	✓	Section 2.5	✓	Trackers	≈
Agent [b, c]	Unit	✓		≈	Config	✓
Import [d]	Unit	✓	Timing	✓	Trackers	✓
Exchange [e–g]	Unit	✓		✓	Dumps	✓
Database	Schema	✓	Section 3.5	✓	Dumps	✓
Extract [h]	Style	✗		≈	Database	✓
Prediction [i]	Builds	✗	Section 4.6	✓	Data set	✓
Visualization [j–s]	Integration	✓	Section 5.8.1	✓	Data set	✓
Plugins [t–v]	Typing	✗		✗	Jira	✓

Table 1.3: Overview of some non-functional requirements regarding the pipeline components, some measurable solutions and experiments that demonstrate how we meet these requirements and how they adhere to our criteria (✓ is good, ≈ limited, ✗ bad/missing).

We perform unit tests for components as well as integrated tests with an entire pipeline, both on a frequent schedule and when deploying new portions. By tracking which parts of the code are actually executed during such tests, we measure the coverage of those tests, indicating how well we were able to test all edge cases.

We also perform tests and experiments to measure the *performance*, i.e., find out how many resources the pipeline uses and whether the component provides the resulting data quickly enough for output or further processing. This is done during stress tests and representative workloads on the platforms that the components run on. In most chapters, we focus on demonstrating that tests and user interactions perform smoothly rather than reporting in-depth performance analysis.

Further, we assess the quality of the code style and the technical documentation provided with the components. In Section 2.4, these considerations as well as other topics such as generalizability of the components and applicability of continuous integration for rapid development and deployment of the pipeline into various software ecosystems.

Perhaps the most important aspect of the data acquisition pipeline is how it handles the collected data. The pipeline should not lose data or lose track of its state, so that it is able to recover its *integrity* from an earlier point in case of an interruption. Finally, privacy and sensitivity of data such as personally identifying information should be handled carefully. We do this by making certain fields unintelligible through one-way encryption of the original information.

1.6 Structure of this thesis

In the next four chapters, we address different themes, objectives and research questions related to the central topic of that chapter. Based on literature review, we discuss the primary concepts and relevant work, providing the necessary context. We introduce objectives, design approaches and implementations for our technical components and models. Through experiments, we validate our approach and provide results. Each chapter also comes with its own summarized abstract at the beginning of the chapter as well as specific conclusions with topical further research.

Most chapters of this thesis correspond to the technical components of the GROS pipeline. In Chapter 2, we discuss the data acquisition pipeline, with a focus on the initial components that collect and consolidate data artifacts. Chapter 3 continues the technical overview with a thorough description of the database design and architecture, which form the second pipeline component and artifact.

For Chapter 4, we shift our attention to the later parts of the pipeline, where we extract features using a novel database query compilation system, select the features and transform them into a data set for machine learning classification and estimation models, allowing us to provide predictive results. Chapter 5 then describes the development and deployment of information visualizations for analytical decision support and ecosystem management to help gain insight in SCRUM processes, using the database as a source for many relevant subject-matter artifacts, such as backlogs and sprints.

Together, these chapters discuss the placement of the research in the larger software development ecosystem, including the sources of data and the users who benefit from the outcomes. We introduce and discuss each chapter's central topic by referring to both the main problem statement and some of the specific research questions and sub-questions as proposed in Section 1.4. In Table 1.4, we outline the chapters as well as the associated research questions and components, which were briefly introduced in Section 1.5.

PART	TOPICS AND RESEARCH QUESTIONS	COMPONENTS
Chapter 2	Data acquisition pipeline (**RQ1a**, **RQ1b**)	Data sources, Gather, Collection
Chapter 3	Database modeling (**RQ1c**, **RQ1d**)	Import, Database, Extract
Chapter 4	Predictive analytics (**RQ2**)	Extract, Prediction
Chapter 5	Information visualizations (**RQ3**)	Extract, Visualization

Table 1.4: Overview of the principal chapters, the research questions that they focus on as well as the technical components and artifacts of the pipeline that play a central role in each chapter.

Supplementary to the technical solutions, we take a deeper look into the data artifacts that are applicable for the chapter. We describe the data sources, model the entities, determine new relationships between them, select descriptive attributes and complex associations, generate data sets and determine which features in the data set make a machine learning model or an information visualization more understandable and helpful to the user. This data-driven approach enables an objective, methodical and goal-oriented view in our research.

In each of the chapters, there are differences and similarities to the focus and setup of the various experiments, where we validate the approach, compare the functionality and behavior of the system with the stated objectives and provide results that are available at that point of the pipeline. For the chapters regarding the data acquisition pipeline and database construction, the measurements and results are often meaningful for the research project internally, relating to the performance of the system. The chapters on prediction and information visualization further lead to observations and evaluations that allow reuse by the stakeholders. We achieve this through multiple levels of detail and different ways to look at our data set, both visually and algorithmically.

We wrap up the body of the thesis in Chapter 6. Through the use of the research questions, implementations, experiments and results, we formulate a retrospective and discuss the main conclusions that encompass all the work described in the preceding chapters, by reflecting on the research questions, approaches and results. Additionally, we mention possible future research in the field of predictive software engineering.

Chapter 2

Data pipeline

GROS gatherer: Agent-based acquisition of high-frequency data updates within existing dynamic software development ecosystems

Abstract of Chapter 2

Introduction: Research into software development processes relies on collecting frequent updates to data stored in systems that help with planning, development and quality control. For this purpose, we design and implement a novel data acquisition pipeline that fits in the existing software development ecosystem.

Research questions: How can we reliably collect data regarding SCRUM software development practices and consolidate the resulting artifacts inside a central database that constantly grows and allows adaptable queries?

Abstraction: We assess which existing concepts regarding pipeline construction, data collection and distributed data processing are relevant for our objective. We consider existing designs with asymmetric communication between agents and a controller, including where the "knowledge" is kept, which connections to other systems are established, how the status is tracked and what kind of processing steps are possible.

Implementation: We implement several components that collect data from issue trackers, version control systems, code review, quality control and other platforms. The distributed setup allows early linking of related entities, encryption of sensitive data and portability within networked setups. The components are configurable through an interface and the status is tracked in a dashboard. Further steps toward database import, data analysis and information visualization have their own components with novel highlights.

Discussion: Experiments show that the pipeline components are able to work within different setups at two organizations and in a central research location. The frequent collection runs do not impact the performance of the ecosystem's platform during normal usage. Steps were taken to allow further generalization and adaptation.

2.1 Introduction

As part of Grip on Software, our research into patterns and outcomes within SCRUM software development processes, we find that there is a need to collect data about activities of software developers related to SCRUM principles. Often, multiple systems are used by development teams in order to handle various aspects of the development process. This includes project management systems that keep backlogs of user stories, bugs and other tasks to work on—usually referred to as an issue tracker—but also version control of code repositories, automated quality control checks of the code and other artifacts, platforms to test a compiled build of the code and other organizational tools for access control.

Limited communication exists between these systems. Often, a change in one system cannot be firmly related to another, such as a quality measurement for a code change based on the resolution of a user story. This makes it difficult to get a complete picture of the situation in the development process, let alone understand past SCRUM sprints.

A data acquisition pipeline helps mitigate these shortcomings within the software development ecosystem by collecting data from multiple sources and generating artifacts explaining changes in a simple format. We have a need for such output in order to compile a centralized, consistent database. From this database, we produce features and relevant data points for further analysis and reporting, respectively through pattern recognition—backed by machine learning and estimation models—and information visualization. The choices made in this first step are important, as they make it possible to reproduce the research within other organizational ecosystems.

We consider the following research question and a selection of two sub-questions regarding the data acquisition pipeline, which we also refer to as the GROS gatherer:

RQ1 How can we reliably collect data regarding SCRUM software development practices and consolidate the resulting artifacts inside a central database that constantly grows and allows adaptable queries?

- **RQ1a** Based on relevant design principles from distributed data systems and agent-based networking, how do we design a data acquisition pipeline applied to multiple organizational ecosystems?
- **RQ1b** Which objectives related to inspection, curation and disclosure do we achieve with a data acquisition pipeline when taking our formulated requirements in mind?

In terms of scope, the SCRUM framework is part of the family of Agile software development methods. Among other values, an important factor of Agile development is the focus on individuals and their interactions within a team, more so than the processes and tools that they use within their work [3]. When teams decide on their own how to arrange their development process, different decisions regarding the use of these systems are made compared to other teams and organizations. This level of autonomy should not be seen as an inconvenience to outsiders. Instead, each application of the development method provides unique insights. For a data acquisition pipeline that supports such a development process on a meta-level, flexibility is an important factor in effectively applying it within multiple ecosystems.

2.1.1 Ecosystem

An important aspect of the implementation of a data acquisition pipeline for research purposes is to properly integrate it into the existing ecosystems that it meant to be used in. In our case, different software development organizations have their own networks, physical computing infrastructure, virtualization approaches, data storage solutions and security measures. A pipeline should be generic enough to fit with multiple of such resources. However, an organization should meet some prerequisites of the pipeline in order to be relevant for our study.

Within the SCRUM software development framework, teams are typically able to fill in how they work on the project in a flexible manner. As an Agile software development method, SCRUM also places the focus on the people within the team rather than the software that they use. Still, the systems in use for development and release are helpful. The ability to track progress centrally in an accessible manner makes the physical "wall of cards" quite outdated. It is reasonable that the maintenance overhead is much lower with a consistent system. The applications used within software development should aim at decreasing such workload.

The research project has a similar goal and it is sensible that a data collection pipeline avoids such burdens. The purpose and functions of individual elements of the pipeline should be open and understandable. At the same time, they should not place a heavy load on the existing ecosystem.

Not every organization uses the same development applications and network solutions. When different systems are used for the same purpose, such as issue tracking or version control, the pipeline should be versatile enough to collect data from one or more of them using similar interfaces and provide interchangeable artifacts. We are then able to use the source data in a comparable manner later on in our analysis. An organization chooses their own approaches in implementing the development ecosystem on different layers, although these layers typically contain elements of the items shown in Figure 2.1.

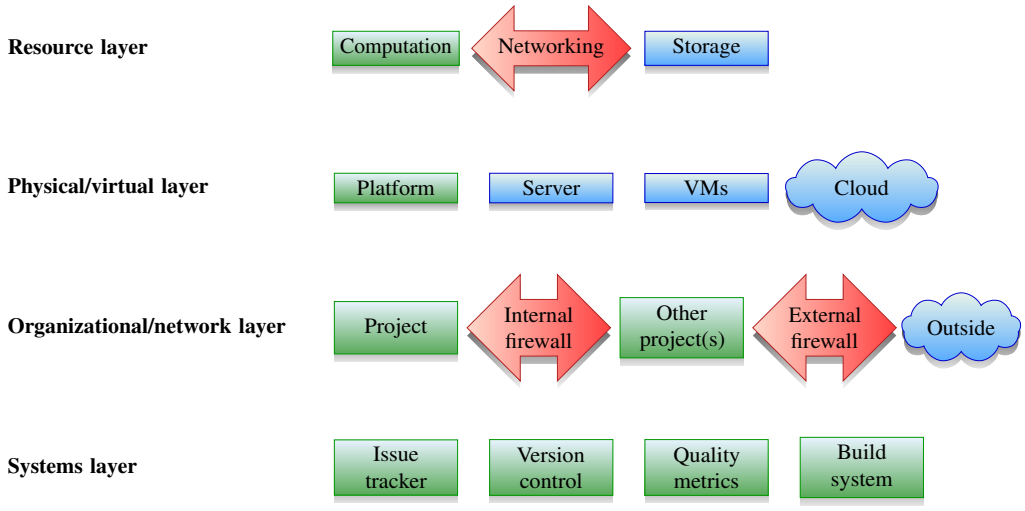


Figure 2.1: General layers of a software development ecosystem. Green blocks represent elements that are the main interests of our research approach, blue shapes are supporting elements and red arrows show interconnection elements.

2.1.2 Structure

This chapter's structure is balanced between theoretical concepts and practical implementation. In Section 2.2, we mention design principles and relevant concepts regarding the construction of a data collection system. We look into similar frameworks and abstractions of distributed systems, including agent-based networks. Some practical matters relevant for pipeline deployment within an organization's platform ecosystem are also brought up.

In the remainder of this chapter, especially Section 2.3, the focus is on the development, discussion and significance of the components related to the data acquisition, introduced in Section 2.3.1. This is the first step from the software development project management tools to intermediate data objects. Other steps within the process are discussed only briefly in this chapter. In Section 2.3.2, we discuss these components, with some technical details of the overall Grip on Software pipeline in Section 2.4. Section 2.5 describes some experiments and results obtained from practical usage of the data acquisition components. We conclude our findings in Section 2.6.

Other portions of the pipeline are further described—in context of the research—in their own chapters. Chapter 3 describes the database construction, Chapter 4 discusses feature selection for prediction models and Chapter 5 demonstrates the use of the data in information visualization.

2.2 Design

We consider the technical implementation of a data acquisition pipeline within the context of abstract design concepts. Often, data pipelines are constructed for a specific purpose within a predetermined environment, for example a business analytics context. There are however methods to make these approaches to data processing more portable and scalable. Therefore, we also look into existing approaches and design choices for certain use cases within organizational data acquisition. In this section, we bridge the gap between theory and established practice.

2.2.1 Distributed data systems

A data acquisition pipeline consists of multiple components that collectively allow to collect, combine, filter, mutate and reuse one or more data sets for one or more objectives, such as further analysis and reporting. When the original data sets—also referred to as *sources*—or the components that provide, i.e., the *sinks*, are separated into different systems, there is a rationale to also separate the corresponding components across systems. Sometimes this subdivision of concerns is a necessity, for example when these systems are required to live in different networks in practical situations.

The use of distributed systems is already established within different domains [25]. When it comes to analysis, there already exist frameworks for data processing and computation that parallelize tasks, allowing different systems to perform them simultaneously. Often, these frameworks focus on the tasks that take place downstream in the pipeline.

When it comes to handling multiple, large data sets, distributed data acquisition plays a major role within the pipeline. At each step, decisions are taken on how to collect, select, filter and alter data in order to make it usable in the next phase. The data sets provided by the sources come in different formats, which requires some overhead to make them usable in another component of the pipeline. Such transformations can be handled for each source. They could be split up according to the context, such as organizational units. This approach then suits existing network infrastructure.

Moreover, the data format plays a role in intermediate storage. Depending on how each component performs and interacts within the pipeline, data is often stored in a machine-readable format before further handling. The location of this storage also differs, ranging from disk-based files in directory structures to scalable database systems [26].

The means of data exchange between pipeline components is also based on the choices regarding the data format. A component could signal another that new data is available, potentially in the same format and message as the data itself.

Multiple dimensions of scalability exist in distributed systems for data collection [27]. First, the number of components—steps in the pipeline—is adjustable. Technical burden should be kept in mind here. A second dimension is the number of parallel workers of a single component. This number is flexible, assuming that the component was designed for reallocating such work. The third dimension is the number of storage locations for intermediate and final data sets. The effectiveness of distributing such tasks is limited by the availability of physical computing systems suitable for scaling the workload.

2.2.2 Agent-based communication

A distributed data-oriented system can be seen as a communication network. In this design, components are agents that perform semi-independent actions. Each agent interacts with some portion of an environment, including receiving input data. Once they have autonomously come up with an intermediate result, they make some parts of their neighborhood aware through signals. In some networks, the coordination of signals is handled by a control system. In another hierarchical network, the relations between agents depend on the stages they are in, similar to a pipeline [28].

Agent-based networks have concrete purposes within real-world applications, including motion tracking and formations of unmanned vehicles. The issues that come up in these practices are often well-modeled and lead to generalizable solutions for addressing effects of asynchronous updates and delays [29].

Another representation of data acquisition using distributed components is the Petri net [30]. In this type of network, nodes consume and produce tokens which are transferred between other nodes. This way, concurrency, control and other communication aspects is modeled for different components or agents within the network. Extensions to the basic Petri net enable the application of semantic attributes and activation rules to tokens, nodes and arcs within the network, making it more viable to follow the exact meaning at a less abstract level. This allows separation of workflows [31]. Detection and avoidance of deadlocks are also be possible through extensions [32]. Further, we are able to verify certain properties of the system that is described by the Petri net, enabling reasoning over a multi-agent system and making it safer to use [33].

2.2.3 Organizational approaches

Within corporate enterprises, data processing is often commonly associated with operational systems which produce a large amount of data, such as web servers. One approach to collecting the data from these systems is through an integrated system that passes messages around [34]. Often, the purpose of the data collection is to be further analyzed for business intelligence goals, leading to visualizations and reports.

In these circumstances, some terminology comes in place that lack a clear, generalized definition. *Big data* often does not refer solely to the size of the data set, but rather to the

complexity in establishing relations and discovering patterns. This complexity often leads to the introduction of data management systems that augment common databases, which are able to model or at least store the data types. Whereas a *data warehouse* stores data that is homogeneous in shape, leading to a common data model, often another option is taken where textual, visual, temporal and diverse data types are stored in potentially separate systems, leading to a *data lake* [35]. The disadvantage of this approach is that raw data is not easily processed and combined into a single model.

Other concerns for large-scale businesses in the area of data processing are auditing, access restrictions, on-premise versus cloud-based platforms and off-site archival systems [36]. The collected data might contain personally identifying information, which requires assessment and mitigation of the impact on privacy of users and staff members. Separation of components into virtual networks helps with reducing the possibility of unauthorized access. Still, both raw and processed data constitutes sensitive information for the organization. Accordingly, early encryption within the data pipeline helps addressing concerns regarding privacy and business information security.

2.3 Method

Prior to discovering what kind of patterns exist that provide indications of progress within a software development process, we need to determine which data holds these patterns and how we collect the appropriate data sets. As mentioned in Chapter 1, we focus on extracting these patterns from SCRUM software development processes, although the collection of data from multiple systems does not rely specifically on the presence of SCRUM terminology in the disclosed information. In fact, not all systems that we consider relevant to the process are specific to one development framework, as some are commonly used within software development for version control, source code quality checking and issue tracking.

The pipeline we design consists of multiple phases, where the data acquisition is the first step towards consolidation in a database, allowing the combined data set to be analyzed for various purposes within the domains of machine learning and information visualization. Figure 2.2 provides an overview of the pipeline at an abstract component level. In this figure, blocks represent states that data can be in, including potential data sources such as issue trackers, version control systems for source code repositories and quality control systems, which are used during the software development process. Later on in the pipeline, the blocks indicate intermediate artifacts and results. The arrows are actions that select, transform and move the data to its next state. These actions are performed by software components that we develop in several programming languages, including query languages. The code is fine-tuned for specific purposes within the pipeline.

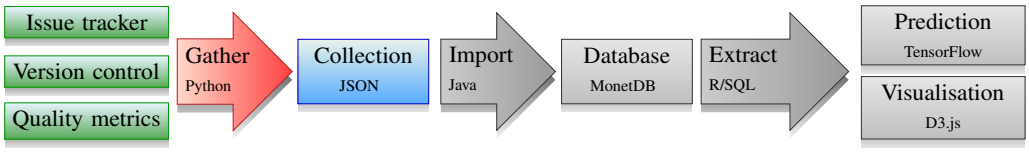


Figure 2.2: High-level overview of the Grip on Software pipeline, with the data sources (green), data acquisition components (red) and resulting artifacts (blue) highlighted. Gray elements are described in later chapters.

The main goal of our research is to improve software development practices by extracting, combining and analyzing data that would otherwise be left in separate systems, with no interconnection between the applications that the data originates from. As such, there are some important objectives when it comes to inspection, curation and disclosure of the data and the results:

- When data from different sources refer to the same entity, such as a team, project, component, developer or code change, we should establish this link as soon as possible. This way, we ensure that the equivalence or relationship is known for later components in the pipeline, when this was not available before the introduction of the pipeline. Additionally, this allows encryption of personal or project-sensitive data to take place early on.

- The pipeline has to work within different ecosystems. The existing infrastructure should remain unchanged, i.e., there are no additional requirements that stem from the pipeline. We split up components across networks without performance bottlenecks or difficult firewall configuration, by using typical connection protocols. Projects that are in separate networks have their own *agent* which collects relevant data. Intermediate artifacts are exchanged using an easily interpretable data interchange format to a central *controller*.

- Because the data comes from different sources, depending on which organization, team or project is involved, there should be an easy and versatile method of configuring the means of collection. This *configurator* indicates which web applications are approached by the data acquisition agents. It then stores the credentials to use and provides specific status information regarding the agent, which helps with control and monitoring of the pipeline.

- The pipeline's entire state should be easily visible within an overview. A *status dashboard* indicates problematic situations. This includes recentness of data collection by the agents as well as access to error logging. The dashboard also allows adjustments to scheduling in order to rerun specific tasks, for example if they failed.

There are further design principles and non-functional requirements that we consider when implementing the pipeline and its individual components. The pipeline should internally track when the most recent update of different data sources was successfully performed, so that we avoid loss of data. There should not be a performance impact on the usual development process of the teams included in the study, i.e., it should not lead to availability problems or other interference of the systems used as a source of data. Several more considerations help with reproducibility of the pipeline setup, such as testability and proper documentation of components and methods.

2.3.1 Data acquisition

We collect data regarding the progress of multiple SCRUM software development projects from various systems. In order to do so in accordance with our objectives, we introduce new components for our data acquisition pipeline.

The agent component connects to various application programming interfaces (APIs), using queries and filters to determine whether there are any changes since the most recent moment that data was collected for a project. The agent then produces structures that describe the fresh data using the JavaScript Object Notation (JSON) format. Table 2.1 lists some types of data collected from different categories of data sources. Additional components configure the data acquisition

Entity	Tracker	Version control	Review	Quality	Build
Issue (story, etc.)	✓	✗	✗	✗	✗
Sprint	✓	✗	✗	✗	✗
Comment	✓	✗	✓	✗	✗
Person (developer)	✓	✓	✓	✗	✗
Commit version	✗	✓	✗	✗	✗
Release tag	✗	✓	✗	✗	✗
Merge request	✗	✗	✓	✗	✗
Metric measurement	✗	✗	✗	✓	✗
Metric target	✗	✗	✗	✓	✗
Usage status	✗	✗	✗	✗	✓

Table 2.1: Types of data that are collected by the data acquisition components and the category of systems that provide this data (✓) or not (✗). Some types of data—metadata and dependent entities—are left out of this overview.

process, ensuring proper authentication at the defined systems. The components also track the frequency of the data collection runs. The agent passes errors through to a monitoring dashboard.

This setup allows us to deploy the components in separate ecosystems. For example, at one of the organizations involved in our research, ICTU, each software development project has its own virtualized network where resources such as a build platform, version control and quality monitoring systems are made available. The platform allows additional systems to be set up through a Docker-based setup [37]. The Docker instances were managed through a specialized interface known as BigBoat [VII] developed by the organization, although they later replaced it with another interface for practical purposes. Two data acquisition components—an agent which regularly collects data from known systems in the same network and a configurator that selects the systems and authentication. Together, the agent and configurator form the GROS gatherer, with one for each virtual network as a distributed system.

The intermediate artifacts are passed by each agent to a centralized controller within another network. A Secure Shell (SSH) connection protocol is specifically allowed through a firewall and adjusted to only accept secure data transfer from the agents. The controller provides additional environment information to the agents at the start of a data collection run, so that these do not need to be configured separately. The controller also acts as an additional checkpoint to adjust the collection interval, as a form of a *preflight* checklist. If this preflight check succeeds, then the controller provides encryption keys for early encryption of personal data.

Once the collection and transfer phases are complete, the resulting data is imported into a database hosted at the organization. Certain data that is not retrieved by the agent, e.g., from a centralized or cloud-based issue tracker, is retrieved by another instance of the gatherer running on a virtual machine (VM) or Jenkins [IV] instance. The controller sends a *trigger* notification to this instance to do so. Figure 2.3 displays the components and their connections in this setup.

The combined update of the data means that the organization's database is usable for local data analysis, prediction algorithms and visualization. The data is also regularly exported and uploaded to a central Grip on Software database, such that analysis also takes place using combined data from multiple organizations in one place.

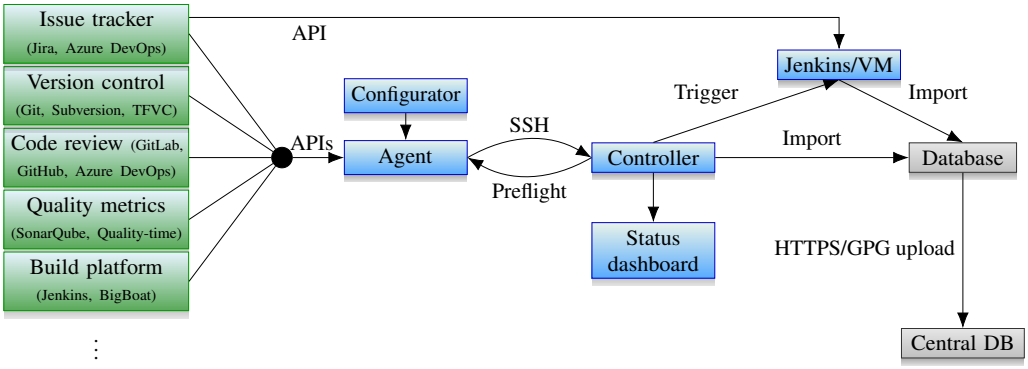


Figure 2.3: Overview of the data acquisition components and their interactions within the Grip on Software pipeline. The green blocks represent possible systems in the existing development ecosystem. Blue blocks are components of the pipeline. Gray blocks are components that are described in later chapters. Arrows indicate the direction that data travels in.

In other ecosystems, a distributed setup would be superfluous. When all teams work on a number of components of the same project, hosted on one platform—such as at Wigo4it—there is only a need for a single access point. The collected data is then either imported into a local database or uploaded directly to the central database.

For many of the specific systems that we use as a source of software development process data, we build modules that provide API connections to them. This makes it easier to update the pipeline components when changes to the APIs are not backward-compatible and unfortunately require changes within the pipeline code as well. For example, in Jenkins or older versions of Azure DevOps—previously known as Team Foundation Server (TFS) and Visual Studio Team System (VSTS) [VIII]—we use specific connections to remain compatible with existing deployments, such as the older version control component of TFS known as TFVC. We build around these modules to define our own models for the data types, which describe the data source systems on a higher level. This domain-based approach allows separating the authentication configuration from the connection itself, enhancing the modularization of each step.

The pipeline mostly operates automatically based on schedules. The agent frequently attempts to retrieve changes to the source data, as is common in a fast-paced SCRUM process. The configurator and the status dashboard are a means to adjust how the automated runs operate, by providing options to change how the agent connects to the other systems or to reschedule the agent's execution if earlier operations failed due to problems that have since been amended. The agent will then collect data as of the most recent finished run. The configurator provides a systematic access to this part of the pipeline. Figure 2.4 displays how the configurator looks like to the user—most likely researchers who monitor the pipeline's progress, but possibly including team members involved in the project that the agent collects data from.

In this interface, several systems are configurable, which includes filling in URLs, mappings of identifiers and authentication credentials. If code repositories or build toolchains are split up across multiple instances, additional systems can be configured as well. For security reasons, after the configuration is updated, the credentials in this interface are instead indicated with a placeholder, which keeps the old credentials if it is unchanged. The credentials are still modifiable

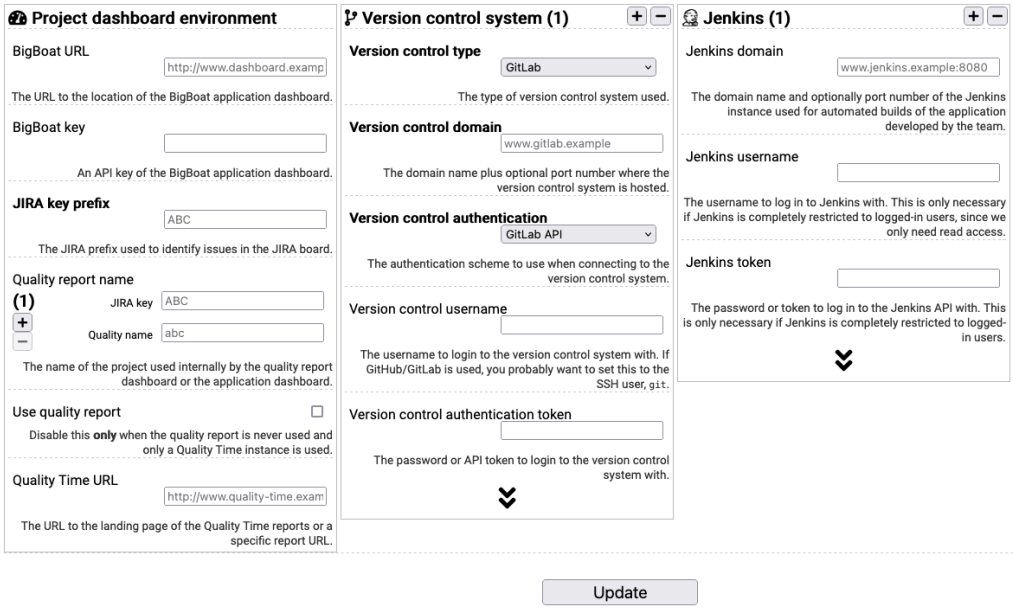


Figure 2.4: Screenshot of the configurator, with options to set up connections to various systems acting as a data source.

at a later moment. The agent always uses the most recently updated configuration to connect to the defined systems. The agent also selects more relevant systems from the quality control system and the tracked code repositories.

Personal data is collected by the agent and included in the artifacts in an encrypted manner. The encryption keys are used to perform a one-way encoding. This means that the data—e.g., a name or email address—can only be inspected if the attacker already has the original data as well as the keys. Still, this form of data protection is considered to be a pseudonymization rather than true anonymization, since it makes use of the original data for the encoding. In general, personal data is not used during the analysis since it is not relevant for the overall SCRUM process. We sometimes compare the resulting hashes against encrypted personal data from other systems or from an earlier run. We use the encrypted names of developers to avoid duplicate entries of the same person across multiple systems. Another instance of pseudonymous data use is when we extract the domain name of an email address to check if people are employed by the organization or by the client, for example. During exports to the centralized database, the encryption keys are not included in the upload and remain at the organization. The data is further secured through HTTPS and GPG encryption [IX], avoiding interception of the data during the transfer. The analysis only uses pseudonymous data.

2.3.2 Further pipeline steps

The data acquisition components form only the first phase of a pipeline that is aimed toward providing insight into patterns of software development processes. The proper functioning of the

pipeline depends on more phases that consolidate the data of software development projects in a (local or centralized) database and that extract relevant data points such as features from this database. These are then used within prediction and estimation models as well as in information visualization. These correspond to the remaining components that are colored gray in Figure 2.2.

Each phase has software in order to transfer data to the next phase and maintain the proper functioning of the pipeline. The following components are part of the pipeline of the Grip on Software research project. The code repositories of these components have all been made open source. The items in the list refer to the repositories whose references are found in Appendix A. We also indicate the programming languages and the kinds of tests for the component. The list is grouped by the phases of the pipeline, which are described more extensively in various chapters of this thesis.

Data acquisition (Section 2.3.1):

- `data-gathering` [a]: A collection agent for data from software development processes. The schedule-based gatherer has deployment options for different data sources and contexts. Module-based Python 3 code with type checking. Integration test runs are possible on Jenkins.
- `agent-config` [b]: A web-based configurator for the GROS data gathering agent. Intended to be deployed with the agent in a Docker Compose system. JavaScript-based, relatively well tested and formatted using routes, templating and some utility functions.
- `status-dashboard` [c]: A web-based dashboard providing an overview of data collection agents and their status, with scheduling options. Written in Python 3 with typing.

Data import and export (Chapter 3):

- `monetdb-import` [d]: A schema-based database importer. Works with MonetDB [38]. Java-based package format, schemas and update files plus some Python 3 and Bash code for database administration as well as validation. Contains some unit tests for utilities.
- `monetdb-dumper` [e]: A Java-based database exporter for MonetDB. The administrative interface for importing and exporting dumps tie in with this package.
- `export-exchange` [f]: A module that handles export of the database and secure upload to a centralized instance. Written in Python 3 with typing.
- `upload` [g]: A web server that receives secure uploads of database exports to load into a centralized database. Written in Python 3.

Analysis and pattern recognition (Chapter 4):

- `data-analysis` [h]: Analysis interface for aggregating data with source traceback from a filled MonetDB database. R-based integration [X]. Configurable, dynamic queries that adapt to organizational differences.
- `prediction` [i]: Models to predict, classify, analyze and estimate features and labels regarding SCRUM data. Python 3 with TensorFlow [XI] and Keras.

Visualizations (Chapter 5):

- `visualization-site` [j]: Landing dashboard site for different/related visualizations. Configuration backbone. Contains integration tests for deployment within a virtualized network based on Docker compose [XII] using several web servers that could act as proxies, testing the main site and the specific visualizations.
- `visualization-ui` [k]: JavaScript modules that are reused in various visualizations. Includes unit tests.
- `sprint-report` [l]: Dynamic generator of various comparative visualizations inter- and intra-project per-sprint.
- `prediction-site` [m]: Human-readable output of predictions.
- `timeline` [n]: Various temporal data from the software development process.
- `leaderboard` [o]: Project-level statistics.
- `collaboration-graph` [p]: Relations between development team members and projects that they worked on.
- `process-flow` [q]: Flowchart of the status progress of user stories with volume and time metrics.
- `heatmap` [r]: Software development code commit activity.
- `bigboat-status` [s]: System reliability graphs for the BigBoat development platform.
- `backlog-burndown` [t]: Effort burndown chart for product backlogs.
- `backlog-progression` [u]: Progression inspection chart for product backlogs.
- `backlog-relationship` [v]: Issue relationship chart for product backlogs.

Supplemental components:

- `deployer` [w]: A web application to deploy repositories to a managed machine, with configuration to use a quality gate, run commands, manage secret files and restart services. Written in Python 3 with typing.
- `server-framework` [x]: A Python 3 module to develop authenticated web servers.
- `jenkins-cleanup` [y]: Maintenance scripts to remove stale Docker, Jenkins and SonarQube data during pipeline development. Written in Python 3 with typing and Bash.
- `coverage-collector` [z]: A service that collects coverage information in JavaScript during browser tests. Used in combination with the visualization site integration/unit tests.

2.4 Technical considerations

We design our data pipeline with the purpose of gathering knowledge regarding SCRUM software development processes. We deploy the pipeline within ecosystems with their own requirements and existing behaviors. For these reasons, there are several considerations when deciding how to design, develop, document and deploy the pipeline components.

Aside from our objective of acquiring and understanding data, we consider the pipeline itself to be a subject for study. Based on the concepts that are realized in a distributed system, we discuss how we generalize the implementation to other development ecosystems. Additionally, we demonstrate how state-of-the-art algorithms augment the components.

In this section, we discuss some of these concepts and characteristics that we find relevant for further consideration.

2.4.1 Generalizability

In principle, the components that are used for collecting, disseminating and presenting data from different sources are able to be placed in different development ecosystems. The individual components work on build platforms such as Jenkins [IV], Docker-based platforms, virtual machines or servers, with a limited number of base dependencies installed. Due to this, the code has run at two different software development organizations as well as at a university, each with different requirements for collecting, publishing, sending and receiving data between internal and external networks.

The sources where we collect data are specific to our research scope. This includes various project management systems, version control systems, quality review systems, build platforms and organizational resources. We support TFS/VSTS/Azure DevOps Server [VIII], Jira [I], GitLab [III], GitHub [XIII], Subversion [XIV], SonarQube [V], Quality-time [VI], BigBoat [VII], Jenkins [IV], LDAP [39] and TOPdesk [XV].

As these project management products get updated or replaced in the future, the data gathering components continue to use APIs that become outdated and eventually become unable to collect new data. This is the main reason that it would require continuous and adaptive maintenance to keep the software relevant.

Certain pipeline components are harder to configure in order to work in a specific ecosystem. Setting up a MonetDB database with different parts written in Python 3, Java and R is cumbersome if there is no proper platform. At each point that a portion of pipeline is deployed to a specific environment, certain decisions need to be made in order to assure that the pipeline is able to work there. Some components provide a configurator, with intuitive means of selecting and entering the details of the connections that they make. The accompanying documentation indicates additional configuration options.

In the end, the pipeline was set up with a specific purpose in mind: collecting data regarding multiple development projects together in order to find relevant patterns. Some components will not be suited for different purposes, given that the data gathering is focused on specific systems. On some points, scalability and performance are kept in mind. The pipeline however does not work out of the box in a completely new ecosystem without prior knowledge. Thus the components are provided “as is”, meaning that some adjustments may be required. This is inherent to any open source system that seeks generalization; other interested parties can contribute to make it fit in a specific ecosystem.

2.4.2 Continuous integration

The components primarily use Jenkins [IV] as a continuous integration (CI) system, which is geared towards performing tests, building the software and collecting data automatically. Jenkins provides an interface to view and manage these steps. Jobs are started to execute tasks based on schedules and code changes.

Our code repositories define their own Declarative Pipeline, a syntax which indicates the requirements and stages to build, test, analyze and deploy the code. The usual purpose of a CI system is to allow rapid merging of development code into a stable version supported by automated review, which is still one of the features used for our pipeline components. Builds are regularly scheduled so that problems with changes and dependencies are found early. Several components have unit tests and integration tests. Code analysis is performed using a SonarQube scanner [V] in order to track the pipeline component's code quality, which also uses information from test results and coverage.

Jenkins usually works with a controller node—running on the server on which the CI system is installed—and agent-based executor nodes. Each node handles one or more concurrent tasks. This way, builds of multiple code repositories take place on separate machines, which are synchronized with regards to the platform dependencies. Additionally, specialistic nodes are possible, for example to perform machine learning tasks on a server equipped with a graphics processing unit (GPU). Portions of a build take place on different nodes, which are aggregated before publishing the results on another server.

2.4.3 Documentation

All of the pipeline components include documentation regarding deployment and configuration. The data acquisition component is documented on function, class and module level. The database component is documented through human-readable versions of the schema as well as individual import modules with documentation on parameters. The prediction and information visualization components also come with documentation for modules and methods. Data formats exchanged between all components are documented with JSON specifications[‡], including API definitions[§], which allows validation of instances of data artifacts against the specified schema as well as meta-validation.

Each code repository has some form of documentation. All components come with instructions that help with building, configuration and installation, usually with several example options using Docker, Jenkins or a standalone form. The documentation often describes how to start up the component or how to run individual programs, with details on command-line arguments through help systems. Web-based systems such as the configurator and the visualizations describe in detail what certain controls and options do, using tooltips, for example.

Aside from the standalone documentation, the implementations of scientific analysis systems are also documented through publications, including this thesis. Furthermore, we write comments for code lines, methods, modules and components, considering reproducibility and maintainability of the pipeline at each level.

[‡] https://gros.liacs.nl/schema/
[§] https://gros.liacs.nl/swagger/

2.4.4 Novelty

As mentioned before, the components of the pipeline beyond data acquisition are described in more detail in their own chapters. But some discussion does not fit the topic at hand when the main focus is database modeling, pattern recognition or information visualization, for example because this would make the narrative there too verbose and technical. Still, we find it relevant to describe some technical details, such as a new implementations of an algorithm.

For the MonetDB database, we developed some administrative programs [d] as well as an importer which is described in more detail in Section 3.4. The importer plays a crucial role when the data acquisition components provide new data that should be stored in the Grip on Software database. One potential issue that we resolved is compatibility with importing older versions of JSON artifacts. Sometimes, these files are kept as importable source objects after a software development project is completed and the original systems turned off. Aside from backward compatibility, forward compatibility could also be relevant if different versions of agents and importers were in use. Although the agents are able to check for updates—with the configurator reporting this—and a new version of the Docker image is easily deployed, the versions could still become desynchronized. The artifacts and importer were designed in such a way that later additions do not hinder the import process, while missing fields due to agents that were not updated are also handled properly. Often, these fields are filled in during a later import as well.

We frequently update the database itself to support a new schema through a program that validates the state of the model. Based on structured information of an update, the program checks if certain database columns or keys exist and have the proper types. Based on this detection system, schema updates will then take place or are skipped.

In order to extract the features that describe characteristics of SCRUM sprints, as mentioned further in Section 4.4.1, we developed an R toolkit program [h] that performs queries upon the database. The queries are based on templates which contain variables and other structures, enabling R code to fill in proper field names and conditions. This hybrid system within MonetDB properly selects data and filters out noisy data, such as the practice of giving a story a very high number of points to avoid working on it in a recent sprint. A structured list tracks the available definitions for these fields and conditions for reuse in other places. Based on the ecosystem, this enables some queries to select the same kind of features from different sources, such as Jira and TFS/VSTS/Azure DevOps. The queries themselves are also tracked, including metadata to describe the source through a specific, human-readable URL with the same data, as well as aggregation options, labels and unit formatting preferences. These come in handy for the display of such features in the information visualizations described in Chapter 5.

As for the pattern recognition methods, we make use of TensorFlow [XI] and related technologies to train, evaluate and predict outcomes of samples within data sets, which are made up from the features extracted for the sprints from the filled database. We implement several models to do so [i]. One novelty here is the use of TensorFlow to implement analogy-based effort estimation (ABE). This algorithm, described in Section 4.4.3, does not necessarily stem from machine learning, but still includes steps that are well-suited to be performed in batch, such as finding similar sprints based on a distance measure to neighboring samples. Such steps can then be performed upon a GPU using vector-based instructions generated through TensorFlow. This allows for further use of specialized hardware within our pipeline, leading to performance improvements. This lets us bring the estimation and prediction results to the SCRUM team members more quickly, demonstrating the advantages and usefulness of the pipeline's construction.

2.5 Results

In order to determine whether the pipeline components can be set up in a stable manner without influencing an existing software development ecosystem, we tested the performance of the components by enabling them for 22 different development projects. The development ecosystem is described in detail for the agent-based data acquisition system from Section 2.3.1: each project has its own virtualized network with a BigBoat [VII] deployment platform, including all services for development such as version control and code review, usually GitLab [III]. Code quality is monitored through SonarQube [V] and Quality-time [VI].

We deploy the agent and configurator on this platform using Docker Compose [XII]. The controller runs on a separate VM in an isolated network, along with the status dashboard. A Jenkins continuous integration system [IV] collects data for the projects from the Jira issue tracker [I] based upon a trigger from the controller.

We tracked the CPU processing load, RAM memory usage and disk storage space of the BigBoat platforms. Overall, the agent did not cause a noticeable impact to the normal usage trend of the platform. The performance did not differ between runs that took place hourly and every 15 minutes. More frequent schedules could be possible, but we do not consider this to be necessary, since most other components within the pipeline were not aimed at live data presentation. Additionally, if the complete collection run—both the data collection by the agent and the database import performed at the controller—would take longer than the frequency, a concurrent run is not allowed to be started for the same agent in order to prevent conflicts. This avoids an avalanche of jobs working on the same task of acquiring a large bulk of new data, for example when a project with a long lifespan is newly included in our research.

One problem that surfaced was the use of large source code repositories by some teams. The data collection would retrieve new versions of code commits, but for some version control systems this still caused some overhead, in particular Subversion. Figure 2.5 shows performance downgrades that we detected in this situation. We included some mitigations to reduce the time spent with file differences and to limit memory usage. This helped ensure that the platform remains available for its primary purpose.

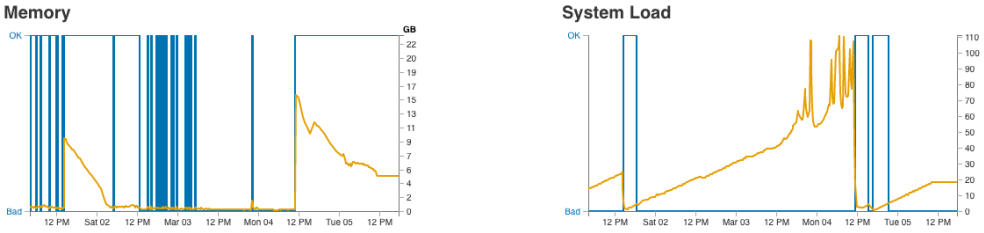


Figure 2.5: Graphs displaying the available memory (in gigabytes) and CPU processing load over time for a development project's platform when problems arise with collecting Subversion repositories. The blue line indicates when the platform signals a decrease in performance, while the yellow line shows the actual value of the metric. After this event, problems with load were mitigated.

2.6 Discussion

This chapter introduces the components of the pipeline for the Grip on Software research, aimed at understanding situations arising from the application of the SCRUM software development method. In particular, we detail the design and construction of the data acquisition components that constitute the first phase of our pipeline, aimed at consolidating various data sources for pattern recognition and information visualization. We demonstrate how we apply concepts from distributed computing and agent-based networking in order to overcome organizational differences in ecosystem structure. We document our objectives and mention the components of the entire pipeline. The other phases of the pipeline are discussed based on their technical aspects, such as source code and documentation. From these components, we highlight some non-functional requirements and new work that they provide.

We conclude that our data acquisition pipeline works well at the two organizations where it has been applied. The early encryption and audit logging provisions within the components help with ensuring secure communication and storage. We are able to uniquely identify entities across systems and keep track of where data came from, as a means of data provenance, making it easier to reproduce results of later stages of the pipeline.

Another non-functional requirement is the performance of the pipeline. We reuse the logging, scheduling and usage statistics to measure and improve the efficiency of the processes. Based on our results from experiments with the setup in the distributed networks, we determine that it is possible to follow updates close to real time, insofar that this is helpful for the machine learning models and reports, which are similarly updated frequently. We are able to resolve issues with high load through code simplification and usage limits without impacting our goals.

The components of the pipeline were developed for the purpose of collecting and analyzing data from software development processes. We demonstrate the application of the pipeline in multiple ecosystems that software development organizations have in practice. The code repositories of the components are open source, allowing reusability.

In the end, the pipeline is the instrument in order to substantiate our main research objective of finding relevant patterns within SCRUM software development processes. It is important to make this instrument work properly from the ground up. It would also be a waste if it would simply be shelved after it has been used just once.

Chapter 3

Database construction

GROS DB: Designing and deploying a database model using extensible and flexible query templates as part of a research-oriented data pipeline architecture

Portions of this chapter are also published in the following article:

- Leon Helwerda et al. “Query compilation for feature extraction in MonetDB”, 2024. Pending submission.

Abstract of Chapter 3

Introduction: Software development teams use several systems that form a development ecosystem to keep track of data related to their projects. We use a data acquisition pipeline to collect records from these systems for research into patterns and outcomes of the process. We store this information in a consolidated MonetDB database with column-based storage. This setup allows us to perform extensible queries.

Research questions: How can we reliably collect data regarding SCRUM software development practices and consolidate the resulting artifacts inside a central database that constantly grows and allows adaptable queries?

Modeling: We define a data model which contains discrete portions corresponding to systems that provide the data, such as issue trackers, version control systems and quality control dashboards. The model consists of entities and relationships that describe states, events and dependencies. We enhance the model with links between different portions, which are not available in the separate systems.

Architecture: We enhance the data selection through the introduction of a query template compiler. Data from different systems and organizations is properly selected and combined into a single data set by adjusting the query parameters for the appropriate context on a high level. We implement more functionality in the database component to handle backups and exports to a central instance of the pipeline.

Experiments: Our initial selection of MonetDB is put to the test. We establish a representative workload of six query templates, each with a refined version and an older version without the changes. We measure the performance of the MonetDB database by comparing two scenarios, one where the queries have been seen by the database several times and one where the database has not created auxiliary internal structures based on the queries. Each version of each query is run in both scenarios ten times. The results show that both our refinement of the query templates and the optimized scenario for MonetDB improve the run times of the queries, justifying the selection for MonetDB and making frequent analysis of the data set during SCRUM sprints feasible.

3.1 Introduction

In order to consistently, reproducibly and frequently perform analytical studies regarding patterns and outcomes of SCRUM software development processes, we design a consolidated location for the data acquired from several systems used by multiple software development projects. As part of the pipeline that we introduced in Chapter 2, we identify a need for a highly-available storage location that allows for a steady stream of additions in order to reflect the situation as experienced by the development teams. Furthermore, the structures and events described by the data should be available for data analysis using various attributes in order to output different reports for involved people.

There is a necessity for a model which describes the interactions between the different artifacts during SCRUM sprints, including product backlogs, code repositories, quality reports and release builds. Often, developers are able to navigate between these items in their current state, but they are hindered by design flaws of the systems when tracking down what they looked like at a specific date, during a sprint from months ago. It is complicated to understand what a metric refers to when it is stored at a separate system without a proper reference point. Thus, our data storage for Grip on Software should have a temporal aspect, where time is a first-class data type with many available functions. It should further allow filters or aggregate operations to determine which factors define a sprint most pertinently.

We consider existing data stores which already implement features that we desire. A *relational database management system* (RDBMS) offers a table-based storage with rows and columns, where some entries refer to other columns in another table. In addition, the RDBMS provides operations in order to add, remove, adjust and search within the relations, allowing manipulation as well as selection. These operations support combining tables in a *query*, with additional arithmetic or programming operators to alter which data points are added, adjusted or retrieved.

This sort of system enables much flexibility towards modeling complex relationships of data structures that reflect dynamics of software development ecosystems. Both the mapping from artifacts to table-based storage as well as the operations involved in manipulating the data are part of the *modeling problem*, which differs for each subject area.

Another side of using a database system is the management and architecture of technical components in the pipeline. This technical aspect largely consists of ensuring that the database has enough hardware resources in order to perform its operations, including establishing connectivity with other components. This further enables exports, backups, migrations and upgrades to take place. These maintenance tasks should not affect the typical performance of the pipeline. The database needs to remain consistent and resilient, with no loss of data due to downtime, for example.

When building a database that should function at more than one organization, there are additional concepts surrounding the *architecture problem*. We consider the integration of the database component in the pipeline and the overall development ecosystem that exists at an organization, the centralization of data of multiple organizations using a separate research pipeline, the administration of privacy-sensitive data and the optimization of queries performed on the database. Meanwhile, the solution needs to support a complex data model based on multiple specialized development platforms. It is important that the solutions for these considerations do not affect the model's restrictions on the outcome of the query operations, among other things.

We summarize the pertinent points of a database system that we find useful for our purposes by proposing a number of objectives for the usefulness of the technical database component:

- We should be able to design a single data model that addresses several complex SCRUM ecosystems, allowing us to easily recognize the original entities and relationships, but also introduce new links between data from previously separate systems.
- The query language allows us to utilize the data model to retrieve relevant data, spread across tables for entities and relationships, with many existing functions for grouping and aggregation as well as opportunities for extending the queries, for example through user-defined functions (UDFs).
- The RDBMS should be efficient enough to perform under high load of parallel updates from independent data acquisition agents, while allowing frequent analysis with complex queries.
- The database component should function as a part of a larger pipeline which operates in different environments, without hindering the existing use of the software development ecosystem.
- Typical guarantees by database systems, such as atomicity of transactions and resilience to crashes, are considered a prerequisite to an operational database, along with management interfaces.

These desirable qualities of a database bring us back to the questions regarding the need and usability of such a system in the overall Grip on Software pipeline, aimed at collecting and understanding metrics regarding SCRUM software development processes. We therefore consider the addition of the following two sub-questions to the research question, mentioned before in Section 2.1, regarding the data acquisition and consolidation pipeline:

RQ1 How can we reliably collect data regarding SCRUM software development practices and consolidate the resulting artifacts inside a central database that constantly grows and allows adaptable queries?

RQ1c How can we model the relationships between different data artifacts acquired from dynamic systems regarding SCRUM software development in order to properly deduce information about their state during different sprints?

RQ1d Which technical challenges are relevant when deploying a database component as part of a pipeline for multiple organizational ecosystems?

The remainder of this chapter focuses on this two-sided problem of database design and architecture. First, we consider existing modeling structures and management systems in Section 3.2. In Section 3.3, we introduce the Grip on Software database (GROS DB), including the data model in Section 3.3.1. The entities and relationships are shown step-by-step, based on subdivisions of the model. The relations between different originating systems are considered in Section 3.3.2. For the architecture problem, we describe how we administer, optimize, engineer and adjust the behavior of the database system in Section 3.4, including a novel query language overloading extension. Section 3.5 presents some experiments regarding optimization and performance of resource usage. Finally, we discuss the entirety of the Grip on Software database as a component of the data acquisition pipeline in Section 3.6.

3.2 Relevant work

Databases are used in a variety of applications and come in many forms. In the scientific field, an RDBMS is often used to persistently store data using a known model. This way, further analysis profits from efficient lookups of the data.

An area of interest within database architectures is the use of column-based storage. This technique helps with efficiently retrieving attributes of a large number of entities stored in tables. Such queries occur regularly when performing analytical research with large amounts of data.

One column-based database store of interest is MonetDB [38], which is used in various research contexts. We find that there are applications of MonetDB for machine learning [40] and statistical analysis [41]. Novel functionalities to database stores allow streaming updates for processing live data, such as in software quality analytics [42].

The architecture underlying MonetDB integrates with programming languages in order to create UDFs [43]. Support for languages such as Python and R has led to seamless integration with more software packages used in data science [44], such as reuse of existing columnar vector formats in libraries like TensorFlow [45].

Some methods for improving the efficiency of in-database operations avoid copying data between memory locations. Such zero-copy integration has shown to work well for statistical problems [46]. Extensions with other systems to provide better query filters using multi-dimensional indexes also exhibit performance improvement [47].

With all these improvements on integration, extensibility and memory usage, it is often another challenge to provide experimental results on database performance. It is relevant to test similar, reproducible configurations. Further, there should be additional tests with baseline situations where the database system has not had the chance use caches and to process the type of query workload [48]. For specific applications, providing a reasonable representation of the query workload serves as a relevant, localized benchmark [49].

We also look into the use of database systems in the context of analysis of SCRUM software development data. We find that there exist models that look into integration of data from multiple sources, such as quality metrics [50]. Similar approaches with transformations of originating data lead to a model which allows generating dashboards for monitoring performance for SCRUM teams [51]. Sometimes, the goal of data-driven software development is reached through the use of an extensive ontology describing the aspects of SCRUM [52].

3.3 Method

We introduce the technical component which handles database storage, imports, exports and management. This component is a continuation of the data acquisition pipeline introduced in Section 2.3. This pipeline is deployed in an existing software development ecosystem at an organization that is involved in the Grip on Software research. An outline of the pipeline is shown in Figure 3.1, with the components of interest for this chapter highlighted with red and blue colors. Multiple instances of this pipeline reside independently at different organizations. Another version of the database plus the remaining machine learning and information visualization components are placed in a central location. Other pipelines send encrypted backups of the database contents—without any readable personal information—to this central database, so that multiple organizations are combined into one data set for further generalized analysis.

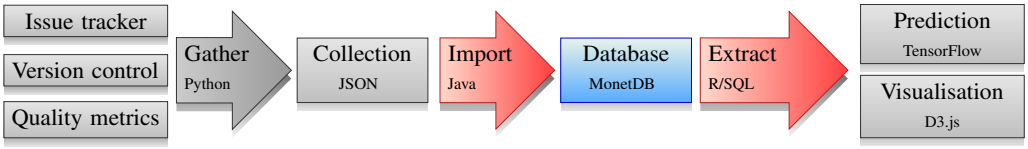


Figure 3.1: High-level overview of the Grip on Software pipeline, with the database system shown as a blue box and relevant technical components as red arrows.

The database uses MonetDB for storage and low-level administration [38]. MonetDB focuses on storage of large-scale analytical information. In contrast to many database management systems that existed at the time of selection, MonetDB is a column-based RDBMS. This means that in this database, all data of a column is stored consecutively, instead of storing row data together. Efficient retrieval of attributes is a main factor in initially selecting MonetDB. Since MonetDB's introduction, other databases such as MariaDB introduced options for column-based storage.

MonetDB is similar to other relational database management systems in that it supports the Structured Query Language (SQL). This query language implements the operations needed for retrieving and manipulating data provided by the database. We consider three main types of invocable queries: (a) data definition statements using `CREATE` as the initial keyword, (b) data manipulation using `INSERT` or `UPDATE`, and (c) data access with a `SELECT` statement. Thus, we first *define* the tables representing entities and relationships, with columns for attributes of different numerical, textual and temporal data types. Then, during the operation of the data pipeline, we *manipulate* the database by filling it with fresh data and altering old data to reflect the most recent situation. We finally *access* the data, selecting subsets for further analysis using combinations of tables through `JOIN` operations and other expressions that filter, order, aggregate, partition and combine data. Such a selection is performed in one go, without having to execute several queries. These features are provided by MonetDB, extending the common SQL language.

The database connection protocols that we use for importing data also support prepared statements in SQL. This allows another system to indicate a query that is to be performed multiple times with different parameters. This reduces the overhead of sending the statement to the database, compiling and optimizing it every single time it is used. For example, we use multiple prepared statements that verify if data did not already exist or needs an update. Then, we respectively insert new rows and perform batched updates for fresh instances of entities.

In addition to query language support, an important feature of MonetDB is integration with other programming languages. The data analysis component of the pipeline connects to the database in order to perform queries. Through UDF integration, a language like R [X] augments the queries with operations specific for our purposes, reducing the need for post-processing. At the moment of implementation, we chose to use R. Since then, MonetDB extended support to Python UDFs as well [53]. Nevertheless, they also continued integration with R [54].

As mentioned in Section 3.2, MonetDB offers more options for extensions as well as in-depth performance optimizations. However, by default, the column-based storage already has compression and dictionary encoding for frequently stored textual values with specified or arbitrary lengths. In this way, the lookup speed and reduction of memory usage are balanced, enabling the modeling of large data sets. The SQL queries themselves are the input of an internal optimizer pipeline, which transforms it to a MonetDB-specific assembly language supported by a relational algebra that works on the internal table structure [55].

3.3.1 Data model

We design a model for GROS DB, a database containing events, artifacts and other relevant elements from SCRUM software development processes. This model is based on the representation of the same entities and relationships as in the systems that we have extracted them from.

An overview of the most relevant entities has been shown during the introduction of the pipeline, in Table 2.1. There, we described from which type of system they originate. This includes issue trackers, version control systems, associated code review, quality control dashboards and build platforms. The aforementioned overview did not mention entities that only exist in relation to another or provide context to a development project, e.g., a software component that an issue applies to. In our survey, we found that some systems contain similar data, such as developers having accounts on multiple services. These systems often refer to each other with an indirect, temporal association: a code commit takes place during a certain sprint.

Software development organizations that use Agile approaches do not always have the same development ecosystem; the same applies to their development teams. As an example, we take the two organizations of our study. At ICTU, there is a frequent use of Jira [I] as an issue tracker, GitLab [III] for version control and code review, SonarQube [V] plus self-made quality dashboards alongside the BigBoat [VII] and Jenkins [IV] build platforms. For Wigo4it, the workflow of refining and implementing stories takes place on TFS/VSTS/Azure DevOps [VIII], with SonarQube for quality control. We summarize the distribution of systems in the two organizations in Table 3.1. Other potential systems in use by such organizations are, e.g., GitHub [XIII] for open source projects or TOPdesk [XV] for internal project asset management.

SYSTEM	ICTU	WIGO4IT
Issue tracker	Jira	Azure DevOps
Version control	Git (some Subversion and TFS)	TFS (Git/TFVC)
Code review	GitLab (most projects)	TFS/VSTS/Azure DevOps
Quality control	Quality time, SonarQube	SonarQube
Build platform	BigBoat, Jenkins	Azure Server
Personnel records	LDAP, Seat counts	Azure DevOps

Table 3.1: Overview of systems in software development ecosystems at two organizations.

In order to model the entities and their interactions in these diverse ecosystems, we first design a rudimentary entity–relationship (ER) diagram. In Figure 3.2, the entities are shown in a fundamental model, excluding attributes, but with most relationships. In order to obtain some "shortcut" relationships, we follow a chain of them, such as an issue belonging to a certain project via the sprint entity. Specific entities, including those based on the issue tracking systems of Jira and Azure DevOps/VSTS/TFS, as well as entities from various types of code repositories, are considered implementations of a class diagram relationship; here, they are condensed into a generic form.

The focus of this section is to model the entities in such a way that we are able to extract a database schema, which defines how the tables are created. One relevant matter is the nature of the relationships between entities. In the diagram, we indicate the type of mapping between entities using *cardinalities* as labels of the relationship's lines. A *one-to-one* relationship, indicated by two 1's on either side, is similar to a bijective function, whereas a *one-to-many* relationship,

where one side is not limited and includes an asterisk $\star$ in its cardinality indicator, is like a function with no specific properties, its domain originating from the entity on the side with the 1. A *many-to-many* relationship is best regarded as a multivalued function to either side. Some cardinalities are *optional*, when a side is shown with a $0/1$ or simply an asterisk as indicator. Then, some instances of the entity on an optional side may not participate in a relationship. Table 3.2 provides an overview of the cardinalities by means of examples that are visible in Figure 3.2.

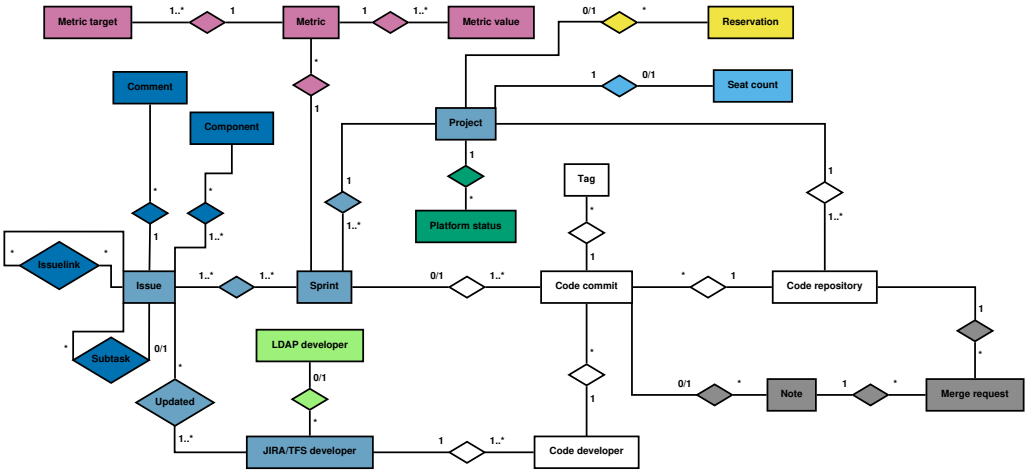


Figure 3.2: An entity-relationship diagram of the main entities and primary relations that are stored in the database. Colors used in this diagram and others indicate the sources of groups of entities: ■ dark blue are retrieved from Jira, ■ grayish blue from either Jira or TFS/VSTS/Azure DevOps (■ light gray in other diagrams), □ white from version control systems, ■ gray from code review systems, ■ pink from quality dashboards, ■ green from build platforms, ■ light green from LDAP, ■ yellow from TOPdesk and ■ light blue from spreadsheets. Not shown here are entities for source tracking and database consistency, colored ■ purple and ■ orange in other diagrams, respectively.

CARDINALITY	EXAMPLE	TARGET	REFERENCE
One-to-one ($1 \leftrightarrow 1$)	Encryption of a project	`project`	■ `project_salt`
Optional one-to-one ($1 \leftrightarrow 0/1$)	Seat count of a project	`project`	■ `seats`
One-to-many ($1 \leftrightarrow 1..\star$)	Sprints during a project	`project`	■ `sprint`
Optional one-to-many ($1 \leftrightarrow \star$)	Comments on an issue	`sprint`	■ `comment`
One-to-many optional ($0/1 \leftrightarrow 1..\star$)	Commits during a sprint	`sprint`	□ `commits`
One-to-many all-optional ($0/1 \leftrightarrow \star$)	Reservations of projects	`project`	■ `reservation`
Many-to-many ($1..\star \leftrightarrow 1..\star$)	Issues in sprints	`sprint`	■ `issue`
Many-to-many optional ($1..\star \leftrightarrow \star$)	Issues updated by users	`developer`	■ `issue`
Many-to-many all-optional ($\star \leftrightarrow \star$)	Issues linked to issues	`issue` (2×)	■ `issuelink`

Table 3.2: Cardinalities of relationships that are noticeable in the ER diagrams of the data model. For each cardinality, an example is shown with the entity table that is the target of the reference and the table where we store the reference attribute.

As a typical example, we introduce two additional data sources—meant to validate data from our primary sources—for meeting reservations and numbers of full-time equivalents (FTEs), including seat counts. These entities show two different cardinalities in their relationship: a project has any number of reservations—and a meeting reservation is related to at most one project—while a singular seat count is related to a unique project, if known. These are respectively a one-to-many all-optional and an optional one-to-one relationship. A many-to-many all-optional relationship is found in issues possibly linking to each other; here, domain and codomain are equal, self-links excluded.

The complex issue entity has other many-to-many relationships: the issue could be worked on during different sprints, by various developers and for certain software components. Entities having multiple versions exhibit such complicated interactions. This is also dependent on the level of detail that the originating system provides in the earlier versions of the entity. Similarly, attributes for an issue from Jira differ from work items in TFS/VSTS/Azure DevOps.

All these entities with attributes and relationships have a specific role. Despite the complexity, it is important to model the GROS DB in such a way that it reflects reality, describing the situation as it is in other systems as genuinely as possible. If, early on, we would discard attributes and elaborate relationships, then the analysis that we perform in later stages is more cumbersome. We find it easier to leave out sensitive data—such as personal information and project names—during an export, so shrinking to the necessary fields afterward is a better trade-off.

In Figure 3.3, the entities with all attributes and relationships are shown[¶]. The colors of tables correspond to the grouping in the smaller ER diagram. Some tables now have specialized implementations for their source systems. Attributes in this diagram have key indicators for *primary keys* that uniquely indicate the entity. Yellow icons are local while red icons indicate a *reference* to another table. Non-key attributes have diamond icons, with only an outline if the value is optional. A line indicates a relationship; the line is dashed if no primary key is involved in the references between the two tables. The origin of the relationship—where the reference attribute is stored—is indicated with a triangle at the endpoint. The endpoint is a circle if the reference is a temporal data type.

We introduce relationships which involve multiple attributes in order to properly reference the tables. When primary keys are concerned, the relationship is usually one-to-one or one-to-many. In this visualization, we intentionally left out some lines, with only the endpoints remaining in the diagram. This either means that this relationship is a shortcut or that multiple attributes refer to the same table. This reduces the complexity of the representation, which would otherwise have many more crossing lines. The full diagram is obtainable from the database administration pipeline component [d] using MySQL Workbench [XVI], allowing more inspection and reuse in other database systems.

These additional references make it easier for us to design queries that combine data from multiple entities, while preserving database normalization at the necessary level to fulfill the details of the entities. For example, we could assume that a story is always worked on in the project that a sprint belongs to. However, there may be situational differences in which the story was moved between projects. This discrepancy could not be modeled without tracking the project in each version of the issue. Thus, we track this data through more direct references.

We address specific parts of the larger, encompassing diagram in more detail in the following subsections. In particular, we examine the entities and relationships from Jira, version control systems with their associated code review, TFS/VSTS/Azure DevOps and quality control systems.

[¶]The diagram is also available at `https://gros.liacs.nl/database-model.pdf`

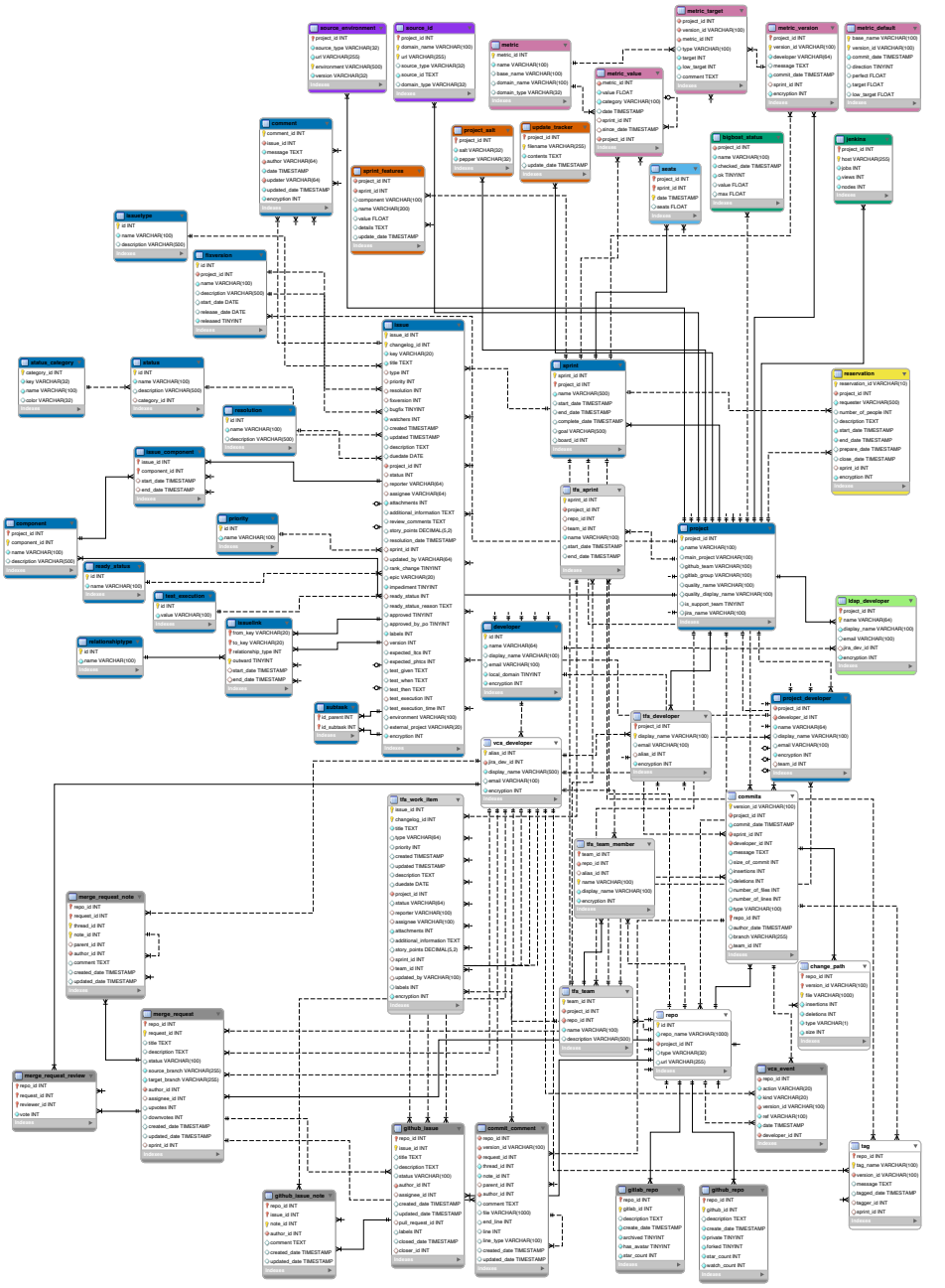


Figure 3.3: Complete ER diagram of the entities and relationships, based on the table schema, depicted as UML classes.

Many software projects arrange their development process using these systems. Some of the smaller portions of the model are not mentioned in full detail, while additional tables for source tracking and internal up-to-dateness checks are out of scope here. Connections between data retrieved from separate data sources are described in Section 3.3.2. Some entities include sensitive data fields that are considered for pseudonymization or excluded from exports. We discuss how we encrypt these specific attributes in Section 3.4.

Jira

The core of our database model consists of entities and relationships extracted from the Jira issue tracker. This system is used by most of the SCRUM software development projects that participate in our research, with the primary goal of tracking backlogs of stories and other types of tasks—commonly known as issues—during sprints. All projects at ICTU use Jira, while Wigo4it instead uses the work item tracking provided by TFS/VSTS/Azure DevOps. Our design approach initially focused on the Jira system, but expansions to the model allowed us to store data acquired from other systems as well. Figure 3.4 displays the central portion of the model that describes the data from Jira, including most of the references between the involved entities.

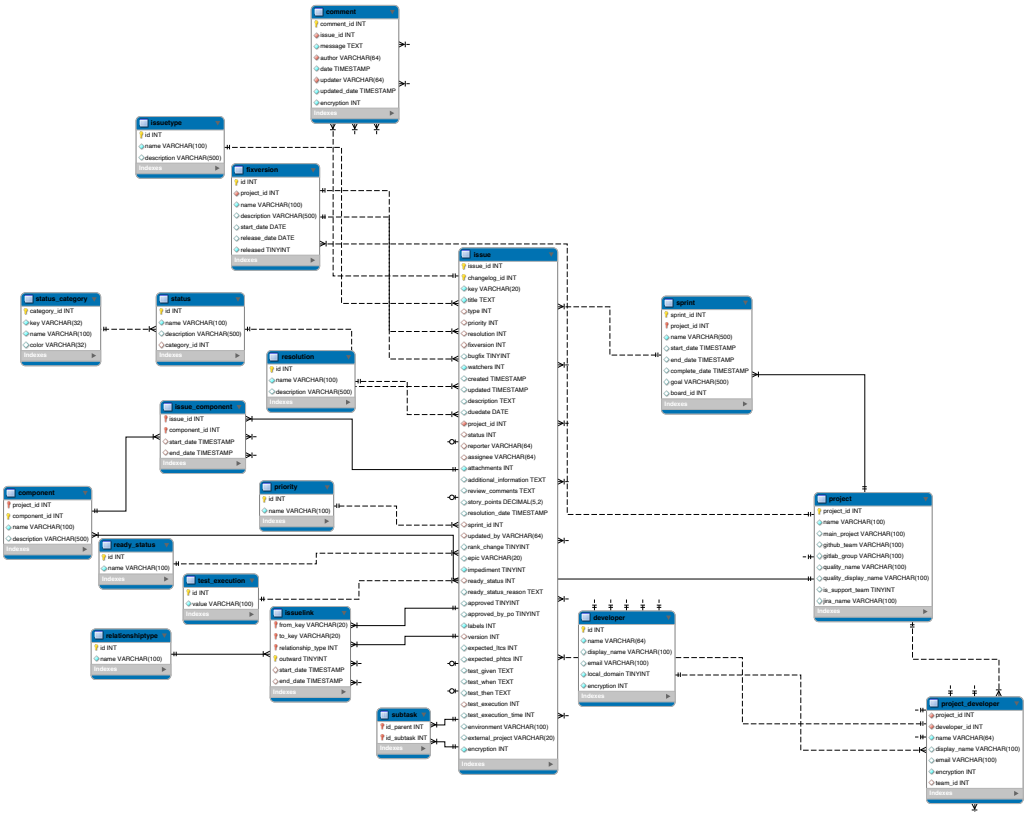


Figure 3.4: Entities and relationships that model data retrieved from Jira.

Jira is the prime data provider for a common entity that other portions of the model heavily rely on: the *project*. The main attributes describe how each project is named in the issue tracker, such as the prefix used by the issue numbering system and the human-readable name. The table also tracks dependent relationships to a parent project, although this is formally a weak reference, since the parent might even be missing from the data set. Through external means, we have more advanced methods of combining projects worked on by the same development team. The project entity is augmented to include attributes deduced from other sources, for more ways to refer to the project or its main assets.

The second pivotal entity is the *sprint*, which tracks important events. The start and end dates are predefined by the Product Owner (PO) when determining which stories to work on. Meanwhile, the completion date of a sprint is deduced from the moment that the team finishes the last of its stories, possibly before or after the planned end date. Other attributes are the description of the sprint goal (SG) and an identifier to the board that we use to construct a URL to the sprint at the issue tracker.

Jira is also a source for determining which *developer* works on which project. Personal data from the user profile, such as name and email address, is stored and encrypted in this table. A second table acts as a relationship between developer and project, using a project-specific encryption key for the duplicated attributes with personal information. This allows later imports and reports to cross-reference the existing data, as long as the encryption keys are available. Section 3.3.2 describes the encryption and linking of personal data in more detail.

The largest entity of the model is the *issue*. Each version of an issue is stored, containing many attributes where a few have been altered compared to the previous version, such as the description or the estimated story points. Some of these are complex, meaning that additional attributes about each of them are stored in a separate table. This includes what type of issue it is, what software component it is for, priority-related attributes, which release version it is or will be fixed in, the status it is in and—if it is done—what resolution it was given. An organizational instance of Jira is often extended with custom fields. The attributes currently modeled are sometimes specific to ICTU, but configuration allows treating the fields differently for another organization. The data acquisition component [a] of the Grip on Software pipeline provides a mapping from custom fields to the recognized attributes. Beyond that, we acquire the comments made on issues, with metadata on developer and time. Finally, developers link issues to others through different types of relationships, with one specialized relation for subtasks. Some of the complex attributes and relationships track their own metadata and chronological information.

Version control

Software development uses generic systems like Git [II] and Subversion [XIV] for version control. Despite differences in how these two systems track changes to code, they are conceptually similar enough to facilitate an abstraction in our data model. In Figure 3.5, we show the entities and relationships that make up this portion of the model.

A version control system provides development teams with the possibility to store and alter code collaboratively. Code related to the same software component is placed in a *repository*, sometimes abbreviated as repo. This entity is central in our modeling of this portion of the database schema. A URL attribute helps with tracking where the code is hosted.

In the context of a repository, a change to the code is considered a *commit*, with increments of commits leading to the most recent version of the code upon a branch. Usually, there is one

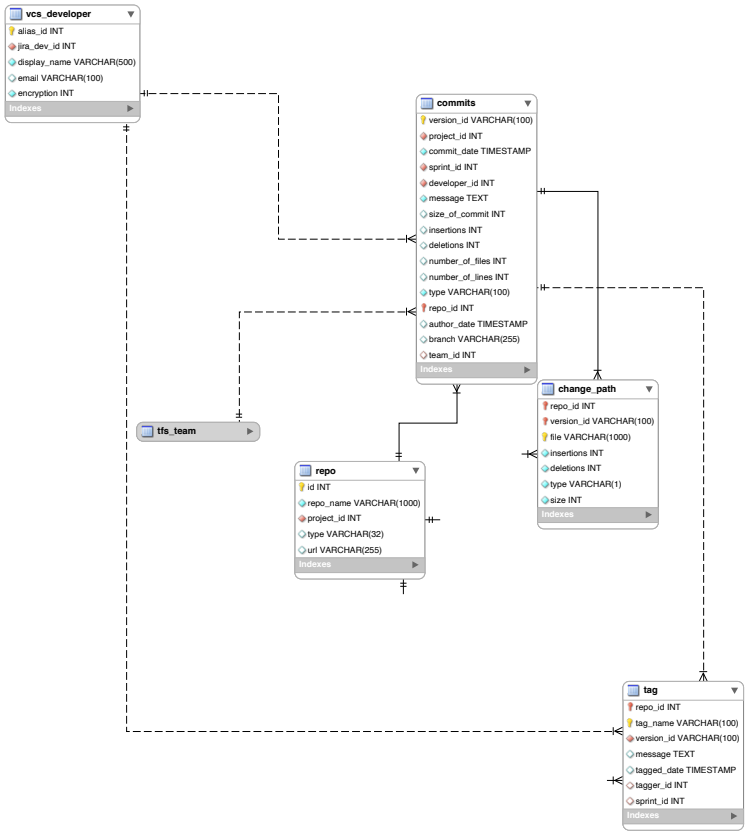


Figure 3.5: Entities and relationships that model data retrieved from version control systems.

main branch which is meant to remain stable, while code for different stories is often developed and tested on separate branches. The branch that the code is initially worked on is stored as an attribute in our model, along with other metadata about the date, message and size of commit. Separately, we track the path to a file that was changed by the commit as an entity related to the commit, with specific statistics similar to the size of the commit.

A commit can be given a *tag* with a version number, which is an indicator that the version of the code is used as a release version or for other purposes. We store the tag as an entity with attributes for the tag's name, message, the developer made the tag and when it was made. Both a commit and a tag are linked to the sprint in which the specific entity was made.

An author of a commit or tag is considered a local *developer* to the version control system. We decide to encrypt personal data like name and email address using a project-specific encryption key, cf. Section 3.3.2.

Code review

Recent version control systems are often accompanied by web applications that allow development teams to propose and review code changes. They are often tightly integrated with the repository

and provide additional data about the development process. We consider three systems that provide the review functionality for their code repositories: GitHub, GitLab and TFS/VSTS/Azure DevOps. In Figure 3.6, the entities and relationships are shown for these specializations, with some entities performing a generalized role for multiple systems.

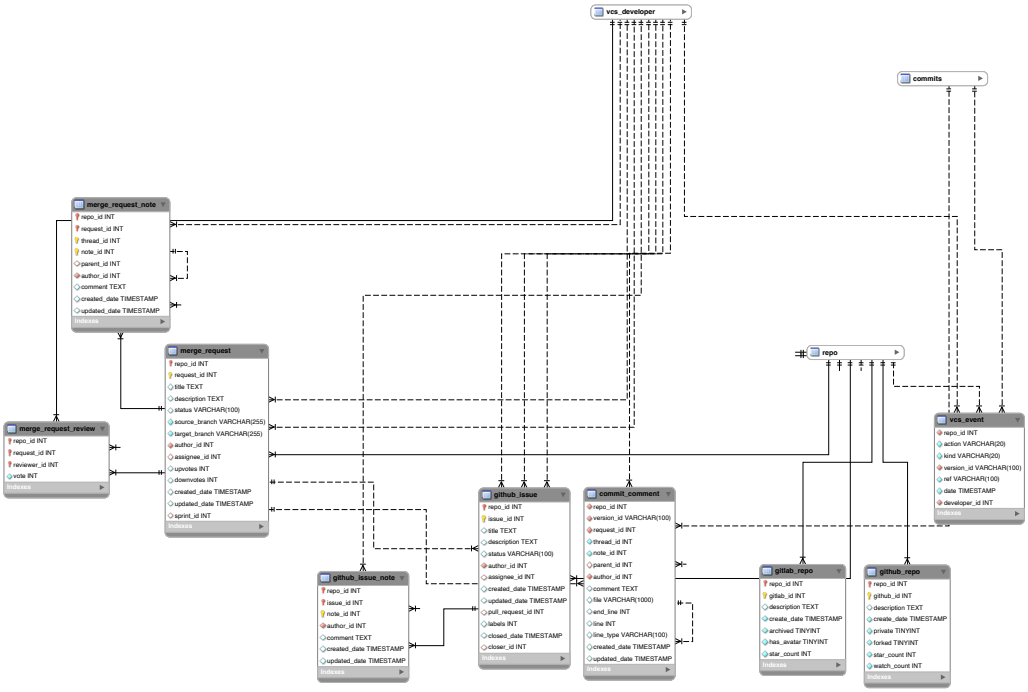


Figure 3.6: Entities and relationships that model data retrieved from code review systems.

We acquire additional metadata regarding the code repositories at GitLab and GitHub. These attributes are stored in specialized tables accompanying the existing entity. We obtain the description, creation date, access restrictions and social interactions such as the number of people giving a star to the repository, as a measure of importance.

The code review systems also contain additional information on interactions such as pushes of commits, tags and branches to the central repository. These *events* are usually not tracked in the Git repository itself, but the time difference between a developer finishing a local code change and publishing it has some relevance for our analysis.

The remaining entities from the code review systems are focused on comments on code, branch merges and standalone issues. A *commit comment* can be made for a code commit in order to discuss one or more lines of code, which were possibly changed by that commit. It is usually related to a *merge request*, which is a discussion regarding the merge of a branch to another—usually main—branch. Depending on the system, a reviewer is assigned to a merge request, who then votes on whether the code that was changed on the branch is ready. Involved developers and bots connected to the build platform and quality control systems sometimes also leave notes on the merge request itself. Additional metadata attributes are acquired for these requests and comments.

These review systems are also able to track issues. This functionality is, however, not used by most teams at ICTU, where they instead track issues in Jira. Issue tracking is done for projects that use GitHub, in particular a few open source projects from support teams at ICTU that are of interest to us. Wigo4it does use TFS/VSTS/Azure DevOps as a work item tracker, as explained in the next portion of the model.

Although limited in use at the organizations in our research project, we consider tracking all types of issues from such sources—including GitHub—to be helpful for the generalizability of our data acquisition pipeline to other development ecosystems, where the workflow of preparing and reviewing tasks may be conducted differently. Thus, we model those issues and notes as entities with similar metadata as the merge requests and notes.

TFS/VSTS/Azure DevOps

The integrated system provided by Azure DevOps Server, previously known as Team Foundation Server (TFS) and Visual Studio Team System (VSTS), encompasses version control via Git—although some older versions only provide its own TFVC protocol—with code review, build platforms and project management through work item tracking. This provides development teams with various options for tracking development process and delivering product increments.

Some organizations use certain functionality of this system or use it differently. At ICTU, TFS was only used by a few projects for version control and code review, with most preferring GitLab. Meanwhile, Wigo4it utilizes more from TFS/VSTS/Azure DevOps by performing sprint planning through work items, next to the version control.

We consider the portion that focuses on the work items to be separate from the version control and code review, which is modeled as part of the other specialized review systems that our schema supports. In Figure 3.7, we display the main entities and relationships that are extracted from the process of work item tracking.

The Azure DevOps system groups developers into teams with their own work item boards. These teams have team members, which have personal data that are encrypted with a common encryption key. Because some people are not part of a team but still interact with the work items—and conversely, we will not find every developer through work item updates—we also acquire developers through information from the work items. This information is stored in a secondary table, with personal data encrypted using a project-specific key. These two entities fulfill analogous roles to the developer tables from Jira, with data acquisition taking place in a similar way for that system.

Moreover, we acquire information regarding sprints from Azure DevOps. Due to differences in the attributes of entities compared to sprints from Jira, we store them separately. The same applies to the work items that stem from this system. Because the focus of our analysis was more clear once we adjusted our data acquisition pipeline to collect the work items, we extract fewer attributes from them. Some of the attributes also have different semantics compared to Jira issues, so the records for work items are stored in a separate table from that system.

Due to this, some of the entities have a dedicated role. When we later want to collect attributes or statistics from these entities, we have to select which data source is relevant to us and adjust the queries based on the definition of the attributes, such as what the status or resolution means to the work left on a story. We discuss some of the intricacies of this process for the way the queries are built in Section 3.3.2.

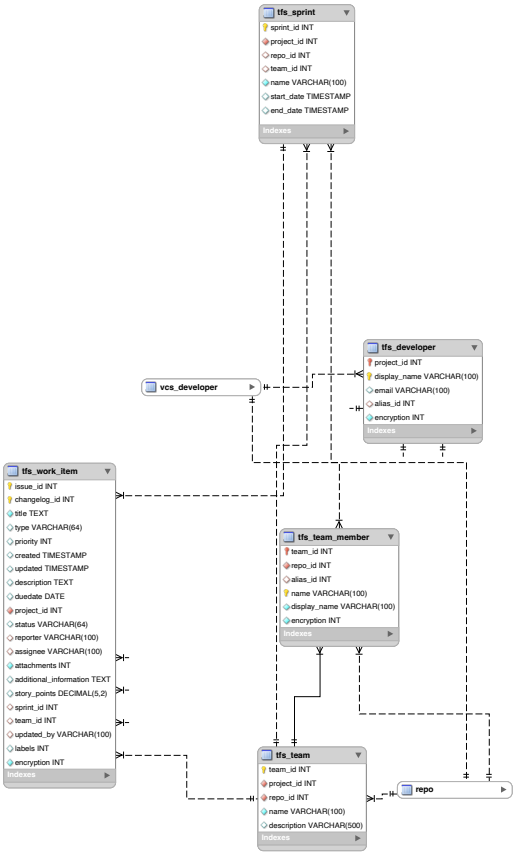


Figure 3.7: Entities and relationships that model data related to work item planning retrieved from TFS/VSTS/Azure DevOps.

Quality control

Code should be regularly inspected for code style issues, vulnerabilities and other indicators that technical debt are present due to maintainability issues. A system like SonarQube [V] automatically performs such checks. ICTU created an additional system to combine the history of those checks as well as reports made by security scanners, build systems and Jira in a central place that is easily accessible to team members. This system, Quality-time [VI], provides measurements in a form that we also model in GROS DB—while remaining compatible with SonarQube directly—as shown by the entities and relationships in Figure 3.8.

The quality control system formats the checks and measurements as a report that displays statistics grouped by software component. Each *metric* has a name, based on what is being measured and which domain object is involved. This domain object refers to other systems and artifacts, such as a code repository, document, build server or Jira board.

The metrics are checked at high frequency for updated information from external systems. A metric value comes with metadata about when the check happened, when the value has most

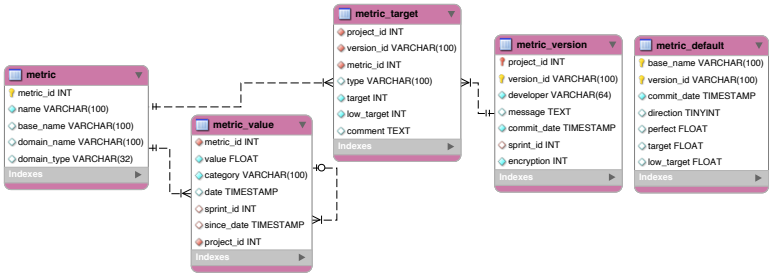


Figure 3.8: Entities and relationships that model data retrieved from quality systems.

recently changed before then and how problematic the value is. This severity category is defined by the target of the metric, which determines whether a lower or higher value is better. The target of a metric also has attributes for threshold values, which determine when the metric should be reported as acceptable, problematic or critical.

The target of a metric can be changed by a development team member or quality manager, usually if it is considered less or more important. Metadata on the version of the metric target is modeled as an entity. Finally, we track what the default value of each target is, based on the version of the Quality-time system, since the default could too have been altered. Combined, these entities allow us to deduce the context of a metric's value at an earlier date. Namely, we check why it was in a certain severity category back then and whether it would still be problematic now.

3.3.2 Linking data sources

The GROS DB contains data regarding events and instances of entities from different points in time, but also from different facets regarding several SCRUM software development processes. Work is undertaken on stories from a product backlog by refining the task description and enriching other attributes and links. Meanwhile, the developers write code that implements the expected change to functionality. The updated code is reviewed by their peers, in additional to quality control systems checking if the software remains maintainable.

While the process itself appears streamlined, one hindrance is that the steps are often separated, with limited linking between the entities and the systems they originate from. This reduces the perceptibility of potential bottlenecks and other issues that emerge during the process, because the ecosystem lacks a single view in which every team member is able to spot those problems. The lack of linking limits the capability for further analysis of patterns in development projects, as these relations are needed to gain a complete picture of the state of a project at any given moment.

Nevertheless, we automatically deduce relationships between different entities, which are then used to combine previously unlinked data in our queries. During our data acquisition, we define beforehand which systems, code repositories and other primary entities are in use by a certain project. We collect the data on a project-by-project basis, keeping this identifier in mind when importing the entities into the database. This gives the project entity a central role in our model. While this is sufficient for some analysis, we prefer to give teams a more relevant role than projects, when multiple projects are worked on by the same team, thus giving a better understanding of their total workload. For such purposes, we combine them through data analysis afterward or use the team entities from TFS/VSTS/Azure DevOps.

Next, in order to understand what is happening during a single sprint, we compare timestamps from the originating systems. For any event not yet linked to a sprint, such as a commit, metric measurement or adjustment to a metric target, we determine in which sprint it took place by finding the sprint with the closest start date before the event's timestamp, provided that the sprint did not end before then. It is beneficial to store this explicit link rather than attempt to find the relevant sprint during a query, because the latter approach would possibly find multiple or even false-positive connections, due to overlapping sprint dates. A "smarter" solution utilizing bisection properly handles these situations by identifying the most relevant sprint.

There are still limitations to automated relationship inference. When used on user-generated data, such practice becomes error-prone. For example, we wish to find relationships between code commits and the stories that they are being made for. One convention is that the developer writes the issue key or work item number in the commit message, but extracting the relevant identifier from free-form text could lead to matching unrelated substrings or mentions of irrelevant stories. There is also not a fixed custom for this, so it does not guarantee proper results for other projects and organizations. This obscures the semantics and proper use of a complex many-to-many relationship. Results from further analysis using this link would be incomplete, hard to understand or incorrect. Therefore, we have decided not to extract relationships from unstructured, textual attributes.

Another way to find what work is being done by the team in different systems is to determine simultaneous actions by the same developer. Initially, our model shows that personal data regarding each developer is scattered across portions of different systems. By linking developer entities across systems when they refer to the same person, we find the developer's affiliated activities in certain time frames. This method is potentially inexact and ambiguous when combining the data, but it allows us to filter out work that is not relevant to the developer through other means.

Problems with a person-based link arise when a developer chooses to use another name or email address in a version control system (VCS), thus having separate profiles. One example is a legal name in the employee records, stored in, e.g., LDAP, which differs from the name used on a day-to-day basis. Therefore, we utilize a mapping that links developers with different combinations of name and email address. The mapping is further employed to detect non-human accounts, such as automated code changes, by explicitly omitting a linked primary account. We then ignore those 'bot' accounts in further analysis.

Another important matter using personal data is the consideration of privacy. We prefer not to include information that can be traced back to a specific individual. We encrypt these attributes with a one-way encoding, in such a way that the developers are given a hash-based pseudonym. The mapping still detects identical developers as long as the encryption keys are available; both the mapping and encryption keys are only stored at the organization that the data originates from. Some data is encrypted with project-specific encryption keys, so developer names are encoded differently across systems if another key is used. For this reason, we have different tables with data using global encryption keys and project-specific keys, as indicated in Table 3.3.

We use `JOIN` operations to involve multiple entities through their relationships, allowing these queries to select attributes from several tables for further analysis. Another method of obtaining data from multiple originating systems is to combine them afterward based on primary identifiers. This way, distinct features have their own queries, and multiple features are combined into one data set based on project and sprint identifiers, for example. This adds processing time after performing the individual queries. Only after this step, the data set is made available to machine learning, where each sample in the data set elaborately describes a sprint.

System	Global encryption table	Project encryption table
Jira	`developer`	`project_developer`
TFS	`tfs_developer`	`tfs_team_member`
VCS	*None*	`vcs_developer`
LDAP	*None*	`ldap_developer`

Table 3.3: Portions of the database with personal data of developers as well as the tables in which we store said data using global encryption keys and project-specific keys.

We have designed another method to select and combine data from our cross-referenced database which takes advantage of the integration of the R programming language with MonetDB. We augment the syntax of SQL with a templating system which allows defining which tables, columns and relationships are involved in a query. This makes it possible to use the same query template for data selection from Jira entities as for those from TFS/VSTS/Azure DevOps, for instance. The columns involved in a relationship are provided separately so that the `JOIN` operation remains flexible. Similarly, we reuse definitions, such as what kinds of issues are considered stories, in more queries for other features or reports, by placing them in a central bank. Technical details are provided in Section 3.4.

After selecting the data from the relevant entities, we also want to report which sources were involved in the query. Tracking these sources is relevant for verifying if the information from this query properly reflects what the originating systems display. This way, there are no unforeseen conflicts between those systems and the database report. This also helps with making the query more insightful for stakeholders, who otherwise only see a number or other attribute without context. The data acquisition component of the pipeline tracks URLs of the systems that it requests data from. Additionally, the quality control system provides metadata to describe the code repositories and other monitored artifacts. Human-readable names make the references more familiar to the developers. For most queries, we directly link to the specific source, which is usually a report in the originating system showing the same information.

3.4 Architecture

The database must work properly and in accordance with our goals. Therefore, in this section, we focus on the technical components that we design and implement, supporting the desirable operation of the database. This includes integration of the database with the Grip on Software pipeline, which ranges from data acquisition to machine learning and information visualization.

The main purpose of the pipeline is to frequently provide new insights into patterns found in the wealth of data regarding Scrum software development processes. The pipeline, including the database, should not cause a huge strain on the existing platforms and processes. We keep in mind that we deploy the database to multiple software development ecosystems. As such, we use data from various issue-tracking project management systems in a similar fashion, for example. In addition, the database component necessitates efficient data backup and recovery functionality, allowing encrypted exports to be uploaded to a central instance for cross-organizational research.

The database administration and import component [d] plays a central role in meeting these objectives. A Java-based program uses Java Database Connectivity (JDBC) as a straightforward

method to interact with the database [XVII]. We import fields from JSON artifacts provided by the data acquisition component. First, the program selects which tasks are relevant to be performed for the provided collections of entities. Then, for each collection, the importer reads the objects for the entities one by one, using a custom buffered line reader for memory efficiency. An entity in the collection is used in a check to determine whether there is an existing row describing the same entity. Depending on the outcome, the database is given an update with the new data using either an `INSERT` or `UPDATE` statement. To sharply reduce the number of statements being sent to the database during the import, the aforementioned checks and updates are performed using *batched statements* [56]. Here, a subset of—or all of—the actions are performed in quick succession on the database's side through the use of a precompiled query template. During the import, MonetDB fills in the sets of values to check for and/or to store in the database, saving time transferring and compiling the query [57].

The importer program has more tasks next to revising the knowledge base on relevant entities. We determine the relationships between data from different systems here, for example when it comes to the sprint in which an event took place. The same applies for developer profiles, as described in Section 3.3.2. Other tasks include normalization of metric names, tracking the involved data sources, aligning changelog numbers and encryption of personal data.

The final encryption step takes place after the linking of developer data has been completed, because it is impractical to find relationships when all the attributes of the involved tables are encrypted. The data acquisition agent already encrypts personal data from version control systems at this point, while the names and email addresses from the issue tracker are still obtainable. Encryption keys use two attributes, a salt and pepper, which are produced by a cryptographically strong pseudorandom number generator. The sensitive data field is then encrypted using the salt, original value and pepper to form a SHA-256 hash [58]. The type of encryption keys used to hash the values is stored in a bit field attribute. The bit values indicate whether the encryption was done with a global encryption key, a project encryption key or both, in a particular order. Double-encryption is not usually done in our database, as seen in Table 3.3.

We also consider the database component's external security. Usually, the database is hosted in an ecosystem with limited access from the internet. To prevent potential dependency problems with separate firewalls, we restrict access to the port on which database connections are made. This does not impact any legitimate uses if all the processing takes place on the same server. This works well for a pipeline that is deployed on VMs, Docker platforms or other virtual network ecosystems, where forwarding firewalls or port mapping further restricts access to the database system.

Using the database administration component, we enable several maintenance tasks to be performed on the database. The database model and its synchronicity with the live system is important. We automatically validate the schema that is used to create the database tables against documentation, which compares properties of columns and primary keys, such as their type. If we want to adjust some of the properties or track a new entity, attribute or relationship, then upgrades to both the schema and the documented model must take place. We use a data-driven approach to determine whether the live database requires a schema change. We indicate for each portion of an upgrade what kind of action will be taken. If the action is feasible, i.e., the table or column can be created or adjusted compared to the current situation, then we perform the action. This avoids performing upgrades that we already applied to the live database. Further, it allows staggered updates which change a column of a table that is created by an earlier upgrade, without interfering with each other. Thus, the schema changes take place in an appropriate order.

Other maintenance tasks focus on database dumps, bulk imports and restoration. These auxiliary systems are configurable for different use cases. This has allowed the dump and import functionality to be reused in a cross-pipeline exchange setup. We do this along with a Java program for database export specifically for our use with MonetDB [e]. This program takes care of tables with encryption fields and large tables, so that the dump size remains manageable. The exported dump uses a hybrid format of CSV data and SQL instructions. Another module further encrypts the exported data using an asymmetric key-sharing setup [f]. A separate, private key of the organization and the public key published by the central pipeline instance are involved in the GPG encryption [IX]. The dump—without any unencrypted personal data or the encryption keys—is then uploaded via HTTPS to the central instance. At this web server [g], the payload is decrypted using the public key of the organization, which is known in advance, as well as the private key of the central pipeline. This ensures that the message cannot be compromised. Automated incremental dumps are thus regularly exported from the organizations and imported in a central database for analysis.

Aside from the bulk importer and export handler, the main pipeline component that accesses the database is the data analysis component [h]. We consider the feature extraction that this component performs in more detail in Chapter 4. From a technical standpoint, we use the query templates and definition banks introduced in Section 3.3.2 in order to determine which entity tables, attribute columns and relationship references are involved in each query, allowing us to build a generic and reusable data set. This integration of the R programming language with the MonetDB database uses an interpolation-based compiler, with recursive steps to expand contextually-defined variables and function calls into correct SQL. In Figure 3.9, the input data and supported functions are summarized.

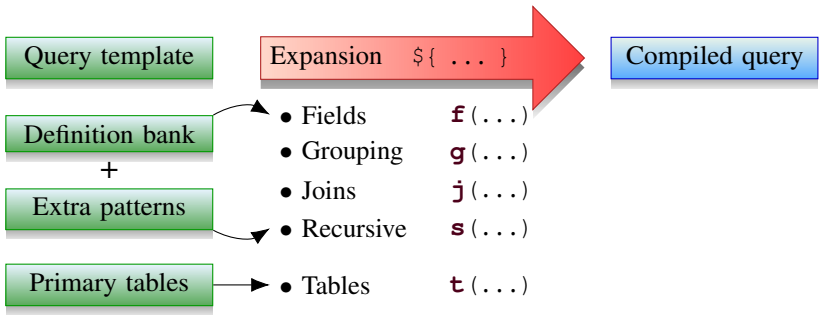


Figure 3.9: Input data (green blocks) and expansion functions of the query template compiler (red arrow), leading to a concrete query (blue block).

We show an example of a query template and a representation of the concrete queries for Jira and TFS/VSTS/Azure DevOps in Figure 3.10. The definition bank is extensible with additional patterns. Recursive expansion of these fields means that any template variables that they contain are further translated. A mapping of tables that are relevant to certain templates allows for translating the specific table where an entity is stored. For example, the issue entity expands to the `issue` table for Jira or `tfs_work_item` for TFS/VSTS/Azure DevOps.

```
1 SELECT DISTINCT ${f(join_cols, "issue")}, ${s(issue_key)} AS
      key
2 FROM gros.${t("issue")}
3 JOIN gros.${t("sprint")} ON ${j(join_cols, "issue",
      "sprint")}
4 WHERE ${s(issue_story)}
5 AND ${t("issue")}.updated > ${s(sprint_open)}
6 AND ${s(sprint_id, "issue")} <> 0
```

(a) Template

```
1 SELECT DISTINCT issue.project_id, issue.sprint_id,
      issue.key AS key
2 FROM gros.issue
3 JOIN gros.sprint ON issue.project_id =
      sprint.project_id AND issue.sprint_id =
      sprint.sprint_id
4 WHERE issue."type" = 7
5 AND issue.updated > COALESCE(CAST(sprint.start_date AS
      TIMESTAMP), CURRENT_TIMESTAMP())
6 AND COALESCE(issue.sprint_id, 0) <> 0
```

(b) Compiled for Jira

```
1 SELECT DISTINCT tfs_work_item.team_id,
      tfs_work_item.sprint_id, CONCAT('#',
      tfs_work_item.issue_id) AS key
2 FROM gros.tfs_work_item
3 JOIN gros.tfs_sprint ON tfs_work_item.team_id =
      tfs_sprint.team_id AND tfs_work_item.sprint_id =
      tfs_sprint.sprint_id
4 WHERE tfs_work_item."type" = 'Product Backlog Item'
5 AND tfs_work_item.updated >
      COALESCE(CAST(tfs_sprint.start_date AS TIMESTAMP),
      CURRENT_TIMESTAMP())
6 AND COALESCE(tfs_work_item.sprint_id, 0) <> 0
```

(c) Compiled for TFS/VSTS/Azure DevOps

Figure 3.10: A query that retrieves issue identifiers in each project's sprints, for a feature that counts the number of user stories, with different concrete versions that are expanded from the template when selecting data from the database.

3.5 Experiments

We test the GROS DB setup with regards to the performance of the database system during the usual workload of the data collection and analysis pipeline. To do this, we propose a number of experiments that look into the performance of MonetDB itself, as well as optimized refinements made to the queries in our template compiler.

Our scope is not to compare MonetDB to other database management systems. We consider the choice for MonetDB to be most reasonable at the moment when we initially created the pipeline. We considered the strong qualities of a column-based storage, relational SQL support and integration with a programming language for its functional deployment in a larger diverse development and research environment. In particular, a performance comparison with another RDBMS would require rethinking the data model for that particular database system. We would have to decide which indexes to include in the model, whereas MonetDB does not require manually-created indexes. Other data type constraints have been made specifically for our feature set and would require a different representation.

Comparisons between database systems are often done with industry-level benchmarks, such as TPC-H [XVIII]. We instead focus on a test set with a small number of queries that we have used during the Grip on Software research. The selected queries should be representative of the usual data collection for further analysis, machine learning and information visualization. Various parts of the data model are involved in the queries. Varying levels of complexity are used in order to provide the feature set and associated details in the query response.

We measure the performance of query templates that we have optimized with our compiler. In order to find out how much these changes improve the efficiency of database system, we go back to older versions of the query stored in our repository and include those in our test set, along with the newer version. We retroactively apply changes that we made to the query in the meantime, but only in case that these changes are not relevant to the performance.

3.5.1 Setup

In our performance experiments, we consider only the data access statements (`SELECT`), not other kinds of queries for data definition or data manipulation. The queries that build the database and fill it are not part of a typical workload in our consideration. In fact, the data import is already covered by other pipeline performance measurements in Section 2.5.

We do involve new database creation and storage in our experiments by testing queries in two situations. First, we run queries one by one on an existing database which has seen the queries before and could create optimized structures for them. This is considered a *hot-start* experiment. We test the same queries again, this time individually where we rebuild the database and import the database, in between each query run. This *cold-start* experiment provides more indication of the inherent complexity of each query for our data model.

We disregard caches in between cold-start runs. In all experiments, we disable the use of tables that hold outcomes of long-running queries. The performance test should not be interrupted by other processes when possible, thus other parts of the pipeline are disabled during the experiments.

Our performance test program reuses portions of the data analysis program that runs the queries [h]. Additionally, it ensures a reproducible and consistent setting for all the queries. We perform multiple runs of each query in order to obtain statistics on deviations. This allows us to determine if the test setup behaves similarly between runs.

The data set of ICTU is used as a representative instance upon which we apply our queries. Some relevant dimensions of the database are listed in Table 4.1, with 192 million measurements of quality control metrics on top of that.

The tests are performed on a Dell PowerEdge 2950 2u rack-mounted server with a Intel Xeon 8-core X5450 CPU at 3GHz, with 2×128KiB L1 cache, 2×12MiB L2 cache, four 4GiB memory cards of DDR2 FB-DIMM RAM of Hynix HYMP351F72AMP4N3Y5 (667MHz) and a 544GiB DELL PERC 6/i SCSI storage disk with LUKS-based 512-bit AES full disk encryption enabled. The database system under test is MonetDB v11.41.5 (Jul2021).

3.5.2 Results

We run six queries related to two large entities in our data model, namely metrics and issues. Most queries make use of related entities, such as sprints. The query templates and compiled versions are provided in Appendix B.

We collect the wall clock time that each query took, from the start of the query request until the data response, as well as the system time that the database used during the full query processing. In addition, we measure the average CPU load during the execution of the query. Finally, we track how many database rows were included in the query. This statistic occasionally differs between refined and original versions, due to a different method of linking with related sprints or by filtering operations. This is because some sample rows are not used in the data set during normal data analysis operations. Only sprints that exist are included in the data set. Thus, the query filters such sprints beforehand.

The performance measurements of the test results on a hot-start database are provided in Table 3.4 for the original queries and Table 3.5 for the refined versions, with 10 runs for each query. We report the mean and standard deviation of the values obtained from those runs. It is clear that the queries that we refined are faster, even if the database also optimizes the queries itself. Our optimization also appears to help with reducing the variance between runs, allowing for queries to run frequently and in a stable rate during research.

QUERY	WALL TIME	RUN TIME	CPU LOAD	ROWS
All metrics (Figure B.1)	11.29 ± 0.32	11.02 ± 0.33	81.3 ± 3.0	1898
Red metrics (Figure B.3)	2.01 ± 0.14	1.74 ± 0.13	72.1 ± 5.7	1775
Team spirit metric (Figure B.5)	10.77 ± 0.34	10.50 ± 0.35	76.4 ± 3.3	1063
Backlog added points (Figure B.7)	1.20 ± 0.03	0.93 ± 0.02	38.3 ± 1.1	16823
Backlog epic points (Figure B.9)	7.98 ± 0.09	7.70 ± 0.09	44.4 ± 0.7	32246
Backlog story points (Figure B.11)	5.61 ± 0.15	5.35 ± 0.13	33.5 ± 0.7	153685

Table 3.4: Results of performance tests of original versions of queries, using a hot-start database system. Times are provided in seconds, CPU load in percentages.

The queries related to metrics use more processing power in order to generate the result. Consequently, they take longer to produce the resulting data set. The queries that calculate product backlog sizes at different moments in time have a large data set, but perform relatively efficiently compared to the metrics queries. With larger data sets from each query, CPU load and in particular query response times decrease. This indicates that query optimization through the template compiler serves a practical use, when queries are rerun many times.

QUERY	WALL TIME	RUN TIME	CPU LOAD	ROWS
All metrics (Figure B.2)	5.76 ± 0.08	5.48 ± 0.09	81.2 ± 1.5	1899
Red metrics (Figure B.4)	1.76 ± 0.04	1.49 ± 0.03	79.3 ± 1.6	1776
Team spirit metric (Figure B.6)	2.54 ± 0.56	2.28 ± 0.04	79.5 ± 1.3	1063
Backlog added points (Figure B.8)	0.88 ± 0.04	0.62 ± 0.02	31.4 ± 1.3	16823
Backlog epic points (Figure B.10)	3.83 ± 0.07	3.57 ± 0.07	26.8 ± 0.9	25833
Backlog story points (Figure B.12)	4.05 ± 0.06	3.79 ± 0.06	27.4 ± 1.2	88032

Table 3.5: Results of performance tests of refined versions of queries, using a hot-start database system. Times are provided in seconds, CPU load in percentages.

We measure the performance of the same queries in a cold-start setup, where the database is restarted and all system caches are removed. The original and refined versions of the six queries in our test set are each run ten times again. The results of these experiment are shown in Tables 3.6 and 3.7.

QUERY	WALL TIME	RUN TIME	CPU LOAD	ROWS
All metrics (Figure B.1)	50.92 ± 0.53	50.86 ± 0.53	20.1 ± 0.6	1898
Red metrics (Figure B.3)	32.03 ± 0.20	31.97 ± 0.21	10.8 ± 0.4	1775
Team spirit metric (Figure B.5)	69.79 ± 0.39	69.80 ± 0.39	15.1 ± 0.3	1063
Backlog added points (Figure B.7)	2.02 ± 0.10	1.97 ± 0.10	22.1 ± 1.1	16823
Backlog epic points (Figure B.9)	8.53 ± 0.09	8.46 ± 0.10	41.2 ± 0.6	32246
Backlog story points (Figure B.11)	6.40 ± 0.11	6.34 ± 0.11	29.6 ± 0.5	153685

Table 3.6: Results of performance tests of original versions of queries, using a cold-start database system. Times are provided in seconds, CPU load in percentages.

QUERY	WALL TIME	RUN TIME	CPU LOAD	ROWS
All metrics (Figure B.2)	49.47 ± 0.49	49.41 ± 0.48	13.2 ± 0.4	1899
Red metrics (Figure B.4)	33.05 ± 0.25	32.99 ± 0.25	10.0 ± 0.0	1776
Team spirit metric (Figure B.6)	59.89 ± 0.70	59.82 ± 0.69	10.0 ± 0.0	1063
Backlog added points (Figure B.8)	1.61 ± 0.15	1.55 ± 0.16	17.5 ± 2.2	16823
Backlog epic points (Figure B.10)	4.45 ± 0.08	4.38 ± 0.08	23.2 ± 0.6	25833
Backlog story points (Figure B.12)	4.93 ± 0.17	4.86 ± 0.18	22.7 ± 0.8	88032

Table 3.7: Results of performance tests of refined versions of queries, using a cold-start database system. Times are provided in seconds, CPU load in percentages.

Again, we observe that the refined variants have a decreased query processing duration and load, although the effect is not as significant compared to the hot-start setup. One query, used to calculate the number of problematic red quality control metrics, is slightly slower.

All queries are faster to process in the hot-start situation than with the cold-start setup. This should be expected, given the presence of additional index-like structures that MonetDB generates

and reuses between the runs of the queries on a hot-start system. The queries that involve the metrics are several times slower in the cold-start setup, while the effect is less extreme for the backlog queries compared to the hot-start test. The average CPU load is however much lower during the cold-start experiment, for all queries. This may be due to longer run times spreading out the actual execution until enough data is obtained from disk. Another potential bottleneck is memory synchronization times between queries, allowing the overall load to decrease. In addition, there is less usage of intricate data structures that help with speeding up the query resolution but use more complex instructions.

Overall, the combination of a hot-start MonetDB database system and the refined queries seems to provide the most benefit to shorter query processing times and advantageous usage of the available resources. The higher load does not impact the system much, given that other processing cores are still available for other portions of the pipeline to be run. This allows us to use the GROS DB at various software ecosystems, which come with different hardware constraints and virtualization options.

3.6 Discussion

In this chapter, we build upon the introduction of the pipeline components in Chapter 2 and further detail the database system as a central component of the pipeline. During the Grip on Software research, there is a need to store information from several systems that are commonly used for SCRUM software development. We focus on modeling and integrating these data sources such that we link previously disconnected entities and perform frequent analysis on the data set.

MonetDB satisfies our purposes and requirements as an extensible RDBMS suitable in academic and corporate environments. The data model of GROS DB makes use of various data types provided by MonetDB, including timestamps and references between entities. MonetDB supports SQL as a core query language, but also enables extensions through the integration with programming languages such as R and Python. As a database system oriented to analytical processing, we find MonetDB to be the most applicable option for our research.

Our data model for the SCRUM software development process has many connections between different portions that roughly correspond to the systems that the data originates from. Initially, systems such as a project's issue tracker, version control system and quality dashboard lack substantial interaction. Through chronological information about events that make changes to entities, we build new relationships, where the sprint acts as a central entity. This model allows us to more easily extract metrics and features regarding this time frame.

Other links exist through the use of personal information, such as when a developer commits some code and resolves a story. We take privacy in mind by encrypting names and other identifiable attributes, with one-way encoding using project-specific encryption keys. We preserve links by using local translation mappings and comparing the hashes.

We enhance the selection of data from the model by introducing a query template compiler. This allows us to write queries that select the same kind of data, such as the number of stories worked in a sprint, while staying agnostic to the underlying portion of the data model used in the actual query. Depending on the organization and project, the templates are compiled to use, e.g., Jira or TFS/VSTS/Azure DevOps as a source. The relevant entities—stories versus work items as well as different types of sprints—plus the references used in `JOIN` operations on their tables are then automatically used in the proper locations.

There are technical challenges to make the database system stable, maintainable, interoperable with other parts of the pipeline and applicable in a wide range of software development ecosystems without many changes or influences on the existing infrastructure. Additional functionalities of the GROS DB allow us to fulfill these requirements, along with tasks for upgrading and exporting data to a central instance of the pipeline.

We test the performance of the database system through the use of a number of queries that are representative of our usual workload, along with older, non-refined versions that perform the query in a different way. We observe that the refined versions use less time to execute. Additionally, we show that MonetDB itself performs optimizations through the use of structures created when queries are run more often, which benefits our resource usage during our research. We easily perform our queries repeatedly whenever new data comes in. This is performed by a scheduled job system that runs hourly or even more frequently, depending on the other pipeline components.

More performance experiments could be done by comparing MonetDB to other potential database systems, such as MySQL, Postgres and SQLite [49]. In particular, MariaDB, as a fork of MySQL that includes options for column-based storage, is a consideration for further research options. Overall, the storage and performance options of MonetDB have proven to be helpful in our data modeling needs, as recognized throughout analytical processing problems.

We also consider an alternate solution where we integrate MonetDB with more Python programming, rather than with R. There exist many modules that make Python work well with column-based feature data, which allows deeper integration with machine learning purposes with fewer intermediate steps. Still, our query template compiler overcomes some issues that would need to be reimplemented in Python. We did purposefully design our query templates to not contain specifics of the language that the compiler is implemented in, in particular regarding the syntax of the expansion functions.

Our data model should be suitable for a large range of software development projects. Some portions are focused on a specification of data originating from specific systems, such as Jira and TFS/VSTS/Azure DevOps. Still, we easily support more integrated issue trackers and review systems, with GitLab and GitHub already part of the database model. Existing entities and relationships are proper templates as a starting ground for such extensions.

In summary, our research objectives are more easily attainable with the GROS DB database system based on MonetDB and extended with our query template compiler. We efficiently extract data sets based on the entities and relationships in our data model, with interconnections between portions based on discrete systems used in SCRUM software development. This data set accommodates different organizations while using the same query template, by taking advantage of the generic and flexible design of our data model and the technical component architecture. Thus, our pipeline remains adaptable to various development ecosystems. The performance of the pipeline component allows involved team members to benefit from recurring, large-scale machine learning and information visualization applications. By making the model inherently more understandable, the goals for improving predictability of the software development process become more feasible as well.

Chapter 4

Pattern recognition methods

GROS ML: Analysis of predictive patterns in SCRUM *software development processes through machine learning and estimation models*

Portions of this chapter are also published in the following journal and/or conference articles:

- Leon Helwerda, Frank Niessink and Fons J. Verbeek. "Conceptual process models and quantitative analysis of classification problems in Scrum software development practices". In: *Proceedings of the 9th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management* (IC3K 2017 - KDIR). SCITEPRESS, 2017, pp. 357–366. DOI: 10.5220/0006602803570366.
- Leon Helwerda, Walter A. Kosters, Frank Niessink and Fons J. Verbeek. "Estimation models for prediction of sprints and backlogs". *Empirical Software Engineering*, 2024. Submitted.

Abstract of Chapter 4

Background: Software development processes such as SCRUM produce various metrics during the lifespan of a project. We have constructed and analyzed a data set of metrics and features related to progress tracking retrieved from two SCRUM software development organizations, namely ICTU and Wigo4it.

Research questions: How can we improve the predictability of SCRUM software development practices, specifically the progress of sprints and backlogs—based on analysis of data selected from the development process—and how do we validate our approach?

Aims: We consider the use of story points in short-term and long-term planning to understand and improve existing best practices in the SCRUM framework. Our goal is to aid the planning aspect of the process through predictions of sprint velocities and backlog sizes, which is beneficial to the framework and workload balancing.

Methods: We apply analogy-based effort estimation, deep neural networks and other predictive regression models to our data set of extracted features. We determine the probable effort during future SCRUM sprint cycles and the rate at which a backlog of estimated and non-estimated work items, specifically in user story form, could be implemented. We validate the results of these predictions against earlier sprints and analyze distributions of Monte Carlo simulations.

Results: Our results of experiments with sprints and backlogs from 60 teams and projects show that the proposed methods for classifying sprints into finished and unfinished stories reaches an F1 score of 90%, while the analogy-based effort estimation is within 25% of the actual value about 89% of the time in validation. These are improvements over naive estimation methods and additional validation shows improvements for larger training sets. The backlog size prediction models have varying errors and deviations in multiple scenarios but overall appear consistent with the outcomes for earlier sprints, where the Monte Carlo simulation using all types of changes to the backlog shows the lowest deviations. Further analysis shows that the Monte Carlo scenarios also fit with the expected distribution.

Conclusions: We find that the approach is feasible, with our models allowing project leads and teams to self-manage sprint planning and product backlogs in terms of acceptable size. Additionally, we predict time points at which milestones could be reached.

4.1 Proposition

In a SCRUM software development environment, it is vital for all involved people to know how much work can be done within a certain amount of time. Many different factors play a role, depending on the period of time. Not only do the tasks differ in size and complexity, but also the team might change in composition and expert level.

The SCRUM framework intends to solve common issues related to planning in software development. On the outset, it is often not clear when certain tasks will be done, when they are completed and at what moment new tasks should be introduced. SCRUM at its core does not provide a general way to answer all these questions, especially when they are asked regarding a longer period of time. The framework does provide some elements that allow for adjustments in the process. We wish to understand the framework in more detail, assess its contribution to the successful release of a product, employ scientific methods that are novel to this area and construct machine learning (ML) algorithms that aid members of a development team in various phases of the process.

The research questions and sub-questions of the Grip on Software research project that we consider to be most relevant in the context of the analysis of regularities and peculiarities, i.e., pattern recognition (PR), of a software development process, are the following:

RQ2 How can we improve the predictability of SCRUM software development practices, specifically the progress of sprints and backlogs—based on analysis of data selected from the development process—and how do we validate our approach?

- **RQ2a** Which features can we select based on ongoing data from the software development process that are most indicative of the progress?
- **RQ2b** What kinds of learning algorithms can we introduce to this problem, which learn from these features and provide feedback on what kinds of decisions can be taken?
- **RQ2c** How do we predict the likelihood of timely and properly finishing a currently-running sprint or a longer-running period aimed at resolving a product backlog within a development project, even before it has started?
- **RQ2d** How do we validate that the predictions and recommendations are within our expectations and based on relevant, explainable factors?

With these questions in mind, we study the accuracy of four machine learning and statistical models which we apply to a large data set of both ongoing and finished software development projects, that took place at two Dutch governmental organisations: Stichting ICTU and Wigo4it. We aim to find meaningful patterns in this data set and identify relations between these patterns and the projects' outcomes. One outcome is at the end of a project, when the backlog is completed and development halts. We also consider the end of intermediate SCRUM sprints as outcomes, given that the development team delivers work that they committed to finishing by then.

Thus, two specific cycles in SCRUM are of interest to us: first, the mutations on the product backlog (introduction of new wishes, refinement, prioritization, effort sizing and moving work to a sprint backlog) and, secondly, the SCRUM sprint cycle (planning, developing code, testing, reviewing and retrospective).

We approach the matter of making reasonable predictions in the planning component of each of these repeating cycles. We propose methods based on machine learning and analogy-based effort estimation to automatically produce predictions. They should predict the amount of work remaining on a backlog, as well as the number of story points that could be pulled into a sprint so that it is still feasible to finish the work within the limited time. We validate whether these methods and results prove to be adequate and accurate. This approach then augments the process compared to the current situation.

In Section 4.2, we discuss the concepts and context of the problem statement in more detail. A study of relevant prior literature is found in Section 4.3. We describe the methods of our workflow in Section 4.4. Specifically, we consider research question **RQ2a** on feature selection in Section 4.4.1 and the learning algorithms of question **RQ2b** in Section 4.4.3. Our experimental analysis—in particular for answering **RQ2c**—is introduced in Section 4.5 and we discuss our observations and results in Section 4.6, which addresses **RQ2d**. We conclude our findings in Section 4.7.

4.2 Background

The SCRUM framework leads to a conceptual representation that helps us to determine the most relevant factors of the objectives within the software engineering domain. In Chapter 1, we provide a contextual overview of SCRUM, but we highlight the relevant points in the context of our analysis in this section.

As a framework focused on interactions and collaboration the SCRUM workflow includes a number of roles, events and common artifacts [6, 8 app. 2]. The main artifacts are the product backlog (PB), sprint backlog (SB) and the product increment, on which people with different (and possibly overlapping) roles can perform mutations. Desired features of a product are introduced and prioritized by a Product Owner (PO). In collaboration with a versatile team of developers, the PO refines some of the desired features into user stories. These can then be added to the backlogs as work that is ready to be picked up.

The members of the development team commit themselves to work on a selection of the stories during a short, fixed time interval known as a sprint. The developers implement the features into the product increment. In this way, the team can demonstrate the changes within the increment to representatives of the user, who provide feedback during a sprint review. Such a demonstration session is held frequently, thus allowing changes to be made based on the feedback soon after the increment. Apart from the user review, SCRUM defines a retrospective session in which the team can adjust how they approach parts of the process based on consensus.

As an Agile software development method, SCRUM focuses on putting team interaction over the tools used in the process [3]. Therefore, the introduction of new automated systems should focus on providing the team members with recommendations in such a way that they can do their work more efficiently and pleasantly. An automated system should remove load from the team rather than adding just another place to check for information. By focusing on the aspects of the Agile manifesto that are placed before the lesser-valued goals, we can ensure that our model and tooling fits in the landscape optimally [5]. We find that our proposed method adheres to the Agile guidelines. The system has been regularly used by teams to discover more about product and sprint backlog sizes. This allows them to determine the progression thus far as well as perform forecasting, both for short and long periods of time.

4.2.1 Framework

SCRUM is a software development framework that is intended to be lightweight and flexible, while describing a software development process according to a set of rules. In this process, a software development team works in sprint iterations of around two to three weeks in order to deliver working software product increments to the client at the end of each sprint. The team members commit themselves toward the amount of work they consider viable for each sprint.

Next to the team, the Product Owner (PO) introduces wishes—as indicated by the user—on a product backlog (PB), prioritizes them and presents them to the team. Together, they refine them into actionable tasks. These tasks, which come in the form of work items or most commonly user stories, are meant to be small improvements upon the current product. The team and PO collaboratively choose a selection of tasks to place upon a sprint backlog (SB). The team works on the current sprint backlog for a fixed amount of time and demonstrates the new features and improvements to representatives of the user. In collaboration with the PO, these representatives provide feedback on newly introduced functionality during the same review session at the end of the sprint.

The team's own retrospective meeting is meant to reflect and learn from situations that arose during the previous sprint. Team members ensure that problems and impediments emerging during the process are identified and dealt with by the Scrum Master (SM), a potentially alternating role within the team who may also oversee the implementation of the SCRUM framework—sometimes this is then referred to as a Scrum coach.

The SCRUM framework indicates several more roles and also defines artifacts such as the backlogs, stories and products. Stories, or work items in general, go through stages of refinement and progress during a sprint toward implementation within the product. When a story is finally considered to be 'done', it is removed from the sprint backlog. A team can adjust the definition of 'done' (DoD) in relation to their stories, which can include reviewing of code by peer developers, manual tests, coverage within an automated test based on code changes, inclusion in documentation manuals, user testing and acceptance by the client.

The software development cycle within SCRUM is known as a sprint. The workflow defines events surrounding and during the sprint, as outlined in Figure 4.1. This includes the refinement, demo and retrospective.

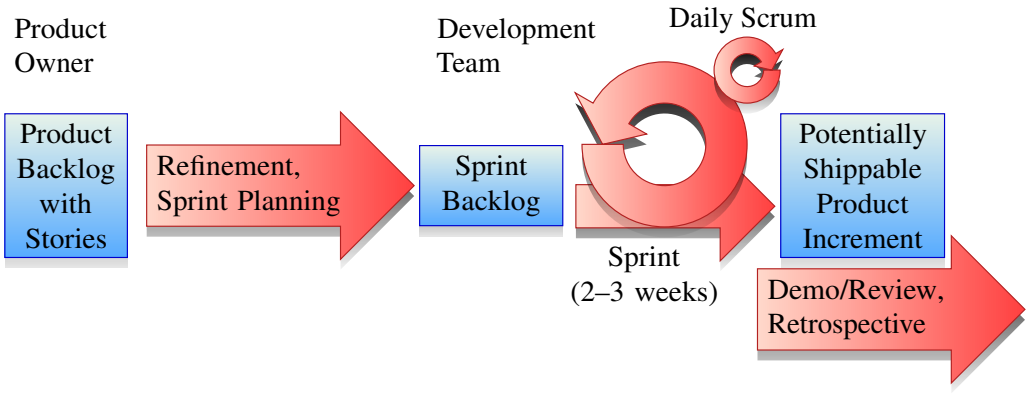


Figure 4.1: Workflow of a sprint in the SCRUM framework.

4.2.2 Story points and adaptations

When it comes to the start of a sprint, the PO and team determine the selection of stories to work on at the sprint planning. This is meant to ensure that all team members are aware of the scope of each story. Then, during every working day, the team gathers for a Daily Scrum—also known as a stand-up meeting—in which they describe and inquire into the status of each story. They discuss what the other team members have done up to that point, the work that they plan to do for the remainder for the sprint as well as any problems that they have signalled.

A major addition to the iterative planning process of SCRUM is that stories can be awarded effort sizing values known as story points (SP) during refinement. The determination of SP is up to the team, based on the complexity and projected workload compared to earlier reference stories, using a Fibonacci-like scale [9], such as the sequence

$$0, 1/2, 1, 2, 3, 5, 8, 13, 20, 40, 100. \tag{4.1}$$

Often, teams use a game-like format known as Planning Poker [60] to collaboratively choose the SP of a story. Each team member decides at the same time, for example by simultaneously showing a card with the number of story points they would assign to that story. Then, outliers are asked to explain their choice, until a consensus is reached.

The story points sizing is a metric that makes stories comparable to each other, to a certain extent. This way, the team members can determine how many story points they should commit themselves to in a sprint. Often, a metric of *velocity* is used as a guideline for commitment, where the sum of the story points of all stories that were resolved during the past three sprints is divided by three. This average indicates the team's recent inclination toward both the awarded story points as well as the amount of effort that they spent in completed work.

Comparing story points between teams is more difficult. Even if one team awards their stories with higher story points than another, this does not necessarily imply that they simply have a different viewpoint to the scale, since the stories themselves have a varying complexity as well. An "exchange rate" could still help understanding team differences, but it cannot explain them. Similarly, over time a team changes in composition, the software increases in complexity or the context requires a different approach to the planning aspect, and these influence the sizing as well. We consider the possible consequences of these changes in Section 4.7.1.

Sometimes, other effort sizing methods are employed within SCRUM and similar Agile frameworks. These methods have varying scales and sources of information, such as T-shirt sizing [61] or function point analysis [62]. Sometimes, these effort metrics can be translated or re-used as if they were story points, but caution should be taken to consider them as distinct metrics, as these methods focus on different levels of detail and context for a portion of work to be done.

Other software development methods have adopted similar practices like those in SCRUM in order to include effort estimation practices. There also exist extensions of SCRUM itself to work collaboratively in small groups, large organizations or co-located teams on shared, partitioned backlogs [63]. Such practices exist in Scrumban, scaled SCRUM and Scrum of Scrums, among others. Although we consider curation of data for teams working on multiple projects at once and vice versa, our focus here is on the most general form of SCRUM. It is relevant to note that each organization and team applies its own practices on top of the framework, leading to slight changes in how data should be interpreted for a proper understanding of what it means in reality.

Whereas the SCRUM framework and the Agile manifesto focus on team interaction, working software, collaboration and response to change, they also allocate some secondary value to

processes, tools, documentation and plans. Especially when such systems allow for detecting defects or other issues quickly by putting the most relevant information first, these systems aid with signalling impediments and reducing effort spent on technical debt, i.e., maintenance of code that ages with time. Such factors improve the throughput of stories from initial concept to working code. Systems that provide this kind of assistance could be considered as if they were an additional team member, since they should reduce work from the actual members.

Common types of systems in use in software development processes include software quality analysis tools, digital backlog and sprint planning systems, collaborative/distributed version control systems and forecasting systems for feasibility of planning. Often, the user interface of these systems is designed in the form of an information dashboard, where critical information relevant to everyone is placed first. Details are usually available through menus or APIs, which allow data acquisition, analysis and reporting by separate systems.

4.3 Related work

We explore existing applications of predictive models in Agile software development and SCRUM in particular. We consider the outcomes of those studies and attempt to find possible avenues for improvements.

Effort estimation of software development projects has been a field of sustained interest and has seen renewed interest through the introduction of novel techniques to the field. One such method, known as analogy-based effort estimation, uses relations between the most significant attributes of a sample set to produce both an estimation for a product backlog or a subset thereof. Additionally, it produces indicators of similar data points involved in constructing this estimation [22].

Various methods for finding similarities, ranging from distance measures to statistical inference [64] and fuzzy numbers [65], provide improvements to the analogy-based effort estimation algorithm. Several effects of identified patterns in data sets, e.g., outliers [66], have also been studied. Overall, analogy-based effort estimation has shown to have a feasible use within software development [67] and improvements in the reported accuracy can be seen for multiple algorithms and Agile frameworks [68].

Other algorithms, such as neural networks [69], have also been applied to effort estimation using specialized data sets. Recent work shows optimizations of basic neural networks are a feasible method for effort estimation [70].

Effort estimation within SCRUM and Agile in general mostly focuses on the story points that are awarded by the development team during Planning Poker. Extensions to this process have also been proposed [71]. Automated estimation is another researched topic [72]. Additional metrics have been introduced to the SCRUM framework, but even established ones, for example functional size metrics, may not always benefit the estimations [73].

A relatively new introduction to predictive methods within software development is the use of Monte Carlo simulations. Initial research shows that the use of such an algorithm may provide a reasonable estimation of the effort required to resolve a backlog of user stories, as well as a due date when such a collection of stories is finished [74].

In the context of Agile software development, most research focus lies on empirical and qualitative studies, such as survey evaluations [75]. There has been research into the relation between the Agile development methods—specifically SCRUM—and the product's eventual outcome in terms of software defects [76].

A large number of studies deal with distributed software development projects which use Agile software development methods. Although these studies provide relevant results for their topic [77], their use in collaborations for teams that are co-located at one site is limited. Studies show applications of SCRUM when there is a requirement of communicating with other teams and stakeholders on a frequent and methodical, documented basis [78].

Recent quantitative research covers topics including Agile software development processes and more specifically SCRUM practices surrounding work planning. Often, multiple metrics surrounding the process and the project are used to determine performance and success [79]. Selecting measures so that the team can effectively monitor the progress of their development is a key principle [80]. Important factors for the selected metrics include long-term usability and availability within the process [81]. Various sources are used to collect metrics, even going so far as tracking and using data for individual members of development teams [82]. The analysis of data from different sources is often combined with frameworks and practices that have proven themselves in other fields, such as multi-criteria optimization models [83].

4.4 Approach

We describe several models used in various applications related to machine learning and decision support. We focus on models that are appropriate for learning from features that describe events that take place during a software development process.

One group of models that are of interest are supervised learning algorithms, which take numeric inputs and attempt to extract mathematical equations to approximate a target feature, which is also numeric. Frequently, these models come in the form of neural networks, where the equations are concrete but variables within the equations are set to numeric values that are adjusted during training.

We also look into regression analysis and Monte Carlo methods in order to discuss a method for inferring an expected value of effort for groups of tasks based on their similarities, known as analogy-based effort estimation (ABE).

Our contributions to the field of machine learning and pattern recognition span the use of score-based feature selection, as well as the evaluation of novel accuracy metrics for the models that we describe.

Our workflow is focused on data consolidation, reproducibility, feature extraction and feedback of annotated results based on our conceptual modeling research [59]. By acquiring data from multiple sources and selecting features from this combined database for inclusion into one data set, we are able to perform analysis of different aspects of the software development process, while remaining focused on providing useful predictive outcomes.

For privacy and confidentiality reasons, we applied one-way encryption on the data, which replaces certain information with pseudonyms, before feature extraction. This specifically concerns portions of data that can be used to uniquely identify persons and projects. This addresses security aspects of sensitive data in our collection mechanism. Still, we are able to cross-link personal or project-sensitive data from multiple source systems while data acquisition is underway, for example to count the number of developers in a team. Multiple encryption steps ensure that, at the location of analysis—in case this is outside of the organization's sphere of influence, for example a central research instance—there is no possibility of identifying specific projects or persons, while still allowing the results to be fed back properly.

An overview of our toolchain is depicted in Figure 4.2. A versatile approach simplifies the selection of different data sources for each project. It also makes it possible to deploy individual components in various environments, including Docker containers, virtual networks and enterprise clouds. We achieve secure communication between the project domain and the location of analysis using these distributed architectures while requiring few adjustments to the situation at hand. This generalized approach reduces a lot of the technical effort, when one introduces the method to multiple, diverse organizations with existing system architectures.

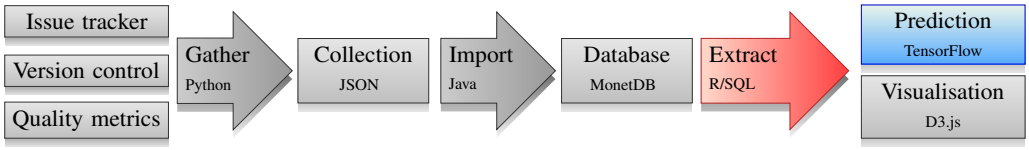


Figure 4.2: Overview of the Grip on Software pipeline, with the data analysis components highlighted.

A major part of the toolchain, which is described in more detail in Chapter 2, is the feature extraction, analysis and reporting. We additionally make the results of the analysis efforts insightful through the use of visualizations which are able to provide comparison overviews across multiple dimensions—such as time and project—as well as detailed information for current and future sprint situations through annotated predictions, as shown in Chapter 5.

4.4.1 Feature extraction

We collect records of the source information that describe entities, changes to them, or metrics related to those entities. These records have various types of information. We build a data set based on sprints and stories with descriptive numerical features. We collect these features from our MonetDB database [38], as explained in Chapter 3. To obtain the relevant sample records, we either perform feature extraction or make use of specially crafted definitions based on a combination of properties, possibly from multiple original sources.

The technical implementation of the feature extraction [h] uses SQL statements and integrated R toolkit programs [x]. This component allows us to select, filter and aggregate the data without losing track of the samples they refer to. This means we can provide detailed information of the data involved in computation of a metric.

Definitions are reused in multiple queries, assuring consistent results while allowing for adjustments for different data sources. This also helps solving some issues with noisy data that could otherwise lead to potential threats of validity in Section 4.7.1. This includes the actual end date of a sprint, which could be defined in multiple ways when stories remain open after the planned end of the sprint. We prefer the earliest date to avoid overlap that only exists digitally, e.g., when a team accidentally forgets to mark a story as done. In addition, we calculate the velocity of a sprint as described in Section 4.2.1, but instead of dividing the total number of story points by 3 we divide by the number of working days in these three sprints. This “daily velocity” formula cancels out differences in sprint length.

This approach allows us to describe and collect novel features of sprints. Meanwhile, we resolve inconsistencies in the source data without losing out on valuable details. We prepare

our data for use in our estimation and prediction models. The feature extraction component also performs feature selection and scoring. Results from some of our models can be used to repetitively try out different selections of features, extending sets of features with new ones to determine if the test set accuracies improve or not, thereby coming up with a minimal set of relevant features. We also use the RReliefF family of feature scores [84] to determine the estimators for a target label. We deliberately attempt to find a small set of most relevant features for our purposes to be able to explain more clearly how the models operate.

We select the following features for the sprint and backlog size estimations because we experimentally determine that they provide relevant, inherent information for further analysis in our prediction models:

1. Sprint:
   (a) Sequential index of the sprint within the project's or team's lifespan.
   (b) Number of weekdays during the sprint.
   (c) Other metadata of the sprint within the collection system, such as whether it is ongoing or if all data has been collected.
2. Team:
   (a) Size: number of people that made a change in the code, or on the issue tracker, during the sprint. In some cases, this is be calculated from other sources, such as an employment time registration system, instead.
   (b) Total number of sprints that the developers have made a change in before the sprint, as an aggregate measure of experience.
   (c) Number of new developers in the team that have not made a change before.
   (d) The overall sentiment of the team about the sprint as indicated during the retrospective.
3. Sprint planning:
   (a) The velocity, as calculated with the number of story points that were 'done' over the previous three sprints.
   (b) Sum of the story points planned for a sprint.
   (c) Sum of the story points that are 'done' during the sprint.
   (d) Mean number of people making a change on a story during the sprint.
   (e) Number of stories that are not closed as 'done'.
   (f) Number of changes to stories that are made later than the start of the sprint, specifically changes to priority or story points.
   (g) Number of stories that are added to the sprint after it started.
   (h) Number of stories that are dropped from a sprint, either placed back on the backlog, moved to a future sprint or removed.
   (i) Number of concurrent stories that are in progress at the same time.
   (j) Average number of days that the stories are in progress.

4. Backlog planning:
   (a) Number of items on the product backlog before the sprint starts.
   (b) Number of story points on the backlog.
   (c) Number of other issues on the backlog, such as epics which contain multiple stories.
   (d) An automated estimation of the number of story points on the entire backlog, based on the points that have been awarded scaled to the non-estimated points.
   (e) Mutations of the backlog, i.e., new stories added to it and redundant stories removed from it, including story point estimations.
5. Code version control:
   (a) Number of commits.
   (b) Average number of added lines, removed lines, total difference size and number of files affected.
   (c) Metrics on number of lines of source code, coverage of tests on source code and technical debt, which is a calculation on time to be spent on detected vulnerabilities or maintainability bugs.

The total number of resolved story points is considered to be the outcome of a sprint. In a sprint backlog prediction, we use this as a label, where models are trained based on the other features in order to predict this label for a future sprint. During classification, we instead target a binary label, which describes whether the sprint finished with no unresolved story points.

Some of our features are generated with predictive values based on calculations or estimations from separate analysis, e.g., product backlog sizes when not all stories have been given story points by the team.

Because the eventual value of a feature is unknown while a sprint is in progress, we either use value that it has at the start of the sprint or estimate the feature using the values as they are known at the end of earlier sprints. Similarly, we are also able to resolve missing data by applying a rolling operation that moves features that have unknowns in the latest sprint to each next sprint of the same project. Finally, a sensible solution is to never include sprints where not all features are knows as part of a data set during supervised learning.

The last method mentioned, which leaves out a few sprints from the data set, is most applicable to our situation. This is because we split our data set in training, test and validation sets while keeping the temporal aspect in place. Usually, machine learning algorithms are fed with random splits of the data set. For us, we need to avoid the circumstance that our models would train on data from sprints that took place later than the ones in the other two sets. A completely random split would violate the temporal constraints and test accuracies would not provide a good indicator of the predictive power of the model. Thus, we use the most recent sprints and future sprints as a final set where the labels are to be predicted for practical feedback to teams.

4.4.2 Data set

In our data set, we include information regarding projects and teams from two different Dutch non-profit organizations. Both organizations develop specific software applications for governmental agencies. The organizations employ their own variations of the SCRUM software development method, sometimes adding different roles such as a software delivery manager (SDM)

to streamline the process of introducing, refining, developing, presenting and delivering stories within the SCRUM teams. Aside from that, various project management frameworks—such as PRINCE2 [85]—and quality control approaches, are in use at the two organizations.

The two organizations, ICTU and Wigo4it, do not specifically share clients. They both develop software for different levels of government. The software realisation department at ICTU focuses on introducing and improving digital processes within ministries, executive agencies and authorities. ICTU is established by the government as a non-profit foundation. Wigo4it on the other hand is formed by four municipalities as a cooperative association and focuses on products that aid with social welfare projects.

The involved teams have worked on 39 projects at ICTU with different lifespans, with a total of 2643 sprints. Wigo4it has 21 teams working on various components related to each other, but each team has their own sprints, leading to a total of 534 sprints at this organization. After combining overlapping sprints of 60 distinct teams from both organizations and selecting recent sprints for a validation set, we have up to 2249 sprints for our training and test set. The raw data set** is made available through an API in ARFF format [XX]. Table 4.1 shows the actual dimensions of various entities in our database that form the basis of our samples and their features, after we performed our last data acquisition in December, 2021.

PROPERTY	ICTU	WIGO4IT	TOTAL
Projects/teams	39	21	60
Sprints	2643	534	3177
Issues	127257	63389	190646
Stories	20155	11333	31488
Code changes	534004	22295	556299

Table 4.1: Dimensions of the database.

In order to understand the features more deeply, we provide some additional details on the teams and their process. On average, development teams work in teams of about 6 developers, although outliers of smaller teams of 2 or 3 developers occur. Such smaller teams often work on smaller projects with a smaller backlog size. Larger backlogs often also appear in projects that have had a longer lifetime, indicated by a number of sprints of 50 or more. The average estimated size of the backlogs of all teams in these organizations is around 300 story points.

4.4.3 Models

We build four models for classification and estimation in total [i]: two models for predicting the probable effort during a future sprint, namely a neural network (NN) and analogy-based effort estimation (ABE), as well as two models for predicting the time before the backlog is finished, a linear regression (Lin) and a Monte Carlo (MC) simulation. In the latter case, we use a heuristic to scale the total size of the backlog based on the actual sizes of earlier stories, or epics that contain multiple related stories. Finally, we employ three scenarios where the predicted effort on completing tasks on the backlog is estimated: (1) by the velocity of previous sprints, (2) by the predicted effort of the future sprint or (3) by a combination of effort and other potential mutations.

**https://gros.liacs.nl/combined/prediction/api/v1/dataset. DOI: 10.5281/zenodo.10878529

We validate the models for recent sprints after we train and test accuracy on data from earlier sprints. In the case of the sprint result prediction, we separate our data set in training, test and validation sets. In the case of the backlog estimation, we compare our prediction by taking a temporal interval of the project up to a point in time, comparing it to the simulations and calculating differences. Some features used in the sprint prediction were rescaled to diminish the influence of differences of extrema of various features when training models.

Sprint result classification: Neural network (NN)

For the purpose of classification of sprints by risk of not finishing some stories, we use a multi-layered deep neural network based on TensorFlow [XI]. This network is able to receive multiple numeric inputs produced by features regarding one sprint. Internally, the neural network calculates weighted sums from these inputs, producing new values that are considered as inputs for the next layer, until the final layer is reached where a binary output is produced from a majority value. This binary output is the label, indicating whether the sprint will have unfinished stories in the end. We train the model in order to reduce the error with the outcomes of sprints in our training set.

In addition to a binary label, the network inherently demonstrates an inclination toward one label or another for a given sample. This inclination can be calculated from the numeric values in the final layer after feeding the sample through the network. This provides us with a risk value. Finally, the network exhibits a confidence score for each sample to indicate how well the network assumes its own result to be correct. We can use these metrics to report on the reliability and compare with other sample inputs.

By training the neural network with labeled sprints, we can adjust the weights within the network through back-propagation of real-valued errors, where each weight is downscaled or increased in order to reduce the error when the same sample would be provided again. The training process can report metrics on its accuracy and likelihood of a predicted classification being correct when we provide an unlabeled sprint's features.

We have used DNN models of various configurations and also considered other models, such as a smaller multilayer perceptron or a more heavy-weight Deep Belief Network, as well as other loss functions that steer the back-propagation process. Eventually, we found that a DNN model with three layers, having 100, 150 and 100 nodes each, performed well for classifying the binary label of sprints—whether they have unfinished stories—in our test set.

Sprint result estimation: Analogy-based effort estimation (ABE)

We use analogy-based effort estimation to gain more insight into how many story points could be feasible to do during a future sprint and which features play a prominent role for this prediction. The analogy-based effort estimation algorithm performs a search for a subset of features that produces the most similar top N sprints of each sample within a data set of sprints. The resulting subset is used for the search, which provides both the most similar sprints and an automated estimation based on the (weighted) average of the number of story points that were resolved during those N sprints.

Superficially, the model has similarities with a simple nearest-neighbor search or clustering approach. The strength is in its ability to select subsets of the provided features in order to find better similarities. This reduces the complexity needed to find similar sprints, enhancing the explainability of the model. Through pre-selection, using the ABE model on different combinations

of features, we find a minimal subset of the features that still provides accuracy and descriptiveness. This makes it easier to understand how the model determines which sprints are most similar to a sample, which features are involved and how the model calculates the estimation.

Backlog size estimation: Linear regression (Lin)

The sprint-based classification and estimation methods form a stepping stone toward backlog-scale estimation. After estimating potential velocity for future sprints and producing an estimation of the backlog including stories that have not been awarded points, we perform a linear regression of the total backlog size over time. Then we determine during which sprint the entire current backlog might be resolved. We consider another scenario where stories are added to the backlog, but also removed when they are found to be redundant.

Figure 4.3 shows an example graph of one project's backlog over time, including estimations for future sprints from various scenarios in the linear regression algorithm. The area within the graph which has a diagonally-hatched gray background displays estimations for future sprints: the orange curve outlines the scenario where only the team's velocity of finishing stories is considered, while the blue curve continues with a regression of all backlog mutations. The lighter lines show probability curves, where the likelihood of resolving all the stories on the backlog according to each scenario using the linear regression method is shown.

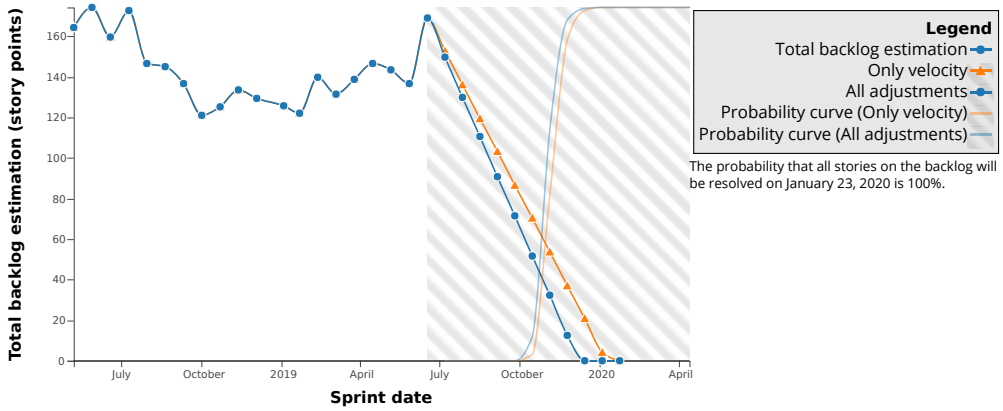


Figure 4.3: Graph of a project's backlog including estimated sizes using the linear regression model and multiple scenarios, where only changes in completed story points after each sprint (continued blue line with circles) or also other additions and removals to the backlog (orange line with triangles) are simulated, including probability curves (light lines).

Backlog size estimation: Monte Carlo (MC)

We likewise simulate the progression of the backlog using Monte Carlo simulations [20 ch. 1.1]. As with the linear regression algorithm, we again consider two scenarios, which are based upon selecting a random distribution of earlier velocities and mutation sizes, respectively. We use a Gamma distribution during the simulations so that the values from most recent sprints are

favored more than older sprints. We choose to apply this distribution because we assume that recent velocity that a team achieves in completing stories and the awarded story points are most indicative of the team's effort.

Each simulation run provides an estimation of backlog size changes for future sprint. By performing many runs, a complete simulation leads to a cumulative density function curve of the sprint in the future where the backlog is completed, indicating a likelihood for each sprint to be finished. In addition to linear or logistic curves to demonstrate the progression of the backlog in these simulated scenarios, we also extract the possible deviation of the average regression line and the extrema or conic contour of the Monte Carlo simulation.

4.5 Analysis strategy

We propose a number of experiments to assess the accuracy of our methods, which we further explain here. For the estimation of the number of points on the product backlog, we validate the simulated changes to the backlog based on the progression of earlier backlog sizes. Similar to splitting a data set, we take one third or two thirds of a project's lifespan from its inception as our training set. This approach preserves the temporal aspect of the data set and allows us to make estimations from the most recent sprint in the training set. We then compute deviations of the estimations compared to the actual values that were not included in the training set.

We wish to establish and understand the probability density curve of the distribution of possible outcomes for a backlog. Therefore, we estimate the range of possible values of the backlog progression using Monte Carlo simulations with 10000 runs. Each run takes random samples of the relevant factors that cause mutations on the backlog. These factors include (i) the velocity of resolving story points during each sprint, (ii) the number of stories introduced or dismissed on the backlog and (iii) an automated estimation of the number of points of the stories on the backlog that have not yet been estimated by the team. For the Monte Carlo method, we generate a distribution based on selected features from earlier sprints with preference to the most recent values while still allowing for variation by selecting values from earlier, but still recent, sprints at lower probability. We compare the distribution of estimated sizes from the simulation runs against a normal distribution to see if the chosen selection skews the eventual outcome.

Not only are these validations relevant for determining the accuracy, they also augment our data and provide context on the likelihood of when a backlog could reach a certain value. This metric can be a benefit to stakeholders when they assess whether a prediction for an individual project's backlog size seems reasonable.

Similarly, for the prediction of the number of story points that are likely to be finished during an upcoming sprint, we measure the accuracy of our models using widely-used error measures. In particular, the F1 score and related metrics are established in the field of machine learning, also when involving ensemble models and expert systems [86].

We also compare our results with earlier sprints through repetitive splitting of the data set during the training phase based on temporal information. This leads to many model runs using training and test sets that were partitioned at the start of each sprint. This means that we determine the performance of the model for each point in time, after training it with only the subset of data available up to that time. We explain our experiments with training set sizes in more detail in Section 4.6.1.

4.6 Results

The models for sprint classification, sprint effort estimation and backlog size estimation were based upon existing algorithms that have been in use in other fields, but include some novel adjustments and make use of newly selected features. Therefore, we validate their accuracy in estimating the labels that we have available from our data set.

We validate the proposed models based on validation sets, which are sampled from sprints which are characterized by features related to, amongst others, sprint velocity and backlog size. The validation sets are based on the most recent sprints in the case of sprint effort estimation, whereas the training and test sets are randomly split based on the remaining samples of sprints. For the backlog estimation simulations, the validation is based upon a subsequent set of sprint data from earlier portions of a project's lifespan, which were not used in the simulation's tuning.

4.6.1 Sprint classification and estimation

The F1 score of the sprint velocity prediction is 89.98% for the multi-layered neural network. This is an improvement over a baseline naive prediction by assuming that problems with resolving stories in the previous sprint cause a propagation to the next sprint, which is at a 73% accuracy. Similar results are achieved if the data set, consisting of sprints from projects of the two organizations, is rebalanced using stratification to avoid overfitting the neural network on too many samples of one label compared to the other.

For the analogy-based effort estimation, we find that the predicted number of story points resolved during a sprint for 88.6% of our samples from the first organization are within a margin of 25%, so the Pred measure [87] in this case is $\text{Pred}(.25) = 88.6\%$. We note that this metric is based on a relative error measure. It is possible that our ABE model does not function well for a data set with more or different data. This can be seen when we also validate the samples from the second organization, which causes the reported metric to drop dramatically to 68.2%. We therefore consider this estimation algorithm to be less stable and more influenced by bias and noise than the neural network is. Still, the ABE model provides helpful intra-organizational details that allow tracing back how the model estimated the number of story points that can be finished within a sprint. This increases the explainability of the model. Keeping the ABE model instances separate for each organization means that the reported analogous sprints remain more familiar to the development team as well.

As an additional experiment for the neural network, we consider what the effect of having a different training set size would be. We do this by splitting the training and test set based on time, such that we add roughly ten sprints into training set per experiment. Each time, we completely restart training the model, to avoid bias or other influences caused by providing the training set in a consecutive order. Thus, while the training set is ensured to not contain time gaps in each run, the training epochs behave independently.

The F1 scores of each training run on the neural network, where the score is based on the remainder test set, are shown in Figure 4.4. We observe that after an initial decent start, there are slight dips in the F1 score, after which the trained model seems to perform better and more consistent when a larger portion of the available data set is provided for training. Any greater proportion in favor of the training set would make the test set too small for an unbiased accuracy score. Similarly, we do not consider a training set with 100 sample sprints or fewer as this would only seem to lead to a nontuned, overfitting model.

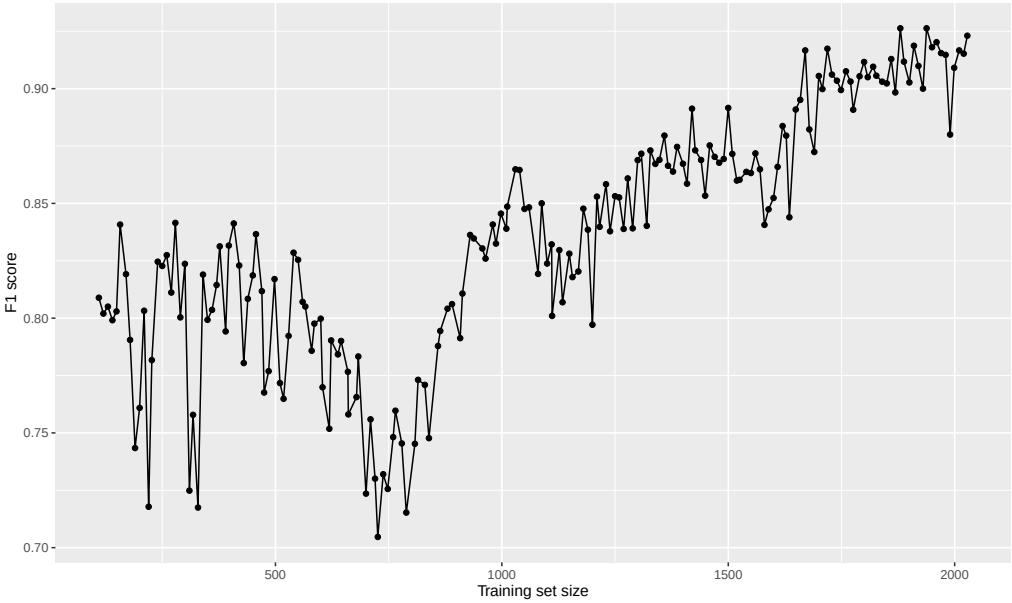


Figure 4.4: F1 score of the neural network when trained with different training set sizes.

4.6.2 Backlog size estimation

We validate the backlog sizes by comparing deviations as well as cross-validating the models. In separate runs, we provide one third and two thirds of each project's lifespan and let the algorithm simulate backlog sizes for another third of the sprints, or when the estimated backlog size reaches zero—whichever comes first. We then calculate the total change between the backlog size in each sprint that the algorithm has simulated and the actual sizes of these sprints. We report on the mean difference of the backlog progression—which encompasses all differences between the simulation at each sprint and the estimated progression of backlog sizes—and also the deviation from this mean, which includes projects being under- and overestimated.

Therefore, our error measure for the results of these algorithms includes every difference at each included sprint from the point of our data set split until the end of the simulation, including those sprints where the simulation takes longer than the project did or vice versa. This means that a simulation is penalized for estimating a longer progression, a shorter one and for differing from the backlog sizes in our remaining data set. This may lead to deviations across projects, especially those projects with lots of planned work which often have unforeseen scope changes. We provide these as raw error measures rather than performing normalization, as we consider that the impact of an estimation error is also greater for long-running projects with many stories on the backlog.

Using one third of the project's lifespan for a velocity-based linear regression, the mean difference between the predicted sprints and the actual backlog progression is 261.4 points lower ($\pm$316.5 points). If we use two thirds of backlogs, the error is -279.2 ± 355.1 points.

For the validation of the scenario where we consider mutations on the backlog as well, the results are similar (-274.1 ± 326.8 points versus -285.3 ± 359.1 points). We observe these large deviations across all projects. At these stages of development, projects vary in their progress as

well as the backlog size, which change rapidly over time. Usually, the backlog does not decrease swiftly, however, since the scope of the project often changes at these stages.

The Monte Carlo simulation shows an improvement over the linear simulation when validating the scenarios using data sets of initial sprints. The mean error that each run in the simulation takes across all backlogs, when using the first third of the project as sample data, is -65.0 ± 176.0 points for the scenario using only velocity and 2.7 ± 172.3 for the scenario that considers backlog mutations. When two thirds of a project's backlog progression are known, then the mean error slightly worsens to -109.1 ± 258.2 points for the velocity scenario and -23.2 ± 211.1 for the simulations where story points on the backlog would be added and removed.

The results indicate that the Monte Carlo model becomes somewhat less precise when a larger fraction of the data set is made available for the selection process. This is in contrast to the linear regression algorithm, which is consistently off by about the same margins. The deviation of the MC model is quite low, despite the use of a simulation where the probability density functions cover a large range of possible values. Using different factors that influence the backlog size, we achieve the most realistic scenario.

We notice that sudden, unexpected changes to the backlog, such as a large influx of new stories, easily cause our simulations to underestimate the backlog size. Such situations are likely difficult to predict, especially given that we only use earlier data from projects to simulate typical situations, either through average regression or generalized normal distributions. All validation results for the backlog predictions are summarized in Table 4.2.

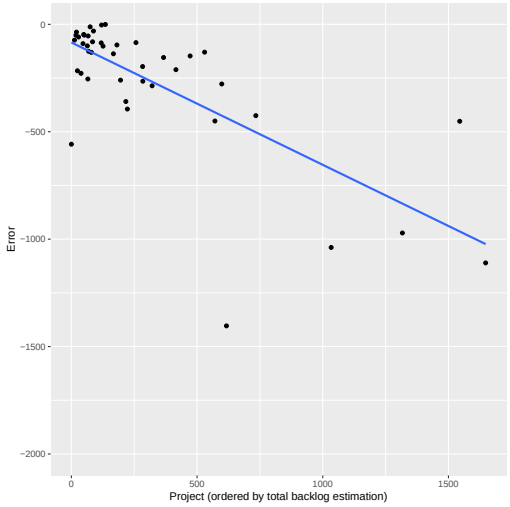
SIMULATION	SCENARIO	ONE THIRD	TWO THIRDS
Linear regression (Lin)	Velocity	-261.4 ± 316.5	-279.2 ± 355.1
Linear regression (Lin)	Mutation	-274.1 ± 326.8	-285.3 ± 359.1
Monte Carlo (MC)	Velocity	-65.0 ± 176.0	-109.1 ± 258.2
Monte Carlo (MC)	Mutation	2.7 ± 172.3	-23.2 ± 211.1

Table 4.2: Summarized validation results for backlog size prediction across 35 projects and teams using our incremental error measure. Closer to zero is better, a lower deviation is as well. Values are based on the distribution spread of projects of different backlog sizes, with no normalization for this property.

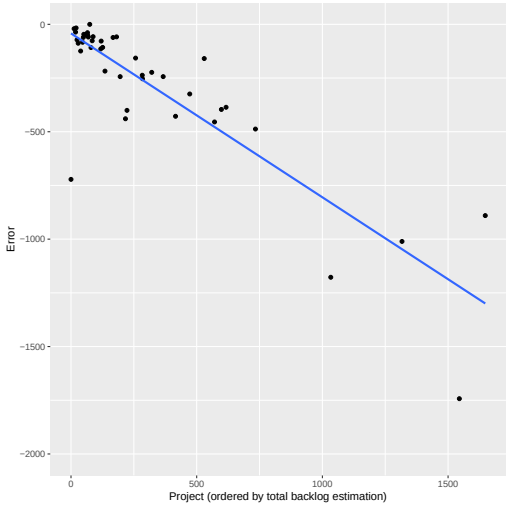
The validation error results for the backlog size predictions of individual teams and projects in each simulation can be found in Figures 4.5 and 4.6. In the plots of these figures, the projects and teams which had enough sprints with backlog data (35 total) are points which are ordered on the x axis by their total backlog size. This backlog size includes automated estimations for user stories that have not been awarded story points by the team. Meanwhile, the validation errors are shown on the y axis. Most of the differences of each project show showing that the models estimate a lower backlog size across all the simulated sprint compared to the actual sprints. This is usually caused by underestimating the backlog at each sprint, which also leads to the simulation reaching an end earlier with an empty projected backlog, while the project itself actually continued on with the backlog often remaining at a reasonable size.

Furthermore, we observe that larger backlogs receive an increasingly lower predicted size. In absolute terms, the validation errors increase with larger backlogs to be predicted. There is a strong tendency by the models to assume that the backlog is done sooner than in reality. For

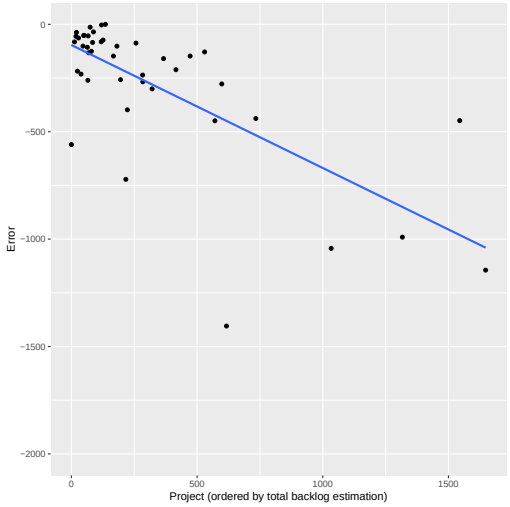
Monte Carlo, the errors show less deviation than the linear regression scenarios, but it is still biased, especially for projects with many stories on the backlog. The mutation scenario has the fewest outliers when using one third of the backlogs for seeding the distribution, but only in the case of Monte Carlo simulations.



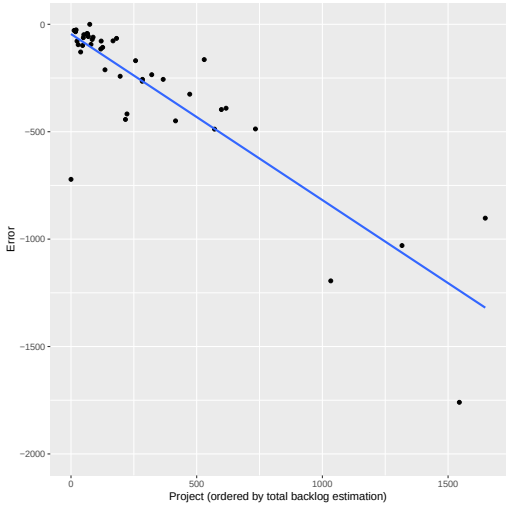
(a) Linear velocity scenario, one third ($r^2 = 0.49$)



(b) Linear velocity scenario, two thirds ($r^2 = 0.76$)

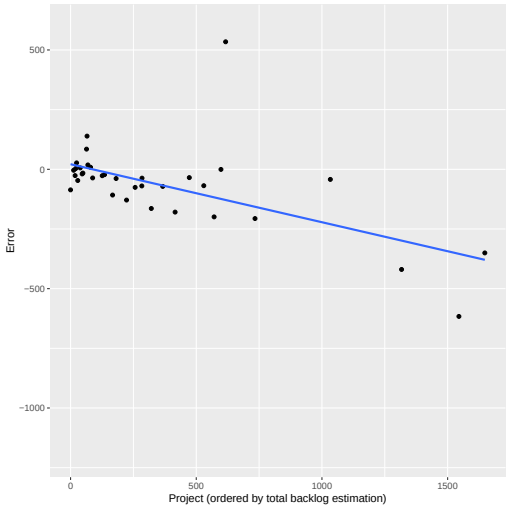


(c) Linear mutation scenario, one third ($r^2 = 0.47$)

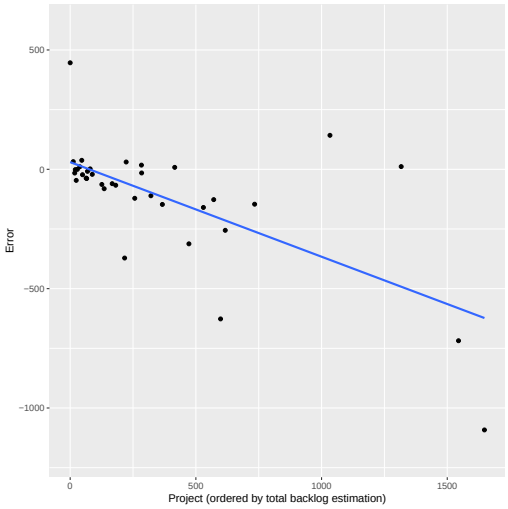


(d) Linear mutation scenario, two thirds ($r^2 = 0.79$)

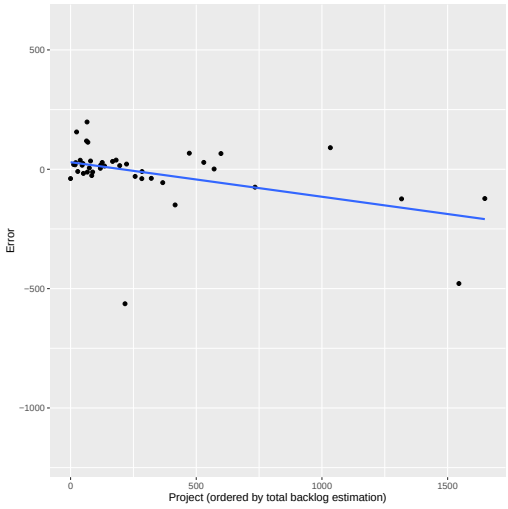
Figure 4.5: Differences in individual project/team backlog size estimations using linear regression, along with a linear trend of the validation error along the estimated most recent sizes of each backlog of the projects and teams (35 total). The reported coefficient of determination (lower is better) helps with comparing the plots.



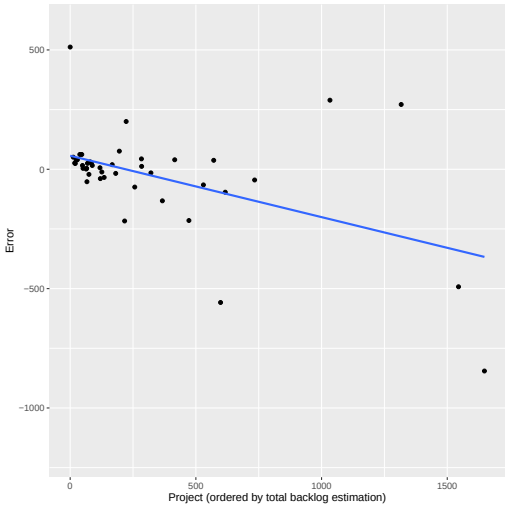
(a) MC velocity scenario, one third ($r^2 = 0.36$)



(b) MC velocity scenario, two thirds ($r^2 = 0.47$)



(c) MC mutation scenario, one third ($r^2 = 0.11$)



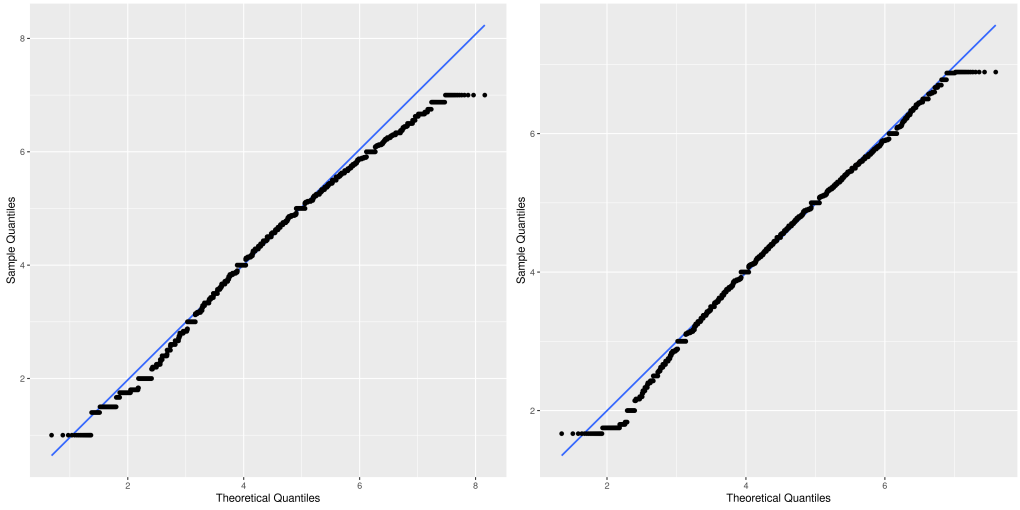
(d) MC mutation scenario, two thirds ($r^2 = 0.29$)

Figure 4.6: Differences in individual project/team backlog size estimations using Monte Carlo simulation, along with a linear trend of the validation error along the estimated most recent sizes of each backlog of the projects and teams (35 total). The reported coefficient of determination (lower is better) helps with comparing the plots.

There appears to be no benefit in taking more of a backlog for estimating the outcome, as most of the projects remain at a stable backlog size during this time, aside from scope changes. Teams seem to work away enough of the backlog to reach such a stability. In fact, there is potential that an even earlier, smaller backlog is already enough input for our algorithms. However, both the

linear regression and the Monte Carlo simulation favor projecting an earlier end time for finishing the backlog, so caution should be taken. Still, in Section 4.7.2, we show that a small subset of a backlog for a newly started project leads to simulations with insights for the team.

We also investigate what kind of distribution the Monte Carlo simulation produces, in an effort to verify whether we implemented the simulation and parameters correctly. We find that the resulting distribution has a large number of ties (in this case, the same outcome of predicted end date) because the steps of the simulation are based upon future sprints, which is limited in the distribution that we achieve. Still, a quantile-quantile plot, depicted in Figure 4.7, indicates that overall the distribution of expected number of remaining sprints is similar to that of a normal distribution [88]. It is slightly offset from the origin due to the nature that the expected moment of finishing a backlog is shifted toward later sprints. We also observe that the scenario where all changes to the backlog are taken in consideration more closely matches the normal trend, compared to the scenario where only sprint velocity is simulated. This helps validating that the additional factors do not degrade the expected distribution.



(a) Monte Carlo velocity scenario

(b) Monte Carlo mutation scenario

Figure 4.7: Aggregate quantile-quantile plots of the resulting distribution of sprints in which a backlog is completed, along with the normal trend.

4.7 Conclusions

This thesis presents improvements for the planning aspect of SCRUM software development processes. We introduce established and state-of-the-art algorithms from the fields of statistical analysis and machine learning into this domain. We provide predicted outcomes to development teams and other involved people, displaying promising applications of artificial intelligence in Agile planning. The results show that our methods are feasible for predicting sprint backlog and product backlog sizes, measured in story points. Thus, our predictions are usable for short and

long time spans of a software development project and form part of the answers to our research questions, which are further addressed in Chapter 6.

We create data set of features derived from metrics that measure various aspects of the development process, with scoring using external and internal models to improve the relevance of the selected subset. The models that we propose are classification and estimation algorithms for sprint planning and product backlog sizing. The two algorithms for the former objective achieve a higher accuracy than naive methods and provide additional insights such as risk analysis and most prominent features. A second set of algorithms predict end dates and provide experimentally determined likelihoods for intermediate backlog sizes. Such properties make it possible to validate these algorithms, but also help establish integrated visualization tools that empower SCRUM teams, in particular roles such as Product Owner and Scrum Master, to make informed decisions. The backlog size estimation algorithms are able to simulate multiple scenarios that can be adjusted to more realistic situations based on individual project standards. Meanwhile, we focus on generalization and explainability by including multiple organizations and teams in our data set and focusing on descriptive features and models. This approach highlights differences and similarities between situations in applications of Agile software development processes.

4.7.1 Threats to validity

During our analysis of the collected data, we make some assumptions regarding relevance of data that could influence the outcome. Additionally, the limitations of scope of the data could mean that our models and results are not directly usable as a generalized method within the field of Agile software development. Our data set has a focus on SCRUM software development at governmental organizations. However, we argue that the approach is reusable in other contexts due to the large number and diversity of the involved projects.

We observe that the number of story points awarded to a story by the development team fluctuates during the lifespan of a software development project. This change in value may be caused by a shift in the focus of the project, such as the complexity of the work being done. Sometimes, teams adjust how the points scale is used or which reference story they base it on. Other influences are composition changes within the team or focus changes based on wishes and requirements from the client or users. We notice however that teams become more experienced with awarding points to the stories they work on during the lifespan of the project and therefore their expert estimations gradually become more associated to the actual effort. By preferring the data from recent sprints as input to our classification and estimation models, we reduce the impact of events from long ago. This might impact our testing and validation scores, given that we only provide data from early sprints during training. Our models should be versatile enough to work with both a small data set with only initial development work as well as large, noisy data sets.

Individual teams working on separate projects, even within the same organization, have different methods and practices when using digital tools to aid them in tracking their progress. Some projects require the use of additional types of sprints in order to separate work in sub-teams. These situations are noticed during analysis of results and discussions with key roles of the teams. We make changes to the experimental approach to either include these disparate situations or to filter out certain subprojects. The solution depends on how well those projects fit along with the SCRUM framework and in consultation with said people. In general, if these disparate sprints use story points, commitment to work being done and proper flow of input on the backlog, we consider them relevant for inclusion.

We experiment with several methods for estimating effort, in the context of a product backlog that is partially estimated through expert judgment using story points. Our samples of metrics and features that the algorithms use to provide estimations should be representative of the process that they encode [89], but the estimation process in particular is still dependent on the quality of the input data. Curation and feature scoring help with improving these aspects of validity, but this remains a constant improvement effort.

4.7.2 Proposed additions

In terms of further research, the annotated predictions of sprint velocity may be useful to apply in other situations. In order to determine how much total work can be done in a sprint, we could extend our predictive model to other types of issues, such as bugs or technical debt. Additionally, risk analysis of sprint progress through classification could cover fine-grained early-warning of changes in other metrics such as clarity of stories, team sentiment, code complexity and coverage of tests. Other tools for text analysis and static code analysis could play a role in this, as well as deeper analysis of links between changes in code and stories in a sprint for fault detection.

We also consider extensions to the estimation models for backlog sizes. Other types of simulations, for example through the use of Long Short-Term Memory (LSTM) as well as other scenarios may further augment the results. By filtering on portions of the backlog with on-line reruns of simulations, better could be made more swiftly and accurately. As an example of further work, Figure 4.8 shows a prediction for one specific milestone with a subset of stories on the backlog, which were selected to be finished before a certain milestone would be reached. This aids the team in determining if this medium-length plan would be a viable approach.

As such, our proposed methods augment and support the SCRUM framework and potentially other Agile software development methods so that more stability and predictability is introduced to the process where it is helpful, while retaining the strengths of the process in the areas of work selection and commitment.

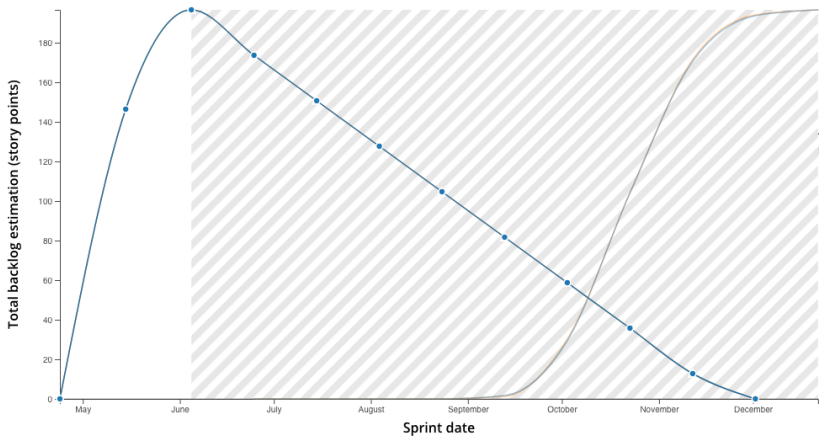


Figure 4.8: Estimation for a subset of a backlog related to a specific milestone version, showing the known and predicted backlog size (with hatch-filled background) at each sprint point in blue and the probability density curve in gray.

Chapter 5

Information visualization

GROS Vis: Evaluating the dissemination of software development patterns through information visualization

Portions of this chapter are also published in the following article:

- Leon Helwerda, Cas H.J. Dekkers, Walter A. Kosters and Fons J. Verbeek. “Information visualization in analytical decision support and ecosystem management in agile processes”, 2024. Pending submission.

Abstract of Chapter 5

Introduction: Software development relies on a set of systems which often produce many metrics which go unnoticed, but are helpful to learn what takes place during the development process. We have analyzed patterns and produced predictions and estimations for SCRUM projects from two organizations (ICTU and Wigo4it), which we relay back to the involved stakeholders in a visual format.

Research questions: How can we effectively introduce visual representations of results and recommendations from our analysis of data collected from the SCRUM development process to the involved parties?

Aims: Important design principles are familiarity of the data format, similarity between teams, simplification of existing processes, new insights and novelty. The design requirements for the visualizations are integration within the ecosystem, new or improved workflows, focus on relevance with details upon request and self-descriptiveness.

Visualizations: The result of our development is an integrated dashboard with access to eight information visualizations, namely a sprint report, results from predictions, a timeline, a leaderboard ranking system, a collaboration network graph, a story process flow chart, a heat map calendar and a platform status monitor. The first four are primarily in mind of analytical decision support, while the others have an organizational ecosystem management theme. Additionally, three visualizations are provided as plugins for the issue tracker, which render the product backlog with focus on burndowns, status progressions and relationships between tasks.

Evaluation: We utilize automated tests, objective measurements and user surveys to determine that the visualizations adhere to the principles and goals after iterative adjustments based on feedback. We demonstrate that it is worthwhile to provide intermediate analyses and results from predictive models regarding SCRUM software development processes beyond our case study.

5.1 Preamble

A major objective of our research into understanding the characteristics and outcomes of the SCRUM software development framework is to provide our results to those that benefit the most from it. Specifically, it is relevant to keep in mind the different roles and technical expertise of the people in the development team, those that assist the team, as well as anyone with a legitimate interest in the outcome of the process, often referred to as the *stakeholders*.

In an Agile software development framework like SCRUM, the core process revolves around team interaction and collaboration in order to incrementally improve working software and respond to changes [3]. With these goals in mind, it is important to arrange delivery of research results in such a way that the core points are boosted rather than hindered.

Simply providing results in a raw format would be cumbersome to use. Such an approach would not be likely to lead to widespread adoption. A framework or system that makes it more clear how to interpret the results is more helpful. We consider this framework in the context of human-computer interaction (HCI) as it particularly provides a means to bridge the gap between computer data and a person's knowledge. The person is then able to make informed decisions that cause changes in real life, which then reflects in the results as well. As such, a feedback loop occurs where an improved software development process has a positive effect on the results, through means of a simplified visual representation of the process. The framework therefore becomes a part of the process, embedding the selected data as it can now be discovered by the involved parties.

In order to interact with such a representation, it first needs to be rendered within a framework. To visually reproduce different types of data, a visualization is often an intuitive method. In continuation of our data acquisition and analysis of predictive patterns, particularly in Chapter 4, we study the choices to make when designing information visualizations for the purpose of supporting people in a SCRUM software development organization. The main goal is to help them gain understanding of the process and to indicate improvements and impediments. For the information visualization part of the Grip on Software project, we consider the following research question, split up into sub-questions:

RQ3 How can we effectively introduce visual representations of results and recommendations from our analysis of data collected from the SCRUM development process to the involved parties?

- **RQ3a** Which concepts and goals are relevant when designing information visualizations for patterns analyzed from a software development process?
- **RQ3b** How do we integrate results from predictive models within existing development practices?
- **RQ3c** Which effects do the introduction of results from predictive models and other intermediate analyses have on the development process, validated across multiple ongoing projects?
- **RQ3d** What is the overall assessment of the proposed information visualizations, considering automated measurements for usability and the adoption according to interviews and surveys?

In Section 5.2, we discuss our motivation for the use of information visualizations in software development more thoroughly. Section 5.3 discusses related work and main concepts for information visualization based on a literature study. The general structure for building our visualizations is described in Section 5.4, followed by descriptions of each visualization based on this flow. We evaluate the produced visualizations in Section 5.8, based on user interviews and automated measurements, including a summary of our findings in the conclusion in Section 5.8.3.

5.2 Purpose

There is a growing need for information visualization within software development organizations as a framework for comprehensive summary of the process. When the organization maintains an ecosystem based around the SCRUM framework, there is a constant generation of data when user stories are created and refined, when tasks are prioritized and given metadata, when code is changed and tested, when metrics are collected on software quality and when a product increment is provided for user acceptance. In addition, considering we have a cyclic software development framework, each step is performed often, resulting in even more data. Development teams are sometimes small, but an entire organization that works on different components or projects in a similar way leads to a large bulk of information. This data needs further inspection to learn from impediments and other problems, to change configurations of systems or to improve how the teams use them in conjunction with the development framework.

However, involved parties within the organization are usually not tasked to scrutinize the entire stream of data as their main focus. Some relevant data is forgotten or even lost if it is not spotted and tracked in some way. An automated system could help with summarizing the important points while providing methods to delve deeper. A rapid process requires an uncomplicated approach to access an analysis result using large numbers of metrics. Data should be shown in a visually appealing representation that would be hard to understand if it was only in raw form. Different people and roles across the organization have conflicting interests when it comes to detail and overview, but these should be combined rather than canceling each other out.

It is important to keep certain goals in mind when designing visualizations for use within software development. Specifically, there may be ideas that are provided by team members, indicating that existing systems and practices fall short in their usual workflow. These ideas are the basis for a new visualization, which can be seen as a goal in itself. However, a newly developed visualization should not just fill in this gap, but provide a bridge from the existing ecosystem, making it intuitive to use. A familiar environment should be created that allows seeing both the source of the data as well as novel viewpoints and levels of detail.

The main purposes of information visualization in SCRUM software development are:

- To provide temporal data of process dynamics in a familiar format to the stakeholders, for example by aggregating or splitting the achieved effort over sprints through visual indicators.
- To bring together indicators that describe similar processes and their slight differences between teams within an organization, which operate independently but have inherent associations due to a common work ethic.
- To simplify existing, mundane processes that rely on manual labor that distract from—rather than contribute to—collaboration within teams, such as writing reports with recent metrics.

- To gain insight into factors that indicate successful situations such as a user story being "done" on time, as well as negative outcomes, for example by highlighting patterns or providing predictions for estimations including their basis.
- To incite new ways of thinking about the framework through creative use of combinations between data from different sources that display the process in an original way, for example with visual relations across different abstractions of work (epic, user story, task, code).

5.3 Relevant concepts

There exist different types of visualizations for various contexts, dimensions of data and intended applications. Data-oriented visualizations often come in the form of diagrams, charts, graphs, 3D renderings and so on [90]. Many of these types of visualizations have established their use within many research fields and applications.

Another way to group visualizations is by determining what kind of data is being shown. For statistical data, the visualization should show data in an undistorted manner with the specific purpose of supporting a conjecture. For scientific data, the visualization's *coordinate system* or dimensionality is usually predetermined by the input. For other information, transformations are necessary in order to understand the meaning, to find underlying patterns or to differentiate between typical situations and anomalous noise coming from outliers. Each type of data or information needs some action to allow the user to understand and include in their knowledge [91].

A third way to categorize visualizations is by their intended goal, namely (a) exploratory visualizations that allow the user to search through data efficiently, (b) confirmatory visualizations that provide controls to test a hypothesis and (c) explanatory visualizations that present data in a preselected, validated manner [92, 93]. A proper categorization of a visualization design includes an assessment of what the intended user can do, what they want to do and how they can do this most effectively [94 ch. 2.3]. By determining beforehand what kind of visualization best matches a task through this goal-based approach, sometimes called *cognitive fit* [95], the resulting visualization may increase a user's performance in solving their problems [96].

The freedom we have when rendering in information visualization (InfoVis) is also affected by the type of data. Often, several choices can be made to transform the data or to find new relations between data points. If the data is left as is, the visualization is less appealing, but when too many transformations are applied, then the data no longer reflects the situation that it originates from. A visualization has to avoid misleading the user. When there is no option to show details and outliers, then a user can easily feel deceived. On the other hand, dumping all the raw data does not spark attention. A visualization should tell a compelling story which supports the existing narrative in the right way [97], by aligning as close as possible to reality.

The aim of information visualization should be clearly defined. In a transparent design process, choices for which data to use should be based on the usefulness of the visualization within the context it is deployed in. For software development, determining which development project performs better than others might be achieved with, e.g., comparison diagrams, but such a method should explicitly state on what grounds a metric is selected. Aside from caution with regards to project-sensitive data, a visualization should also adhere to privacy aspects, where the availability of personal information is limited in such a way that they can only be traced back to an individual if the user of the visualization already knew that individual. The focus is on the greater picture rather than the singular contributors.

With regards to the application of information visualization within software development, there has been substantial research into knowledge sharing within Agile teams [98, 99]. Visualization can also be used as a means for monitoring a SCRUM process, for example, by selecting metrics based on preferences of the developers [100]. Others focus more on visualization in a broader sense of collaboration and management, where it is improves the quality of communication in groups of people with a responsibility for a project when there may otherwise be an information overload [101].

There are more concepts that should be applied universally in information visualization. Icons can help with consistency and familiarity, by indicating the same meaning of a type of data, a selection control, a data source, etc., across visualizations and parts of them. We use similar elements in an interface control to indicate that an action can be taken to expand or collapse. When an icon allows the user to perform an action, a tooltip can describe what it does succinctly.

The use of *glyphs*, abstract geometric figures or symbols, as well as colors also helps in visual appeal and swift assessment of the data. A glyph may indicate a scale of the data by changing its size or shape. A symbol could also indicate how the user can interact with the user interface (UI) in order to alter the rendering. Such an interaction need not necessarily meet a user's goal, but should satisfy a certain inquiry [102]. Data can also be presented next to a common glyph or even within a shape such as a box or a rounded graph node connected with arrows or lines to other nodes, which makes it tie in with the rest of the structure [103 ch. 5]. Visualizations can employ different techniques to make patterns easier to perceive by the human mind [104 ch. 2]. Colors can differentiate or highlight certain types of data. However, care should be taken to choose a color scheme that has enough contrast and avoids the use of hues that appear the same to people with color blindness [105]. Otherwise, any meaning given to this color is to be provided in another manner as well, such as using different shapes for colored points in a chart.

All these principles should be taken into consideration when designing and tweaking an information visualization. Sometimes, additional features are not easily usable for everyone. If this is not a concern because only a limited number of people would use it, then one can question if it should be added in the first place. Some visualizations cover a broad audience, which means some elements are used less frequently and can be presented less prominently, aiming at those that do feel familiar with advanced controls. In short, a combination of usability and utility of a visualization is often necessary to achieve the aforementioned goals [106].

There exists a general flow for designing information visualizations [107, 108 ch. 1]. First, the selection of data that is relevant to show within a visualization is determined. It may be necessary to reduce complexity more than narrowing the scope or limiting earlier data. We perform *transformations* on the data to aggregate multiple points, to select which attributes are relevant for the information visualization and to determine a coordinate system. Based on this selection of data, we are able to define relationships between different entities or data points. The relationships provide a scheme for a *visual form*, such as a graph structure, tabular data or other information where it is not yet set in stone how it is rendered precisely. The *rendering* is only considered in the next step, where a mapping determines how the abstract representation is placed at precisely specified coordinates. Combined with color schemes and related symbols, this specification produces a complete *view* of the data. Finally, the visualization is given *interaction* by assigning actions to controls and motions, also known as *gestures*, that make additional controls show up or allow the user to pick a subselection of the data. A new selection adjusts how the other steps take place and thus allows a new rendering to appear. This general flow for an information visualization is depicted in Figure 5.1.

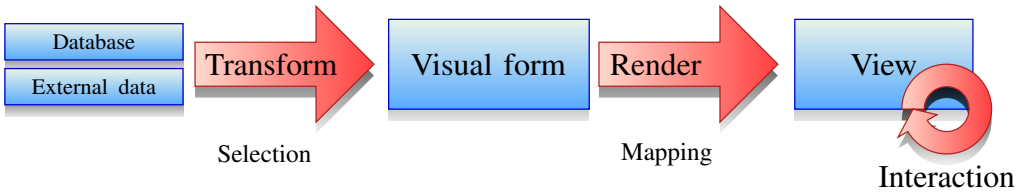


Figure 5.1: General flow of constructing an information visualization structure. Blue blocks are data formats, red arrows are translation steps.

5.4 Dashboard framework

We develop multiple information visualizations and make them available in an integrated dashboard. This way, we provide access to (intermediate) results of experiments and promote new viewpoints for stakeholders related to the projects that are included within the data set. One principle that helps with adopting results and recommendations within the usual development workflow is to make them feel familiar to the users, by highlighting factors that are commonly agreed upon as being relevant. We wish to provide regular updates of the results provided in the visualizations and to encourage reuse of the state-of-the-art analysis for estimating effort within sprints and backlogs.

The dashboard offers an integrated approach to information visualizations for the Grip on Software research. Different types of data can be rendered and interacted with. The visualizations have controls to filter and select data as well as zoom in on details of a visual representation, particularly following the visual information-seeking mantra [109]. From a usability perspective, the influence that an interaction has should be unsurprising and obvious. The controls that allow these actions and gestures should be easy to use and work similarly across different visualizations in the dashboard. That way, the user can learn to select data without losing this acquired skill, which could otherwise happen when another visualization presents such an interaction control in an alternative, confusing way [110 ch. 2.2].

Based on the concepts and approaches described in Section 5.3, we build novel information visualizations that focus on different parts of the Grip on Software data and provide interactions that are useful for various roles within the software development organization. The visualizations include reporting formats such as tables and charts, metrics from predictions, timelines, ranking systems with statistical plots, network graphs of the organizational process, flow diagrams, calendars and status indicator dashboards.

We integrate these visualizations in the dashboard. Each of them provides access to the others through a common menu, which is shown at the top of the screen. On a desktop or larger screen, these menus can be expanded by hovering over a category of visualizations, while on a smaller, mobile screen the menu is expanded through a “hamburger” control, a typical UI element to optimize the use of screen space. The menu provides an option to show the visualization in full screen for presentation purposes during meetings. The language can be switched between English and Dutch from within the menu as well.

The dashboard includes introductions for each of the visualizations as well as links to open them, which is deployed to a web server [j]. All the visualizations work within modern web browsers. Figure 5.2 displays the overall dashboard.

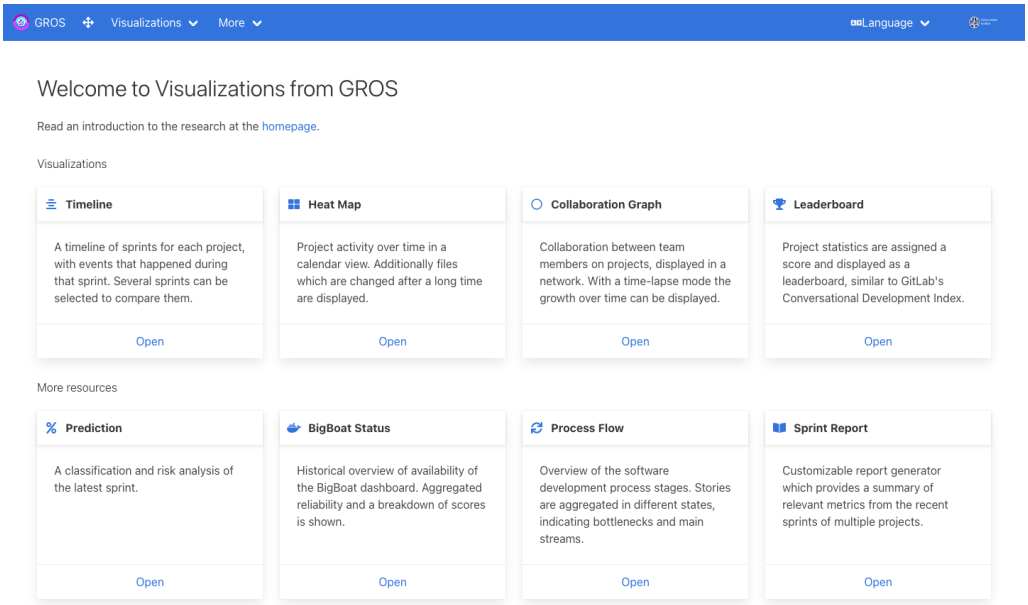


Figure 5.2: Dashboard of Grip on Software, providing access to eight information visualizations.

The visualizations share a set of design principles, detailing requirements that should be met:

- We aim to fulfill one or more of the concrete purposes and goals mentioned in Section 5.2. Each visualization focuses more on certain objectives depending on its categorization.
- The visualization should integrate well with the other visualizations, existing SCRUM practices and the ecosystem. Access is provided through the dashboard and the visualization is also accessible from other systems.
- The main goal is to boost the delivery of the product by making relevant data accessible to those who are involved in the process—our stakeholders—by simplifying and speeding up existing workflows.
- The main feature of the visualization should be the elegance of simplicity, by highlighting the most relevant data/patterns, instead of what's under the hood, i.e., the raw data stream. A visualization can provide a means to zoom into the data in order to view more details.
- The visualizations should use clear, human-readable descriptions of the data being displayed, available in languages common to the work environment.

From a technical viewpoint, we implement each information visualization in a similar manner. The rendering and interaction use HTML templates for structural page layout, CSS presentational stylesheets based on the Bulma framework [XXI] and JavaScript which retrieves the data from JSON endpoints, transforms the structures, provides effects and handles events for triggering actions. Specifically, we use D3.js [111] to bind data to HTML elements in a data-driven document,

render the elements based on context and add transitions for animations. We create reusable fragments [k] for a shared UI, enabling localization of the entire page, a navigation bar, a selection control for switching between teams, projects or organizations and finally a spinning wheel that can be shown while the page is rendering with newly requested data.

In the following sections, we elaborate on the construction of more than ten information visualizations according to the flow in Figure 5.1, roughly corresponding to the final component of the Grip on Software pipeline in Figure 5.3. We group the visualizations in three categories based on their data use and intentions: analytical decision support in Section 5.5, ecosystem management in Section 5.6 and novel product backlog visualization in Section 5.7. This third category includes visualizations that are meant to be integrated with an issue tracker like Jira [i], and are thus separate from the dashboard itself.

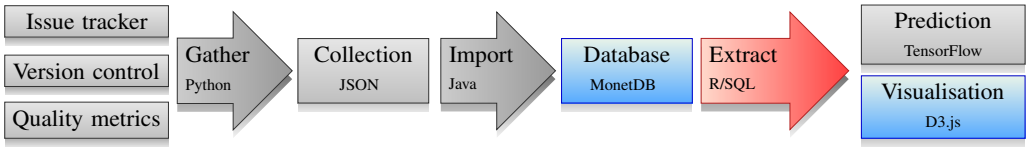


Figure 5.3: Overview of the Grip on Software pipeline, with the components related to information visualization highlighted.

In each subsection, we introduce the visualization by means of a motivation. Following that, we describe the input data, rendering steps and interaction functionality in detail, including figures with screenshots. As the interactions are not easily showcased using static figures, we use, in addition, QR codes to videos of interactions. As components of the GROS pipeline, the code of each visualization is publicly available as well. Table 5.1 provides an overview of the subsections, figures of the QR codes and references to code repositories provided in Appendix A for each information visualization.

Visualization	Part	QR codes	Repository
Sprint report	Section 5.5.1	Figure 5.11	[l]
Prediction results	Section 5.5.2	Figure 5.13	[m]
Timeline	Section 5.5.3	Figure 5.15	[n]
Leaderboard	Section 5.5.4	Figure 5.17	[o]
Collaboration graph	Section 5.6.1	Figure 5.19	[p]
Process flow	Section 5.6.2	Figure 5.21	[q]
Heat map	Section 5.6.3	Figure 5.23	[r]
Platform status	Section 5.6.4	Figure 5.25	[s]
Product backlog burndown chart	Section 5.7.1		[t]
Product backlog progression chart	Section 5.7.2	(Jira plugins)	[u]
Product backlog relationship chart	Section 5.7.3		[v]

Table 5.1: Information visualizations described in their sections as part of this chapter, the figures with QR codes for the demonstration videos as well as the code repositories of each visualization.

5.5 Visualizations for analytical decision support

In this section, we discuss visualizations which aim to help team members and other stakeholders with making decisions regarding SCRUM sprint progress, among others. The focus here is to provide results from the analysis done within the Grip on Software research. The information visualizations are often provided in a format that is more in line with existing reports that are used by Scrum Masters, developers and quality managers to pinpoint potential issues in the process. Each visualization is described in terms of data considerations, rendering steps and interaction controls.

5.5.1 Sprint report

Based on feedback from several stakeholders, we observe that there is a need for an overview of past, current and future sprints. This overview should have different indicators that describe what has happened at various levels of detail, e.g., by component, per sprint and for each story. This ranges from information about planned and completed story points to metrics from quality control dashboards. The overview is presented as a *sprint report*.

Often, existing systems make it difficult to obtain and compare earlier data with the current situation. In addition, the metrics are scattered across multiple systems. This means that, without an integrated visualization, some stakeholders, such as Product Owners, quality managers and software delivery managers, would manually compile data from various sources in order to report on these factors. This is additional labor that hinders them in spending more time helping the team in addressing impediments or providing feedback, which conflicts with principles from SCRUM and Agile frameworks. A manual approach to reporting also means that this process is more error-prone or at least harder to keep the collected metrics consistent in this case. This is because non-automated data collection might take place at a different moment in time or some information cannot be found by the user.

In order to encourage the option to make comparisons over time and across similar projects or components developed by a team, we develop a sprint report visualization. We provide an integrated location for building, displaying and exporting a report, which contains metrics from various sources. The most important attributes are suggested more prominently by a configuration panel that controls the report. Various levels of detail are made available about the selected data, which is laid out per SCRUM sprint and indicates how it was collected, possibly with the individual stories or other measured units that were involved.

The principle behind the sprint report is to make the collected data from our research available in different formats without having to use a specialized method for each type of data, in order to stimulate usage of intermediate analysis and results by software development teams. The available data is not limited to the features that we use for prediction. The visualization provides access to a large set of attributes, many of which are geared toward reporting and separate analysis. Results from the predictions are available in the sprint report and other experiments that provide supplemental data are also compatible.

The visualization uses SCRUM sprints as a basic unit of time. For some specialized purposes, the sprint report can be generated for other time spans instead. For one support team, the data is provided based on release versions, which they work on during a specific length of time. Another support team instead uses milestones to indicate their progress, which is used as the unit of time in their customized version of the sprint report.

Flexibility is an important factor of the sprint report. This is not limited to the output format, the attributes from our data that people can select or the lengths of time to display. One can also select the fields of metadata available from the Grip on Software to be shown with each time interval, such as sequential number, name of the sprint or milestone and start/end dates. Finally, the configuration panel can be hidden, so that others can focus on the data rather than the tuning.

Data

The sprint report includes features that were extracted from multiple systems used within the development ecosystem introduced in Section 2.1.1. These features are primarily from (a) the project's issue trackers, such as Jira or TFS/VSTS/Azure DevOps, (b) version control systems and their code review front-ends, e.g., GitHub or GitLab, and (c) quality control systems, for example SonarQube. The report also provides attributes that were not selected for the prediction algorithms as mentioned in Section 4.4.1. For example, we provide alternative calculations of the velocity of a team, which use the number of story points that were "done" per day or averaged out across more sprints. Other attributes, such as the number of attachments added to issues, are also available. Details on specific metrics and code commits are also provided. The entire list of attributes can be found in the `data-analysis` code repository [h].

The bulk data is not just provided in a raw format. For optimization, data from the five most recent sprints are split from the older data. A selection of most important features and metadata—such as planned and completed story points, plus the sprint's start and end dates—are split again from the most recent data. This means that a default selection of sprints and attributes can be loaded efficiently.

Similarly, many attributes come along with details on how the aggregate value was calculated. Depending on the attribute, this includes information like issue key, story points, release version, code repository and component metric count. Details regarding attributes that are selected less often are placed in separate JSON files, improving load times.

Aside from technical considerations, we remark that some teams work on multiple projects and components, which have their own boards and repositories in the issue tracker and version control system, respectively. Some projects even have their own development platforms. Different interests of the stakeholders exist for these combined projects when it comes to tracking them, as some of them still want to receive the report per project or even display certain features for only some types of components worked on by the team. The collected data is combined for teams, which gets tallied up differently for each feature, e.g., sum, average, maximum or latest value.

While the features are usually collected from the database where the different sources from the teams are stored, external data can also be included. This is used to display results from experiments with the prediction algorithms, where the data set is increased in size by including more sprints in each run. In every instance of this run, the validation set consists of the most recent sprints. The effort estimations are reported for these sprints. This totals up to predictions for almost the complete data set, which can then be displayed using the sprint report by loading these results. Other intermediate results, such as sentiment analysis on comments made on stories during each sprint, could also be shown in the sprint report.

The results from the prediction algorithms extend beyond existing sprints. Simulations with linear regressions and Monte Carlo algorithms also provide trend progression of backlog sizes for future sprints. These results are made available, including probability density curves and likelihoods of finishing the entire backlog by a certain date.

The sprint report not only provides details on how features were estimated or collected, many of them also contain links to the original sources where they are based on. Often, we can link to a human-readable report in an issue tracker, version control system or quality control dashboard. This helps with verification of correctness of the reported values. For these sources, we determine what the latest collection date has been. This clarifies if the report is not completely up-to-date, in case the collection and report generation happens at a hourly, daily or weekly frequency, for example. In case there are any outdated sources, this can also indicate if there are any problems with earlier components of the pipeline.

Some metrics that are acquired at a low frequency can be carried over to the next sprint so that the sprint displays the previous value if no measurement has taken place yet. Other features have default values when no data is known, such as having zero attachments when none were uploaded. These transformations can avoid the absence of data where it is expected in the report.

The attributes are grouped together in categories, e.g., when they relate to team composition, velocity, alternative indicators for sprint progress, backlog sizes, code changes and quality metrics. Another portion of information regards metadata of attributes. This includes the following localization data:

- Human-readable and translated descriptions of the attributes that can be displayed as labels.
- Longer descriptions that are appropriate for tooltips.
- Units that are optionally displayed with the attribute in a context where it is helpful to further describe a standalone value.
- Shorter units that are necessary to indicate the scale.
- The standalone unit that can be used for axis labels.
- An indicator describing when a feature is used by a prediction algorithm to simulate future sprints. The indicator mentions how this feature affects the prediction compared to when it is not used, i.e., which adjustments on the backlog were factored in.

Some of the locale text is used in other visualizations as well, but the sprint report uses all of them in several formats. For attributes that are composed of other features by calculating a mathematical expression, we denote the expression and which features were involved, so that these are displayed in a human-readable manner.

We further note which features are prominent enough to select by default and which are less likely to be selected at all, which ones have predictions for future sprints and how many future sprints we would have at most, which ones have details and which ones have *targets*. The metric targets define thresholds below or above which the value is considered acceptable or good. Each target has its own direction, perfect value, good and acceptable values. These targets can change over time, thus they have a date from which they apply.

Other attributes have specific ways of displaying their data, such as emoji that indicate the team spirit, fractions for story points or time durations (days, hours and minutes) for technical debt. This group of metadata is relevant for displaying the data in a format that is practically useful, human-readable and easy to understand.

Configuration

The sprint report visualization initially renders a configuration panel where a large number of selection options are displayed. In order to keep this panel structured, the different options are labeled and clarified using tooltips. Less important options are made smaller. Figure 5.4 displays the configuration panel in full.

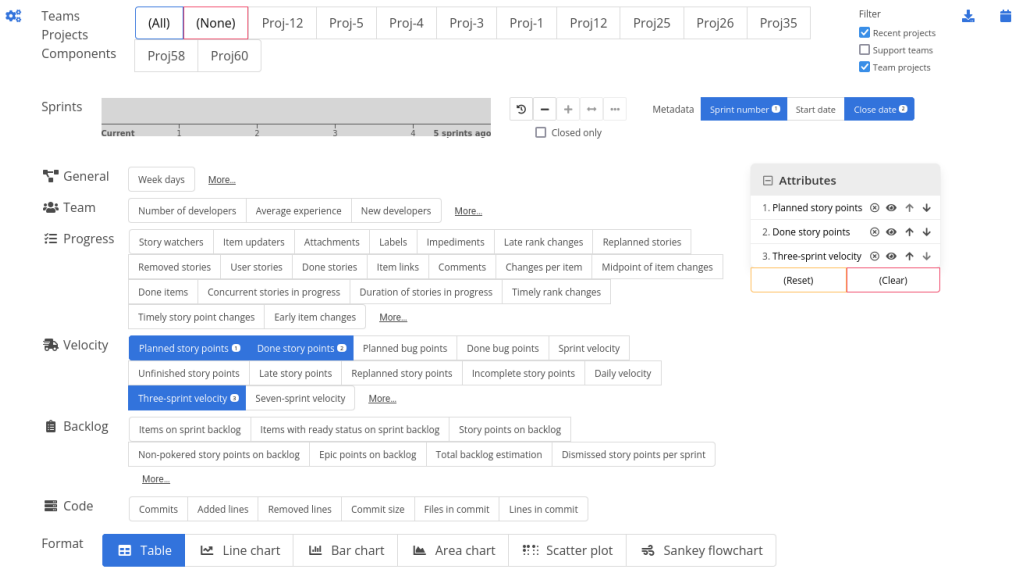


Figure 5.4: Configuration panel of the sprint report.

The first selection control allows choosing teams, projects and components that they work on. The first two options allow selecting all of the visible options, and to deselect everything. Otherwise, individual teams and their projects or components can be selected and deselected. A number of checkboxes to the right allows filtering on which teams are visible in the selection control. These filters make it possible to show or hide projects that have not been worked on recently, support teams, subprojects and projects that the user is not involved in. The latter option is based on the user's IP address range that is part of an isolated virtual network and is only available in organizations that compartmentalize their projects in this way. As such, hiding projects that are irrelevant to the user is mostly meant as an ease of use.

Another line in the configuration panel is dedicated to selecting sprints and their metadata. A bar shows how many sprints have their data included within the report. The bar by default only has five sprints. Once projects are selected, more sprints can be selected. The bar can be dragged to move the selection or to increase the number of sprints. Adjacent buttons allow resetting, fine-tuned removal or addition of sprints, including all previous sprints and showing future sprints. The future sprints—displayed with a diagonally-striped hatching pattern in the selection bar—are only available when a feature is selected that has predicted values and the output format supports them. A checkbox allows removing ongoing sprints, i.e., each project's most recent sprint that has not ended, thus reporting on finished sprints only. Finally, there are options to show or hide metadata for the sprints, such as their name, sequential number and relevant dates.

The attribute selection is collapsed by default. In its expanded state, features are grouped by categories and less relevant attributes are still hidden. The user can find those using an option to show more attributes in that group. Each category is collapsible to save screen space. Selected attributes are shown with numbers indicating the order in which they appear in the report, based on when they are selected. By default, a few attributes are selected already.

The selected attributes are also shown in an adjacent list. This list also allows resetting or removing all of them, removing individual ones, changing whether they show up for only teams, projects or components—or all of them—and reordering the attributes. The list order can be changed by clicking on arrows or by dragging the items over each other. In fact, if the user drags the selected attributes among the categories, they are also reordered. Dragging sprint metadata fields over each other similarly changes their order. Likewise, teams, projects and projects are draggable, which leads to reordering that selection. This is then reflected in the report itself, depending on the format.

The configuration panel is collapsible. The collapsed state is set in the URL so that the user can share the link to the report without the configuration panel showing up. Two more panels closely related to the configuration menu can be expanded. The first panel provides export options. CSV, JSON and HTML formats—with complete resources within a compressed archive file—are available. If a PDF rendering service is available, this is also linked. The user can always copy the link here or open a print dialog, which provides printing to PDF as well. The final panel displays the date on which the sources of each selected project has been most recently collected.

The configuration panel itself allows choosing a report format as its final selection line. The sprint report is rendered as a table by default. The visualization also supports displaying the data from development projects as line charts, bar charts, area charts, a scatter plot and Sankey diagrams, as further described below for each format.

Table

A table shows each selected team, project and component, with their attributes shown after a heading row. In the header, the name in the first column is linked to the project's issue tracker—if it is reachable from the location where the visualization is displayed—and the other columns describe sprints ordered from most recent, including the sprint metadata. The rows alternate in background color and each attribute has its own values shown. Figure 5.5 shows a sample of the table output format.

Proj-12	Jan 3, 2022	Dec 6, 2021	Nov 15, 2021	Oct 25, 2021	Oct 4, 2021
Planned story points	10	69	70	70	49
Done story points	0	20	74	80	34
Three-sprint velocity	31 ⅓ points/sprint	58 points/sprint	62 ⅔ points/sprint	45 5/6 points/sprint	38 1/6 points/sprint

Figure 5.5: Example of the table output format of the sprint report.

Some features have details that can be expanded using a clickable icon to show subtables within that row. These nested tables can be sorted by another column, such as story points, which applies to all the subtables with details for that feature. Where possible, sources of the information are provided with icons or links.

Line chart

A chart area is generated, one for each selected team, project and component, with a legend indicating the name as well as the selected attributes. Within the chart, lines are drawn with different colors for the attributes. The lines pass through points which indicate the sprints. For further differentiation, each attribute is denoted with its own point-like symbol. The line and symbol are shown in the legend as a colored sample.

The axes are dates in the horizontal direction—unlike the table, the most recent sprint is at the end of the chart—and values of the attributes in the vertical direction. If two features have a widely different scale of values in order of magnitude, then the chart displays a secondary vertical axis with the larger scale.

In case that simulated values of features for future sprints are available and selected in the configuration panel, additional lines for each simulation are included in a diagonally-striped hatching pattern area at the end of the chart. Probability curves and likelihoods of the predicted progression are then also displayed. The lines of all the attributes use monotone cubic interpolation to connect the points, which makes these continuous lines flow smoothly between the discrete points, without making new local extrema show up in between sprints. Figure 5.6 shows an example chart.

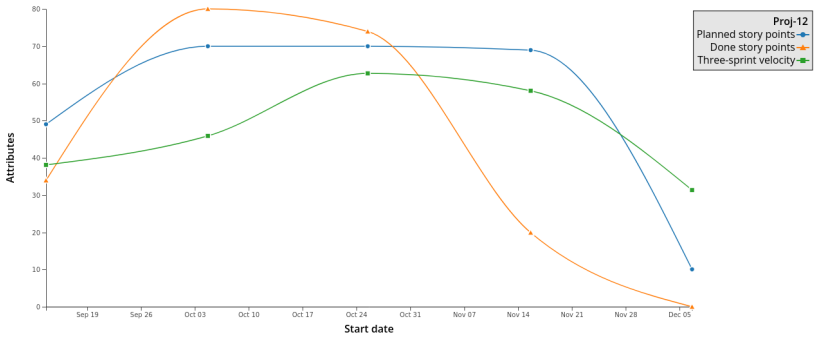


Figure 5.6: Example of the line chart format of the sprint report.

By hovering over the charts, the visualization displays a vertical line with an open circle at the closest point for a pair of sprint and attribute, with the line extending to the bottom of the chart. Additionally, a tooltip with attributes and metadata shows up. When the user clicks on the chart, the line and tooltip stick there until the user clicks again or when they hover away from and back into the chart. The charts are redrawn when the window changes size.

Bar chart

The bar charts use the same setup as the line charts with regards to axes and future sprints. Instead of points with symbols and lines, each attribute is drawn as a solid rectangle. The height of a bar is determined by the attribute value, growing vertically from the origin of the chart. The bars of the attributes for each sprint are placed next to each other with some empty space between each sprint. The bars shrink in horizontal size if necessary. An example of a bar chart in the sprint report is shown in Figure 5.7.

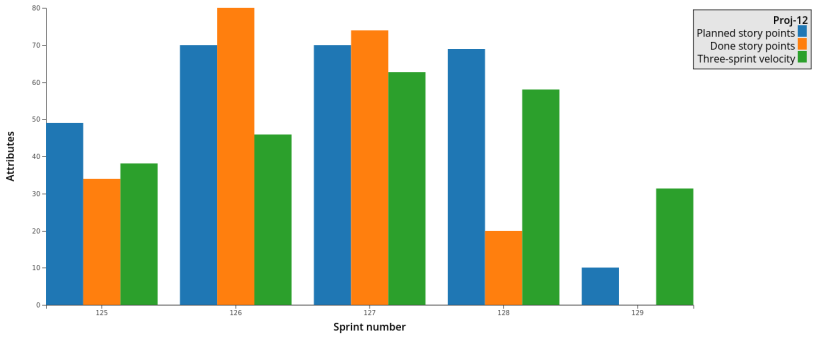


Figure 5.7: Example of the bar chart format of the sprint report.

Some features that are composed of an addition of two other features are shown as a bar with two colors, the first part extending from the origin to the first value and the second one up to the actual value of the composed feature. The bars show up like this as samples in the legend as well. The tooltip does not come with a line and open circle, but otherwise shows the metadata and attributes of the closest sprint.

Area chart

Similar to the line chart format, the attributes of the teams, projects and components are shown in area charts. Instead of displaying each attribute individually as a line with symbols, this format instead stacks them on top of each other, so later attributes end up higher in the graph. The chart is filled from the previous attribute's line until the monotone cubic interpolated line with the color assigned to the attribute, where composed features using addition are split up into the individual features. The symbols shown at the sprints are open circles. The tooltip is accompanied by a vertical line while hovering or selecting a nearby sprint and feature.

This format is meant for features that have the same unit; no secondary axis is generated if they have different scales. Figure 5.8 shows an area chart with some relevant features.

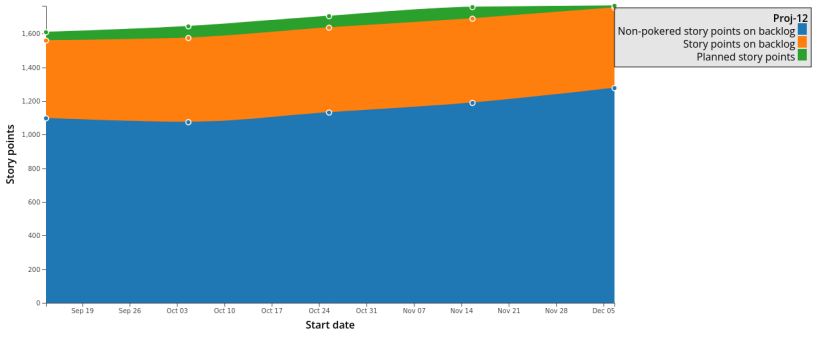


Figure 5.8: Example of the area chart format of the sprint report.

Scatter plot

A single scatter plot is generated, containing sprints from all the selected teams, projects and components. The chart has two dimensions, based on the first two selected attributes. Each sprint is plotted based on these dimensions using an open circle at the relevant coordinates, with different colors for each team, project or component. In addition, the diagonal of the chart is drawn using a gray line. This diagonal line makes it more clear where one attribute is higher than the other for a sprint, since the aspect ratio of the rendered plot is not always equal.

We plot a few more lines which perform a curve fit of the sprint's data for the two attributes. Depending on the appropriateness, the outcome of a linear, exponential and polynomial regression are shown with differently-colored trend lines as well as labels that indicate the formula and the coefficient of determination r^2.

When the circles of sprints are nearby each other, then they have similar values for the two attributes. Based on only these two dimensions, such close sprints are clustered together, forming a larger, blurred circle with a number indicating the size of the cluster. Figure 5.9 shows a scatter plot with multiple clusters.

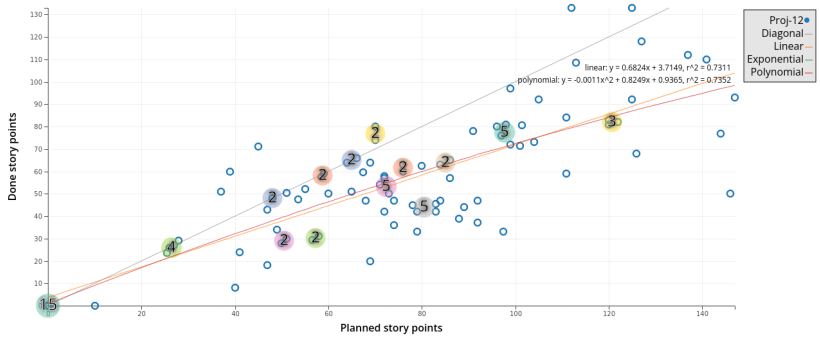


Figure 5.9: Example of the scatter plot output format of the sprint report.

If the user hovers over the chart, a nearby individual sprint is selected by "filling" their open circle with another circle, with the tooltip indicating metadata fields and all attributes of the sprint. By clicking, the user can make the tooltip stick. For this format, additional options allow cycling through the sprints within the same cluster, zoom in on the sprint—e.g., to examine the cluster it is in—or to remove the sprint from the plot. This final option is useful for temporary noise reduction, but it does not change the outcome of the regression fits.

Aside from the zoom option in the tooltip, the user can also click and drag to select an area to zoom into. Double-clicking the chart resets the zoom. Zooming in on a cluster can decompose it into individual sprints or sub-clusters, allowing further analysis into similar sprints that were readily detected by this unsupervised clustering algorithm.

Sankey flowchart

The final output format of the sprint report is a Sankey diagram, which we generate for each team, project or component if certain attributes are selected. The diagram uses the width of transitions between situations of a system to indicate the volumes that flow between states. Visualizations

based on Sankey diagrams are often used for energy flows in complex systems [112]. In our case, we display the attributes of each sprint as boxes with colors. The boxes are sized vertically based on the value of the feature. Each sprint is laid out across the horizontal axis.

Transitions indicate how much of a feature is preserved between two sprints and how much moves to other features. Based on the details of each feature, we can determine how many items—or the total weight of the applicable items—composing that feature end up being used in another feature in the next sprint. As a concrete example, we can track how many of the story points that are planned for a sprint end up being completed, placed back on the product backlog or moved over to the next sprint.

Only features with enough details on individual stories are supported. When not all relevant features are selected, we miss out on some data. The features then do not cover all possible transitions and some volumes become incomplete. Any volumes that are not accounted for when they flow from an earlier sprint or flow into a later sprint, are shown with a transition from or to a gray, dashed box, respectively. Figure 5.10 shows an example flowchart. Hovering over the boxes and transitions shows tooltips indicating which volume or flow it describes and what the size is. The user can drag the boxes vertically to reorder them. This allows untangling some transitions, leading to a clearer diagram.

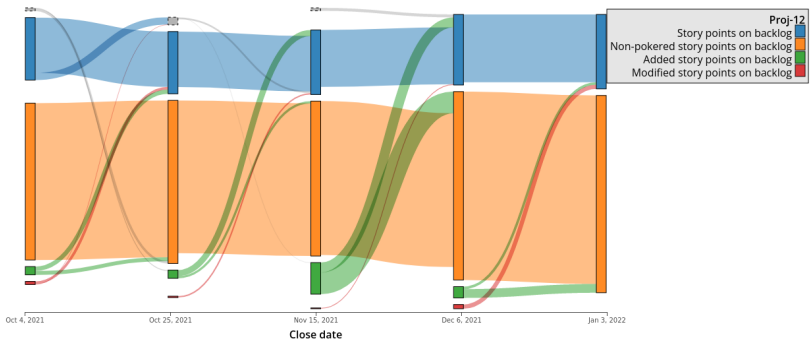


Figure 5.10: Example of the Sankey flowchart format of the sprint report.

Retrospective on interaction

The sprint report focuses on usability and clarity of the data. Many output formats allow the user to look into the details of features. The features themselves are shown with units or in a format appropriate to them, where possible. Some formats support temporarily adjustments by the user, which helps with understanding what the data means.

The diagrams are responsive to screen sizes. Tables with many columns can be scrolled. Transitions make the data render in or reposition smoothly. This helps with making the user recognize the newly shown data more easily. Figure 5.11 provides a QR code to a video demonstrating interactions with the sprint report.

Despite the interactive elements of the visualization, the export formats should still allow the data to be understood when on a static, printed piece of paper. An external PDF renderer is told to wait until the data is loaded before generating the document, which avoids missing results.

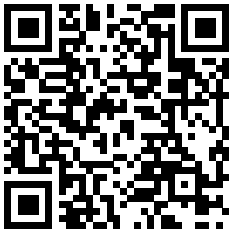


Figure 5.11: QR code of video demonstration of the sprint report.

5.5.2 Prediction results

We wish to make the predictions from the various models introduced in Chapter 4 available to various members from the SCRUM teams. This enables them to look into whether their current approach makes it feasible to resolve the stories that they have committed to within the expected time frame. We specifically focus on the prediction algorithms that determine if the planned number of story points can be resolved within the current sprint, based on a training set of earlier sprints of the projects. The models used in forecasting an entire backlog of stories are used to augment data in the sprint report visualization, detailed in Section 5.5.1.

The two prediction algorithms, namely the deep neural network (DNN) and analogy-based effort estimation (ABE), provide different kinds of details about the sprint and its context. The main part of the result, a classification or estimation of story points, is supported by accuracy metrics, analogies and configuration parameters. While some stakeholders are interested in the substantiation of the predicted result, most will focus on the label. Still, care should be taken to explain how it has come to be and what the result means.

Data

The results from the prediction algorithms contain the following data, based on experiments with the models that runs on the data set of features that were extracted from the Grip on Software database:

- The classification of the sprint in terms of finishing all planned story points. Some models provide an estimation of how many points could be finished.
- A numeric weight based on the output of the DNN and a reliability metric of the individual prediction.
- Metrics such as AUC, precision, recall and loss from the training steps.
- Configuration of the model, such as the target label and input features and actions taken on the data set before training, including rebalancing or stratification.
- The values of the features for the current sprint.
- The date when the relevant sources of the features, such as the project's issue tracker (Jira), has been most recently accessed to collect the data.

Additionally, metadata about the project and sprint are available, such as the name, sprint number, start and end date. The same is true for the sprints that were picked as analogies—similar sprints where the features were close to those of the current sprint—by the ABE model, as well as the label and the features that were relevant. Details on the two models are explained in Section 4.4.3.

Rendering

The results and relevant factors of the prediction algorithms are displayed in a plain manner. It is essential to put the focus on the most important items, using specific elements and colors within our design of the results view.

The predicted label or value is shown in a message box, which changes color depending on whether the result indicates problems with the current sprint. A positive label or a value higher than the target, which means that all story points should be possible to be finished within the sprint, makes the message box turn green, while a negative label or a lower value uses a yellow box. If the model also provides a risk indicator, then a progress bar is used to indicate this risk. A value below 20% is shown with a green bar. When the risk value is between 20% and 80%, a yellow bar is used. Finally, a risk higher than 80% leads to a red bar. The value itself is shown and described using a tooltip. The reliability value is also displayed with a percentage value and a tooltip, but no progress bar is used.

Other data, such as the dates when data was most recently collected from the ecosystem, the values of the features and the configuration of the model, is shown using tables. An example of the results is shown in Figure 5.12. Depending on the type of data and accessibility of the source, there are icons with links to the source of the data, e.g., a human-readable report from Jira providing the same values. Some compound features also have tooltips explaining how they were derived from other features, using human-readable terms instead of internal names and code expressions. Finally, analogies from the ABE model are shown in a list with links to the sprints as well as the values of the features.

Interaction

The results overview of the predictions does not provide much interactivity for the user. We consider that having many animations or other effects distracts from the main focus, which should be on the data. The tables containing the features used to train the model, the metrics from the training steps and the configuration of the model are all collapsible and expandable. This leads to a cleaner overview of the main prediction result.

The user chooses projects using a selector. When the model has trained on a combined data set of multiple organizations, then the user needs to switch between organizations to see the other projects. If the team has created multiple future sprints for the backlog—a practice that did not occur often at the two organizations—the algorithms provide predictions for all of them based on their planned story points. As such, these sprints are selectable as well. Any selection changes the URL of the visualization page to reflect this. This way, the link is shareable with others.

In order to switch between prediction algorithms and configurations, there is a pull-down menu where different experiments are described. These are based on the currently available branches of the prediction pipeline component [i]. Each branch has different parameters for the algorithms which are added to the description as a distinguishing factor. The video behind the QR code in Figure 5.13 demonstrates these selection options.

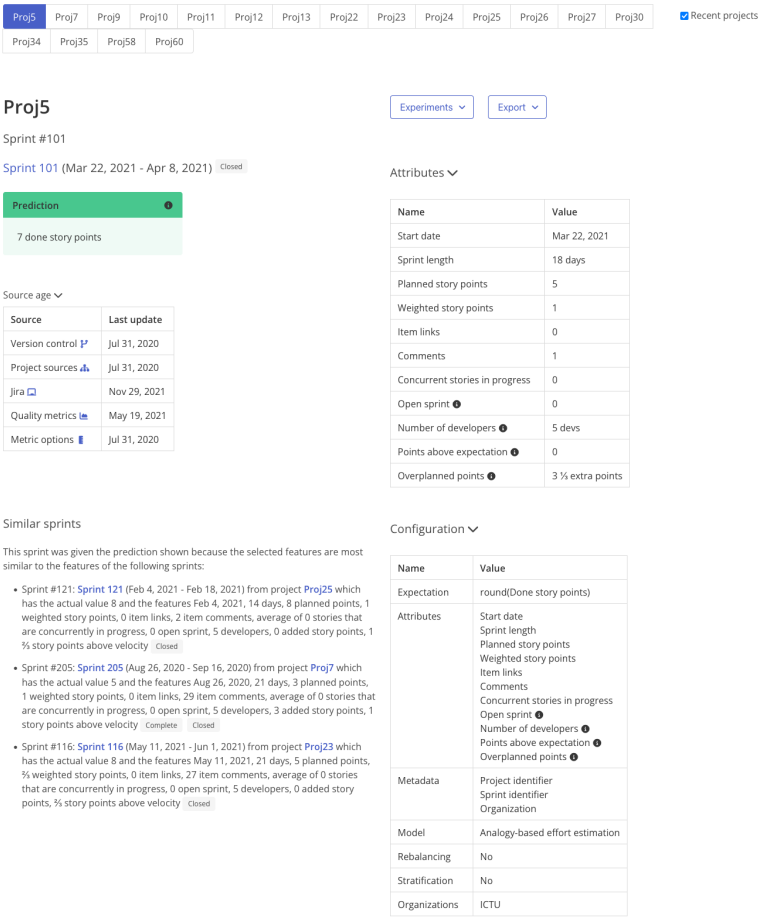


Figure 5.12: Estimated label for a sprint based on its features as shown in the prediction results.

The user finds more information about the sprint's prediction by hovering over the tooltips of the predicted label and the feature names. Links to sources are only accessible at the organization. Export options are provided in a pull-down menu, such as data sets and API endpoints. Optionally, the hub page provides documents explaining the prediction algorithms for interested users to read.

Figure 5.13: QR code of video demonstration of the prediction results.

5.5.3 Timeline

A software development process is based on several sequences of events that are well-suited for a visual representation of the volume of activities that take place. Specifically in a cyclic SCRUM process, we could see repetitions and slight alterations between each sprint or each story that is being worked on. By looking at these patterns, development teams could gain more knowledge in how they work according to this process and how small differences could change the outcome of a sprint, compared to earlier sprints or other projects. This is an essential goal for our research.

An essential aspect of the progress of a sprint and the entire project in general, is the passage of time. A visualization that displays information over time should provide enough control over this dimension, e.g., by allowing to zoom in or to exclude periods of time that are considered to be unimportant within the context of software development. When a team never works on weekends, then it should be possible to only look at week days, for example.

A well-established visualization that assists SCRUM teams during their Scrum sprint is the *burndown chart*. This time-based diagram shows a limited number of events taking place, based on changes to the number of story points left to work on during a sprint. When zooming in to a specific period in a larger *timeline*, it makes sense to connect the more familiar burndown chart to the patterns seen during that period of time. This helps to clarify both the timeline and the burndown chart, by associating other types of events, such as commits or impediments, with the changes to the remaining story points.

The timeline is a showcase of combining multiple events and features that we collect for SCRUM sprints, into a visualization based on multiple time series. While the focus is not to compare projects as a whole to another, it makes sense to show each project alongside each other to allow the patterns that arise from each time series to be seen more clearly across an organization.

Data

The main type of data that we collect for the timeline are events. These are indicators of a type of change that takes place at a specific timestamp. The following events are collected from the issue tracker, version control system and quality control dashboard:

- **Sprint start**: The date and time when a sprint of a project starts.
- **Sprint end**: The date and time when a sprint of a project ends. This is based on when the sprint is either completed (all stories are done) or when the sprint was planned to end, whichever comes first. Together with the start of the sprint, this defines the date range in which a sprint took place.
- **User story**: The date and time when a user story is done. The event also includes when the story started to be in progress, although the event is not considered a range to avoid spreading it out over a large period of time.
- **Rank change**: When the rank of a user story on the backlog changes. Here, a rank is the position of the story on the product backlog, which roughly corresponds with a priority towards resolving it. The rank could still change while the story is planned for a sprint, although this is considered late. Most rank changes take place when the Product Owner is assigning which stories come first in future sprints.

- **Story point change**: The moment when a story is given a different number of points during a sprint. This might indicate a change of scope or when a story cannot be completely done during a sprint, at which point unfinished tasks are split off into a follow-up story.
- **Red metric**: An indication of when a metric in the quality control dashboard of the project has been given a red status for more than a week. These metrics, related to problems in quality of the product's code or impediments in the process, are considered to be below a certain norm, but could not be resolved quickly.
- **Impediment**: When an issue is marked as having an impediment, for example when the team expects more details from an external party.
- **Commit**: The date and time of a commit that makes a change to the code of the project. Due to the high volume of commits, these are separated from the other events in the data but are available for a more detailed level.

Aside from the events that take place in a sprint, we also collect features that describe the sprint, including those mentioned in Section 4.4.1. Furthermore, we collect information relevant for a burndown chart, namely the following:

- The total number of story points from stories that a sprint starts out with as planned in the sprint backlog.
- The story points of stories that are "done", i.e., given a resolution status during the sprint.
- The story points of stories that are added during a sprint.
- The story points of stories that are removed during a sprint.
- The difference in story points of stories that have a change in points during a sprint, after and before the change.

Rendering

The timeline is displayed as strips of events, indicated using dots with colors. The vertical axis displays different projects, while the horizontal coordinate uses a time-based axis. At the top of the timeline, the labels of the time axis are shown, which are more or less granular depending on the zoom level and the screen size. For example, when month names are shown at a zoom level, the beginning of a new year uses the year instead of January. An additional pair of labels on either side of the axis shows the precise date range that is displayed. Here, additional options for interacting with the coloring, the scale and the zoom level are also provided.

The events within the timeline are rendered mainly using circular markers. The start and end points of the sprint are delineated by black borders and a gray background that spans between the two timestamps. All the other events use different colors for their type of event. A legend at the bottom of the timeline indicates the colors used for each event type. When multiple events of the same project take place in a close proximity of each other, then the size of the marker increases and the area that is filled by this marker becomes more blurry. This has the effect that other events that took place around the same time, whether they are of the same type or not, become more associated with the larger area around these grouped events. From a larger perspective, this makes

it clear that more activity took place at this moment in time, compared to other periods. The timeline is limited in the type of events shown, which are meant to act as a proxy for other, related events not shown in the chart.

When mapping the event types to marker colors, special care is taken to select a color scheme that is distinguishable for people with color blindness [105]. This is important because the events are only rendered using a colored marker, with no other factors that make each type of event stand out from the others. While it is possible to include more details of each type through interaction, such as with a tooltip, this would not aid in making the entire timeline easy to understand for people with color blindness. Another option would be the use of glyphs to differentiate the event types, but the blending effects would make some glyphs hard to distinguish after all. Such problems are hindrances to the aim of allowing the user to see patterns within the timeline. Therefore, we use a color scheme that avoids certain combinations of colors, such as red and green, or darker shades of blue and green. The chosen color scheme uses black, yellow, green, orange-red, blue and pink as indicators of different types of events, respectively for sprint starts, rank changes, story point changes, red metrics, impediments and completed user stories. Figure 5.14 displays the main timeline for several projects using this color scheme.

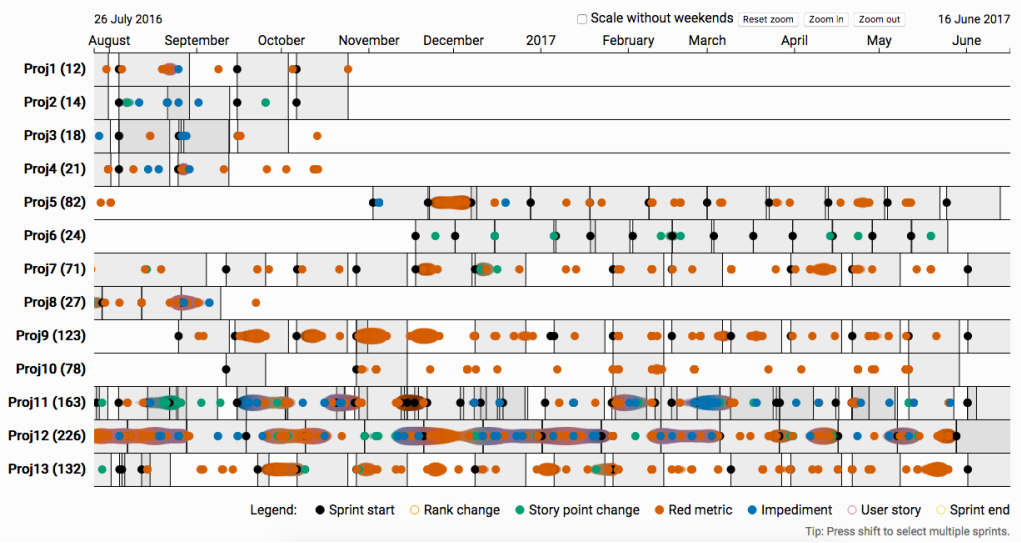


Figure 5.14: The default timeline view with colored event glyphs for projects displayed over time.

Interaction

The rendering of the timeline is not complete without an interactive element, as already suggested. The level of detail that we provide is simply too high to display in the default rendering. Thus, more control is provided to view these details. The buttons to zoom in or out at the top adjust the time coordinate to display a smaller or larger subset, respectively. Moreover, if the user drags the timeline to the left or to the right, the coordinate system changes to display another time period. Another button resets the zoom when pressed, which also brings the most recent date back into the visible range at the right of the timeline.

To the left of the zoom controls, a checkbox provides the option to change the scale to exclude weekends when selected. This compresses the timeline by removing Saturdays and Sundays. We found that there is rarely much activity on these dates at the two software development organizations. When the weekday-only scale is used, any activity on weekends is counted as taking place at the end of the Friday before the weekend days.

A selection box and numeric condition left of the checkbox allows labeling sprints with colors. The features from our analysis are selectable. A configurable threshold value determines which label to assign. The label then becomes visual through the background color of the sprints within the timeline: a green background color indicates a matching sprint, while dark gray sprints are non-matching. This provides a customizable method of finding sprint patterns.

When the user hovers over the sprints and events, a tooltip shows up with more details. For sprints, the name and exact time period of the start and end is shown, along with all the features of the sprint. Hovering over an event shows the sprint name and the exact time when the event occurred. Some events carry more details, such as when a user story started for done user stories.

The legend is also interactive. By clicking on an event type, all events of this type are shown or hidden. The sample circle within the legend indicates if events of this type are displayed in the timeline or not, by being a filled circle or only outlined, respectively.

When the user clicks on a sprint, a sub-chart is rendered below the timeline. Here, a smaller timeline shows all the events, regardless of whether they are shown in the main chart or not. Additionally, a line with commit events with gray markers is drawn. Below the sprint's timeline, a sprint burndown chart is displayed. The vertical axis of this chart indicates story points and the horizontal axis shows time, aligned with the timeline above it. A green line indicates the ideal progress of the sprint backlog, while a blue line with different event markers graphs the actual progress. An orange, vertical line is drawn at the end of the sprint.

By holding the Shift key while clicking a sprint, the user is able to select multiple sprints after another. This way, the sub-chart below the timeline displays the sub-charts of all the selected sprints stitched together. The burndown chart continues along after the ends of each sprint. As such, any late changes and continuations into the next sprint are visualized. This allows the user to compare the progress of multiple sprints and see if one sprint influences the other, for example. By selecting a sprint without holding Shift or choosing another project, the multi-selection ends. Figure 5.15 shows a QR code for a demonstration video of these interactions with the timeline.

It is normally not possible to perform the multi-selection on devices without a keyboard. The timeline is not optimized for display on smaller screen sizes, so mobile devices are already affected in this way. We have found that the timeline renders and interacts nicely on larger screens, as the visualization adjusts to such screen sizes. We configured certain input devices, such as the Wiimote, to emulate pressing the Shift key when a certain button on the remote is held.

Figure 5.15: QR code of video demonstration of the timeline chart.

5.5.4 Leaderboard

Software development teams within the same organization have different approaches toward working with the systems and frameworks that are available. Even when SCRUM teams use the same framework and have a similar mindset, each project has its own requirements that lead to distinct situations. It is not clear whether one approach is better than the other, but we quantify them using the data that we collect from the support systems. This way, we demonstrate on a high level what kinds of projects there are, for example based on how many stories or other types of issues are resolved in a sprint.

While the main research questions do not necessarily involve comparing projects or teams to each other, it is still beneficial to have a visualization that makes it more clear how they operate. For the SCRUM teams, it helps with seeing whether they are splitting up their stories into small, workable chunks or if other teams do this even more, for example. A low score on certain project-wide metrics could help with indicating that more resources are required in areas such as build systems, versioning control or structured issue linking.

The concept of a *leaderboard* supports the process with making such deficiencies within an organization more clear. It is not meant to directly compare projects with another. Therefore, care should be taken to avoid making an actual, definitive ranking, such as when an actual competition takes place [113]. Scoring should be possible in multiple ways, on top of displaying the metrics in a separated fashion. Inspiration for the notion of a leaderboard to show scores for specific areas relevant to software development teams within larger organizations comes from GitLab's DevOps Score [XXII], previously known as Conversational Development Index.

Data

We collect the following project-wide features that indicate values across the life span of each project, from the issue tracker, version control system and build system:

- The number of all the issues within the project.
- The number of stories in the project.
- The number of comments added to the issues of the project.
- The number of links between issues.
- The number of test cases created within the project.
- The number of repositories used for storing code.
- The number of version tags added to commits of the code.
- The number of commits made to make changes to the code.
- The number of times that the developers upload or *push* the commits that they made to the collaborative version control system. This is only available if the system tracks this information, which is the case for GitLab.
- The number of merge requests that are made for changes made on branches of the code in the repositories.

- The number of release versions made for the product.
- The number of build jobs within the continuous integration build system.
- The number of sprints that the project has had.
- The life span of the project, in days.

The last two features are included mainly as normalization factors, which are useful for dividing other features by. This enables us to have more reasonable scores for both long-term and new, shorter projects. Other features are also applicable as normalization factors in the eventual visualization. As such, the scores are not generated during collection, but within the visualization.

We also include other metadata, such as which factors to use to normalize each feature by default—or to initially display them as is—as well as grouping of features and links to the sources of the values, if accessible in the organization.

Rendering

The leaderboard visualization is composed of some selection controls and the features, displayed as cards with several metrics. The user selects a project to view. We display the name of the project along with an average score across all features. Adjacent to this heading, there are options to change the order of the cards and the means of scoring. The order is based on the name of the feature, the group in which it belongs—with features collected from the same system grouped together—or the score, with the lowest value showing up first.

The feature's value determines a ranking of projects, where the project with the best value is awarded #1 for that feature and so on. The average total rank of a project is based on all ranks added up, meaning that the project with the total lowest ranks is again awarded #1. Another option is to compare against the leading value across all projects, where the project with that value receives 100% and all others have a lower percentage based on the ratio between their own value and the leading value. The average total score is then comprised of the scores in the same fashion. Another option is to compare against the mean value, which for projects that perform above the mean leads to a score above 100%. This follows the same calculation as for the lead score.

Each card is headed by its feature name, optionally the normalization factor behind a division slash, the score, possibly a link to the source of the feature and a link to "swap" the cards. When the cards are swapped, instead of showing all features of a single project, the chosen feature is shown for each project. This allows the user to inspect the distribution of this feature across all projects, including the scoring metrics.

Within the card, information about the feature's value for that project, the leading value and the mean value is shown. Below, a box plot of the feature's value of each project is displayed. The box within the plot indicates the percentiles 25%, 50% and 75%, i.e., the three quartiles. The whiskers extending from the plot are dashed lines displaying the interquartile range of the first and third quartile. Additionally, the box plot indicates outliers with small circles outside of the area in which the lines of the whiskers appear.

The box plot provides more statistical background behind the visualization. For example, this helps in determining whether a well-performing project for a certain feature is exemplary within the organization or in fact an outlier. The project is shown within the box plot using a blue line at the value of the feature.

The scores are made more prominent using colors that relate to the score's value. For ranks, the top 3 projects receive a green-colored rank, the remaining top 10 a yellow color and the other ranks are red. For scores, any score above 75% is colored green, while it is yellow between 40% and 75% and red for worse scores. Figure 5.16 displays an example of the ranks within the cards detailing a project.

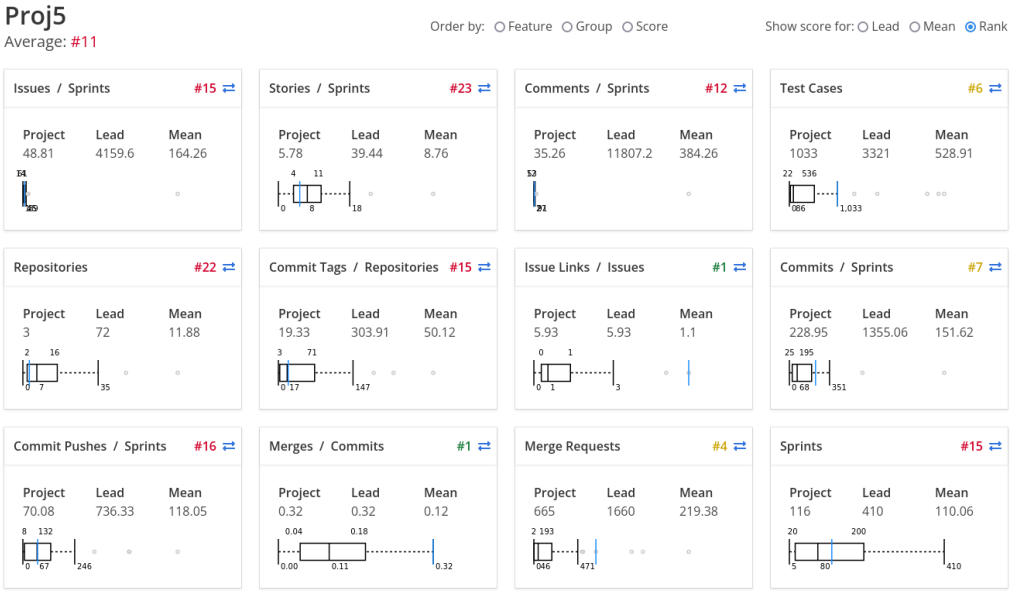


Figure 5.16: The leaderboard visualization for a project showing its ranks and cards.

The scoring assumes that higher is better, which is sometimes the case but irrelevant for some features. The number of repositories a project uses is more of an indicator of the complexity of the code base than a necessity for all projects. Still, there is some rationale in using a simplistic scoring where the focus should be on sparking conversation between developers within teams.

Interaction

As already mentioned for the rendering part of the visualization, there are several selection, ordering and scoring utilities as well as a means to swap cards between single-project and single-feature state. Additionally, by clicking on the lead value in a card, the user jumps to the project that has this value.

The user can also drag and drop cards in the visualization. This either changes their ordering or adjusts a normalization factor. When a card is dragged to another card and the other card already has a different normalization factor—or, in case of the swapped cards layout displaying a feature for each project—then their positions are switched. They lose this adjusted placement when another order is chosen. However, when a card is dragged to a non-normalized card, it provides the former card's feature as a normalization factor. The card's score is also recalculated. If the dragged card has the same feature as the normalization factor that it is dropped on, then the normalization is cancelled. The video behind the QR code in Figure 5.17 shows this in action.

Figure 5.17: QR code of video demonstration of the leaderboard.

5.6 Visualizations for ecosystem management

This section presents visualizations that help with discovering situations within the larger software development ecosystem within an organization. These information visualizations are intended for different audiences and often have a more specialistic aspect to them, which means that quality managers, DevOps engineers and Scrum Masters are among our most prominent stakeholders. We describe each visualization according to the flow in Figure 5.1: data selection, rendering of a visual form and the interaction around the view.

5.6.1 Collaboration graph

One practical aspect of a large software development organization is that there are multiple teams working on different projects. These software development projects share similar traits between them. This is not only on technical basis—programming languages, software libraries or development applications—but also in the sense of the development framework, i.e. SCRUM. Team members have knowledge about former, completed projects that they continue to use in other projects. Their experience benefits the velocity and the quality of the delivered product.

The large scale of an organization makes it difficult to keep track of this common knowledge factor: who knows what, who has worked on which project, who could benefit which team? Some of the organizations also have support teams. These teams help with maintaining the development platform, for example. Additionally, guilds are formed for discussing technical topics or coaching Scrum Masters. This helps with centralizing the knowledge about the various teams. For management planners, Scrum Masters and other stakeholders, it is beneficial to know how to distribute former members across new teams to help kick-start a project.

Often, it is mentioned that everyone in a SCRUM team should be aware of the work that the others are doing, hence the need for meetings such as the Daily Scrum. In fact, developers should be at a similar level on each of their peer's area of expertise—with such versatility sometimes referred to as a T-shaped skill set—so that they are able to take over their work in case of illness or other circumstances. This is jokingly referred to as allowing a team member to be driven over by a bus while remaining resilient enough to continue as a team.

Keeping track of each developer's skill set is outside the scope of the Grip on Software research. However, we provide a means to find out who has been involved in current and finished projects as well as what their roles were, insofar that this is deducible from the collected data. This visualization is provided as an interactive network graph, where the nodes are either projects or people, while their connections indicate a person's involvement in the project. This visualization is called the *collaboration graph*.

Data

The Grip on Software database contains a number of models describing projects and people, which are collected from different sources, such as project trackers like Jira [I], version control systems, e.g., Git [II] and potentially the project registration system, which keeps track of estimations for full-time equivalents (FTEs) and seat counts that a development project has appointed.

From these sources, the following data fields are selected in order to report on the people involved in the projects as well as their probable role:

- The person's display name from the project tracker.
- The project name that the person worked on according to any of the tracking sources or an anonymized identifier of the project.
- Whether the person has an email address that is linked to the organization or that it is an external email address.
- The number of commits in the version control system by the person.
- The number of changes to stories and other issues by the person.

Some of this data is encrypted for privacy reasons before the data is selected in order to update the visualization. To resolve problems with this, we ensure that selecting the encrypted data works the same way. For example, we do not use the email address directly, but keep track within the database with a Boolean field whether it is an internal address or not. Additionally, we compare encryption hashes between different tracker data on persons in order to avoid multiple nodes for the same person.

In order to show a timelapse of changes to the network, the data is also collected at intervals of one month. This means that only commits and changes made during an interval will be counted for that person's actions at that time. As such, the person is not considered to be active on a project before the first moment that they work on that project and similarly after they finish working. However, the "complete" visualization continues showing these, for reasons described later on.

Rendering

In order to use the selected data for a visualization, we consider the means of displaying this data. First of all, the relations between persons and projects is best defined as a graph $G = (V, E)$. The set of nodes $V = M \cup P$ in this graph consists of disjoint sets of team members M and projects P. The edges $E \subseteq V \times V$ indicate whether a person has worked on a project. There are no edges between two persons or two projects. The graph could therefore be seen as a directed bipartite graph where persons only have outgoing edges and the project nodes only have incoming ones:

$$\forall (v, w) \in E : v \in P \wedge w \in M \tag{5.1}$$

However, for information visualization purposes, it is better to distinguish the types of nodes in a clearer manner than using a directed graph. The display of arrows on the links would not be suitable for making these different properties clear. So while the graph remains directed, the rendering uses another mapping to demonstrate the types of the nodes.

The mathematical representation of a graph is best visualized using a network, where nodes are circles and edges are lines, the latter replacing the arrows in a usual directed graph established by Equation 5.1. In this rendering, while the precise coordinate locations of each entity is not intrinsically relevant, there are some common preferences toward drawing the graph, such as overlap of nodes and lines and closeness of nodes. These and more preferences are guided by a force simulation which determines a proper layout for the graph.

The network visualization itself does not need to be static. Several techniques exist to render a graph which take several correcting steps that improve the qualities of the displayed network. A force simulation can have different weights to nodes and links, which then cause them to attract or push away other entities. By configuring parameters for the forces of nodes and links, these forces take steps into a direction that makes the network visualization easier to be understood by the viewer. We define the following forces for this purpose:

- A *link* force: The nodes that are directly connected to each other should be closer to each other than ones that are not.
- A *many-body charge* force for each node: It is preferred to have as little overlap of circles as possible. Also, project nodes push other nodes away more than developer nodes. The Barnes–Hut simulation [114] plays a role in approximating the many-body forces.
- Three *center* forces: We do not want to force the nodes to be too far apart, because the whole network still needs to be visible on a limited screen size. Separate forces make the network remain focused upon the center of the screen and avoid growing larger than the view box, taking into account the aspect ratio.
- A *radial* force: Preferably, the graph is drawn in a circular or oval shape rather than having nodes forced into corners of the screen.

All these forces are included in a graph drawing algorithm [XXIII] that uses velocity Verlet integration [115] to calculate new positions, velocities and accelerations for particles, using a decay to slowly halt the drawing. Figure 5.18 shows a fairly stable state of a network.

As mentioned before, the type of each node—displayed as circles in the network—is not determined by the direction of its links. Instead, the size of the node is larger in case of a project and remains small for persons. This makes projects stand out more as hubs within the network. Additionally, support teams will have their project show up in a different, dark blue color. Support team members themselves have a bright blue color and people with external email addresses are colored orange. Finally, if a person has not made commits in the version control system, they are considered to not be a developer, thus we fill the node using a different, light blue color.

Interactions

The network visualization is presented as a web application where several additional options allow the user to adjust the visualization and gain more insight into the collaborations. Next to the network, a legend is displayed, indicating all the roles and sizes/colors. The legend also includes a count of each role shown in the network, as well as the number of links. The network itself is interactive: the user can drag nodes to move them toward another location. Meanwhile, the force simulation drags connected nodes along with it and other nodes away from their positions. Afterwards, the simulation finds a new layout to stabilize toward based on the forces.

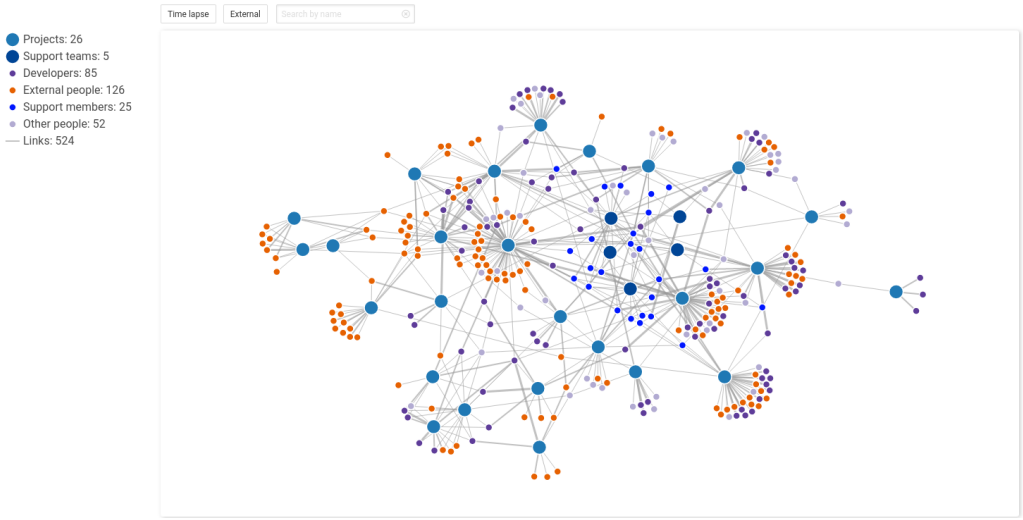


Figure 5.18: Visualization of a collaboration graph of a software development organization.

The view box of the network visualization also responds to changes in screen size. The visualization will attempt to encompass a larger space when it is made available, for example when the browser window is resized, maximized or when a larger screen is connected. Similarly, the nodes within the network will move closer to each other and show up slightly smaller when the screen is less wide, for example on a mobile or tablet device.

The visualization provides an option to exclude all external people from the graph. Links involving the nodes of these people are removed and this is reflected in the legend. This might make the graph disconnected, but there is no precondition that the graph was connected in the first place. Given that all people from the lifetime of all the projects are shown, this is however rare. Any disconnected project or small graph component will move itself away from the main component due to the forces of the abundant links and nodes in this part of the network. The option to exclude external people helps with determining if knowledge regarding projects is outside of the periphery of the organization, which potentially leads to missing out on retaining skills relevant for those projects.

It is possible to search for people and projects in the network, assuming that it has not been encrypted and anonymized. If the keys used to perform encryption are available at the organization, live encryption of the search keyword is used to compare the hashed versions. When a search leads to a match, the view box zooms in to the position of the node and the node becomes highlighted. Canceling the search brings the full network back into view. By hovering over a node in any view mode, the user is able to find the name of the person or project.

The final interactive mode of the collaboration graph is the timelapse. Here, the data of each interval is used for the network visualization, which is shown after another at time steps. The current month is displayed with a label above the view box. There are controls for pausing/resuming the timelapse, for slowing down and for speeding up each time step. This way, the evolution of the organization is shown in a manner that feels familiar for people involved at any moment in the timelapse. Figure 5.19 holds a QR code for a video with the timelapse.

Figure 5.19: QR code of video demonstration of the collaboration graph.

5.6.2 Process flow

There are various ways to look at how a software development team resolves stories on the product backlog. Regardless of whether the team uses SCRUM, another Agile framework or even other non-Agile software development principles, the tasks that the team needs to do are generally managed by assigning different statuses in an issue tracker. Such a status provides an indication of the maturity of the issue. To be more specific, the status describes progression toward resolution of the topics mentioned in the task.

A status has different meanings within the software development frameworks. Even between teams, the actual use of each status has slight nuances. Generally, a new story starts out as an *open* or *new* issue, then it goes through an *approval* process, e.g., a refinement. After it is selected to be worked on, a developer assigns it to work on it and marks it as being *in progress*. Most likely, the developer then *resolves* the issue by making changes in the code. The change usually leads to a testing phase. After all necessary actions have been taken on a story, it is *closed*. This corresponds to a definition of "done" (DoD) often in use in SCRUM.

While this approximately follows the route of a story that leads to a successful code change, not every backlog item takes the same path. When a story at some point cannot be resolved in this way—possibly even after closing it—the story may be sent back to an earlier state, often referred to as *reopening* it. A story that does not lead to a code change is often given a different resolution, which marks it as invalid, redundant or fixed in another manner.

We wish to track all these possible states and to find potential bottlenecks in the procession of stories from the backlog to completion. The *process flow* is a visualization that makes clear what the main flows of these stories are. It also includes details about the volumes and amount of time involved in the state transitions.

Data

For each project within our data set of the organizations, we collect the following data from the project's issue tracker:

- The old and new status of a story or subtask.
- The old and new resolution of a story or subtask. Most issue trackers keep this as an additional field with its own possible values, but that only has a change if the status is in a done or closed state.
- The timestamps of the change when the new status/resolution is made and the change when the old state was selected. Potentially, the latter is the creation time of the story or subtask.

The status and resolution is encoded in different manners in the issue tracker's API. For example, Jira uses numerical identifiers for these fields, but also provides a mapping to human-readable names as well as status categories. The rendering of the field within the issue tracker depends on the status category to define its color.

We also convert the timestamps of the two changes into a date interval. We here subtract the weekend days within the interval in order to more accurately obtain the number of working days that the story or subtask has been left in the old status or resolution. Note that the heuristic of subtracting weekend days does not consider holidays.

The combinations of old status/resolution and new status/resolution are grouped together, so that we perform aggregate operations over the state changes. We calculate (a) the number of stories and subtasks that had such a change on the project's backlog and (b) the average of the number of working days that issues have spent in the same state before going to the other.

Rendering

The processed issue tracker data describes the aggregated progression of stories and subtasks toward their resolution, but it could include "backward" transitions. The grouped data is therefore formulated as a weighted, directed graph $G' = (V, E, M, N)$ where the nodes V are status/resolution values and the edges $E \subseteq V \times V$ are transitions between these states. Two sets of weights provide the volume and duration properties of each transition. The mapping $M\colon E \to \mathbb{N}$ defines the number of stories and subtasks that followed the state change and $N\colon E \to \mathbb{N}$ the average number of working days that those issues stayed in the old state.

Each of the nodes from V consists of a pair of status and resolution $(s, r) \in V$, where the resolution $r \in R$ may be empty, which we denote as ϵ. The status $s \in S$ is involved in a mapping that determines the status category $T\colon S \to C$. The following statuses, resolutions and status categories are known, based on Jira and TFS/VSTS/Azure DevOps:

$$
\begin{aligned}
S_1 &= \{\text{Open, To Do, New, Requested, Design, Accepted}\} \\
S_2 &= \{\text{In Progress, Approved, Reviewed, In Review, Ready}\} \\
S_3 &= \{\text{Reopened}\} \\
S_4 &= \{\text{Resolved, Committed, Done, Removed, Completed}\} \\
S_5 &= \{\text{Closed, Validated}\} \\
S &= S_1 \cup S_2 \cup S_3 \cup S_4 \cup S_5 && (5.2) \\
R &= \{\epsilon, \text{Fixed, Won't Fix, Duplicate, Incomplete, Cannot Reproduce, Not Fixable,} \\
&\quad \text{Building, Manual Testing, Automated Testing, In Review, Processed} \\
&\quad \text{Known Issue, Redundant, Works as designed, Invalid}\} && (5.3)
\end{aligned}
$$

$$
\begin{aligned}
&\forall s_1 \in S_1 \colon T(s_1) = \text{Open} \\
&\forall s_2 \in S_2 \colon T(s_2) = \text{In Progress} \\
&\forall s_3 \in S_3 \colon T(s_3) = \text{Reopened} \\
&\forall s_4 \in S_4 \colon T(s_4) = \text{Resolved} \\
&\forall s_5 \in S_5 \colon T(s_5) = \text{Closed} && (5.4) \\
&\forall (s, r) \in V \colon s \in S_1 \cup S_2 \cup S_3 \Rightarrow r = \epsilon && (5.5)
\end{aligned}
$$

Here, Equation 5.5 implies that only statuses in S_4 or in S_5, i.e., the Resolved or Closed categories, can have a non-empty resolution; for all the other categories it must be empty.

The status categories are rendered in the visualization with a colored outline of the nodes within the flow diagram. A status in S_1 is given a blue color, S_2 is yellow, S_3 gray, S_4 is colored light green and S_5 is dark green, akin to colors given to these categories in Jira. The nodes themselves are rounded rectangles with the status and resolution within it. The directed edges are rendered as arrows. The volumes and time spans are placed nearby the midpoints of the arrows. The rendering attempts to avoid too much overlap between the rectangles, arrows and numerical labels. The volumes also determine the thickness of the arrows. The color of the numerical labels is defined by the working day deltas, using a color palette from blue to orange that assigns a "hotter" color to longer waiting times.

The status categories also define the vertical position of the rectangles within the flow graph. The nodes that are assigned to the Open category S_1 receive a placement at the top, while the Closed category nodes (a status from S_5) have their corresponding rectangles at the bottom. A status not in these categories has its node placed in between these layers, aligned vertically. This means that the states from the remaining status categories, namely In Progress (S_3), Reopened (S_4) and Resolved (S_5), are on mixed levels or even in an unexpected order. However, if there are enough transitions that follow the usual route, the rendering should give precedence to untangled arrows.

The rendering uses the Graphviz markup language known as DOT [XXIV] to encode the layered, directed graph. This DOT exchange format is used by various programs, such as those included with Graphviz [116]. We use a library [XXV] to render the graph in our visualization using a Web worker [XXVI]. An example rendering is shown in Figure 5.20.

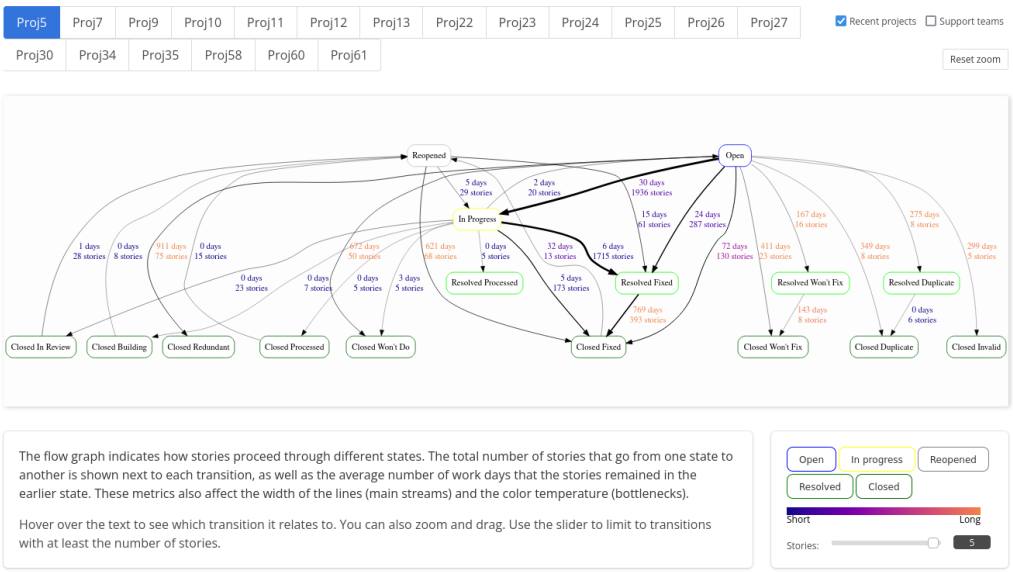


Figure 5.20: The flow graph of a project including legends and other controls in the process flow visualization.

Interaction

The Web worker allows the visualization to be non-blocking, i.e., the view is available while the flow graph is being rendered. Using a selector, the user chooses which project to view. A toggle allows showing and hiding older projects as well as support teams. Because we render the diagram as data-bound HTML elements, selecting a different project creates a transition effect. For each state and transition that the newly selected project has in common with the previous project, their positions move toward their new locations. The arrows change in thickness, length and arc angles as well. This makes it easier to visually compare two projects, even if one cannot select multiple ones at the same time.

An additional slider, included with the descriptive legend of the diagram, allows limiting how many transitions to show based on volume. When the slider is lowered, the state transition with the fewest number of stories involved in them are removed first. A numerical indicator shows the minimum volume for the visible transitions. Nodes that no longer have incoming or outgoing arrows are also removed. Adding or removing transitions involves a similar effect to selecting a different project. This option is helpful for focusing on the main stream of the progress.

When the user hovers over a numeric label, a tooltip shows the two status/resolution pairs that are involved. This helps clarify in case it is unclear which transition the label refers to. The view of the flow graph can also be zoomed and panned. A button resets the view to its default zoom level and position when pressed. Figure 5.21 provides a QR code to a video with these interactions.

Figure 5.21: QR code of video demonstration of the process flow graph.

5.6.3 Heat map

In a software development process, the main goal is to create a product that satisfies certain functional and non-functional requirements, adding value to the product. There exist different techniques to increase the reliability in the process towards this goal. Our research focuses for a large portion on estimating the effort and tracking the progress toward an acceptable product. This mainly includes the data that is stored at the project's issue tracker. However, the changes to the code base of the product provide more insight into the more technical part of development. This allows us to consider other estimations of effort or indications of problematic events during the development phase.

An alternative indication of effort could be the volume of code changes. For example, we count of the number of commits, the number of lines or bytes changed in each commit or other activity metrics related to frequency of commits. Developers sometimes indicate the story they are working on in their commit message, which helps tracking the progress of fixing the problem or considering the feature to be "done". However, this practice is not standardized across many teams

and does not cover enough of the project's life span, thus meaningful indicators that associate commit volume with story effort are hard to extract reliably.

Still, commit volume is a property that is familiar to software developers and testers. They often have a feeling of success when a commit with their code changes proceeds to be tested, merged into the main development branch and included in eventual releases. There is some individual sense of achievement. While it is not a competition, being successful in developing features sparks a drive which improves the team effort as a whole. We wish to focus on the collaborative aspect and find if teams have different patterns over time with regard to code commits.

Visualizations of commit volume over time exist in popular version control systems such as GitLab and GitHub [XXVII]. A calendar shows a developer's commits on each day. This *heat map* is sometimes shown on profile pages found on these systems. In our case, the focus is on the collaborative aspect instead of individual performance. The entire team has a responsibility for a stable throughput of code changes. A heat map helps visualize this metric over time.

We also want to extract more knowledge from the heat map and its commit-based metric. We wish to include other data regarding events that took place on the dates of the commits. External data helps with further testing of the metric, for example by finding whether there is a relationship between the day's weather and the number of commits.

Another possible use for a temporal visualization of commits is a detailed look at the files that are changed. When files have not been touched for a long time, this might indicate a long-forgotten component that has worked without problems. It would then become difficult to keep the component functional within a changing environment. A code change after a long time may suggest that this file contains other hidden problems, such as dead code, that should be reviewed more comprehensively. In some programming languages, deeply-nested components could lead to loss of expert knowledge. The heat map visualization indicates such situations when they arise.

Data

We collect the following data from the version control systems of the projects in order to fill the heat map:

- The number of commits on a date, excluding commits made by automated systems.
- The number of developers that made a commit on a date.
- The code repository, possibly a human-readable URL of a specific file in the repository, the filename and earlier date that the file was changed. This is only included if a commit was made to the file on a date and the difference between the this date and the data of the previous commit to the same file is more than a certain threshold.

Additionally, we collect the average temperature of each day from the closest weather station to the location of the two organizations included in the Grip on Software database, namely The Hague, the Netherlands [XXVIII].

Rendering

The heat map is rendered in the form of a calendar. Time is laid out in two dimensions, where the vertical axis has days of the week and the horizontal axis shows weeks. The labels of the days of the week are shown at the start of each year in abbreviated form. Each month is split up from

the rest using a black border between the days of that month and the next. The horizontal axis displays month labels. The days themselves are squares, where the background color of the square is given a bluish hue. The darkness of the color is based on the number of commits on that day. Below the beginning of the year, a scale legend indicates the highest number of commits on a single day that year.

In addition to the background color, each day may have a temperature bar which makes the bottom part of the square darker, extending further upwards based on the temperature on that day, as shown in the example in Figure 5.22.

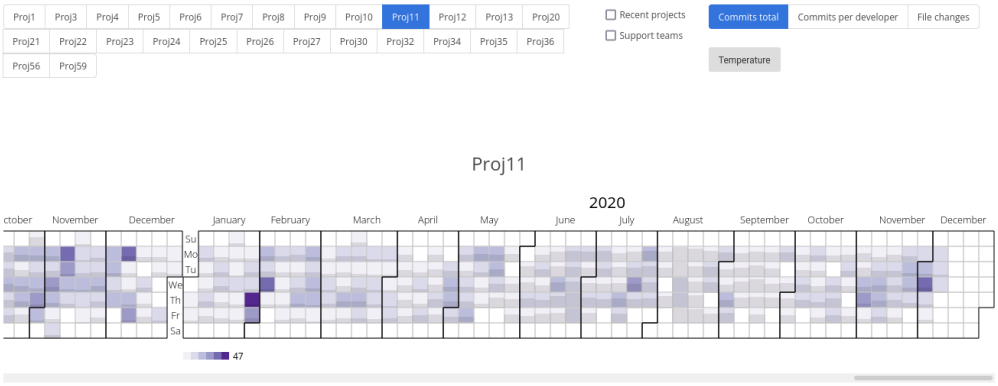


Figure 5.22: The heat map of a project, showing the most recent commit volumes including temperature bars.

Interaction

Similar to other visualizations, the heat map has a project selector which lets the user switch to another project. A pair of checkboxes provide filters for the project selector to only show projects that had recent activity and to hide teams that provide support systems to other teams.

An additional set of toggles select the scale. By default, the total number of commits is used. If the user choose another toggle, we instead color-code the days based on number of commits per developer or the total number of files that were changed. A final toggle allows turning the temperature bars within the day squares on or off.

The calendar is scrollable to the left or right, using a scroll bar, by mousewheel-clicking or dragging, i.e., by pressing and moving, in order to move the calender to earlier and later years. This scrolling behavior was adapted to work on desktop setups with a mouse, on mobile devices and even for use with a remote, such as a Wiimote.

When the user hovers over a day, the metrics known for that day—number of commits, commits per developer and average temperature—are shown. If the user clicks on a square, a list shows up below the calendar, including files from the project's code repositories that were changed on that day, limited to those that were not changed for a long period of time. The changed files are grouped by repository and each group is collapsible. A link to the code repository is provided next to the name of the repository. This functionality is only provided when the visualization is deployed within the organization.

Figure 5.23: QR code of video demonstration of selecting projects and scales in the heat map.

5.6.4 Platform status

An important technical requirement for software development teams in order to produce new features is a development and testing platform that provides reliable stability as well as flexible scalability. Multiple team members should be able to concurrently set up new instances of their software in order to test them separately from each other, without reaching resource limits. If the team does approach limits on disk usage, processing power or network load, to name a few, there should be a way to warn those that are able to restore the platform to a nominal state without much effort. A *platform status* dashboard would indicate the usual workload of the system as well as alert the user of potential problems when a threshold is reached. We could even perform anomaly detection on the data points.

It is beyond the scope of the Grip on Software research project to compare different uses of development and deployment platforms, although our pipeline works well using Docker containers [37] as explained in more detail in Chapter 2. At one of the organizations, ICTU, a Docker-based development environment known as BigBoat [VII] was in use during the time that data was collected from the SCRUM teams—they later moved to another platform. One issue that some teams faced was that this environment often hit resource limits. The platform provided an overview of the current state, including warning indicators when the monitored metrics reached a level that would decrease performance. However, there was no information on past measurements or frequency of failures. The lack of temporal data made it difficult for the teams to pin-point the problems, to relate them with timestamps from build systems and to adjust deployment processes, preventing further downtime.

Data

We collect data from the BigBoat development environment to help find out what events and problems took place. The following metrics are collected for each project for each timestamp:

- Whether the Docker agent is healthy.
- The usage and capacity of various data stores for Docker, including persistent data.
- The number of IP addresses in use within the virtual network.
- The memory that the Docker instance is using, as well as the maximum available memory.
- The system load, which relates to how much of the available CPUs are in use.
- The amount of time that the system has been online.

For the numerical metrics, there is also a threshold value defined by the platform at which the status of the metric is defined as “not OK”. For the metrics with a limit, this is usually a percentage of the maximum value; the threshold for the system load could be set to a fraction of the number of available processors. This “OK” status augments the healthy state metric of the Docker agent.

Rendering

The collected platform metrics are rendered as an aggregated line chart as well as individual line charts. The horizontal dimension of each chart indicates the timestamp and the vertical dimension has various possible axis labels, depending on the metric. The appearance of the visualization is similar to other system monitoring suites familiar to development and operations teams.

The first vertical axis on all the charts is the “OK” status. A line is drawn with a range between 0 to 1. Here, 1 means that the metric is “OK”, while 0 is “not OK”. For the aggregate chart, the axis indicates the reliability of the entire system based on all the statuses, so the value indicated by the line at a certain time is that of the ratio of statuses that are “not OK” at that time over all the known metrics.

The individual metrics also have another vertical axis, except for the Docker agent health metric. Some metrics have an associated unit for this axis, but others do not, such as the number of IP addresses and the system load. If a maximum value for the metric is predefined, it is used as the limit of the vertical axis. Otherwise, the axis ranges from 0 to the maximum value within the displayed time interval. In the two instances of storage and memory, the values are shown in units of gigabytes, with the maximum disk space and memory capacity used for the axis limits.

To make the difference between the two displayed lines more clear, the status indication is drawn using a blue line and the metric value with an orange line. The first axis uses a blue color for the two axis labels to indicate this association. The individual metrics are displayed with a heading and description to explain their meaning. The visualization is headed by the name of the project, a link to the dashboard if it is reachable in the organization and the date and time when the metrics were most recently collected. The entire status dashboard is shown in Figure 5.24.

Interaction

The platform status is available for different projects, like most of our visualizations. A selection control allows switching to another project. In addition, it is possible to limit the charts to showing the most recent week, month or the full project’s life span, using a similar range selector.

It is also possible to zoom into the metrics using the charts themselves. By hovering over a chart, a vertical line with an open circle at the top appears. This line spans from the bottom of the chart to the value at the current timestamp. Moving to the left or right will put the focus on an earlier or later value, respectively. A tooltip next to the line indicates the time, the status and optionally the metric’s value. At the same time, all other graphs show the line but without a tooltip. By clicking on the graph, the line and tooltip remain static at the chosen position until the user clicks again or leaves and re-enters the chart with the mouse.

When the user presses and drags the mouse to the left or right, an area of the chart is selected. After the dragging is finished, the charts will adjust to show the chosen time range. Another set of selection buttons provides a few standard time ranges: the full lifetime of the project, the past month and the past week. The QR code in Figure 5.25 leads to a video with interaction demonstrations of the platform status charts.

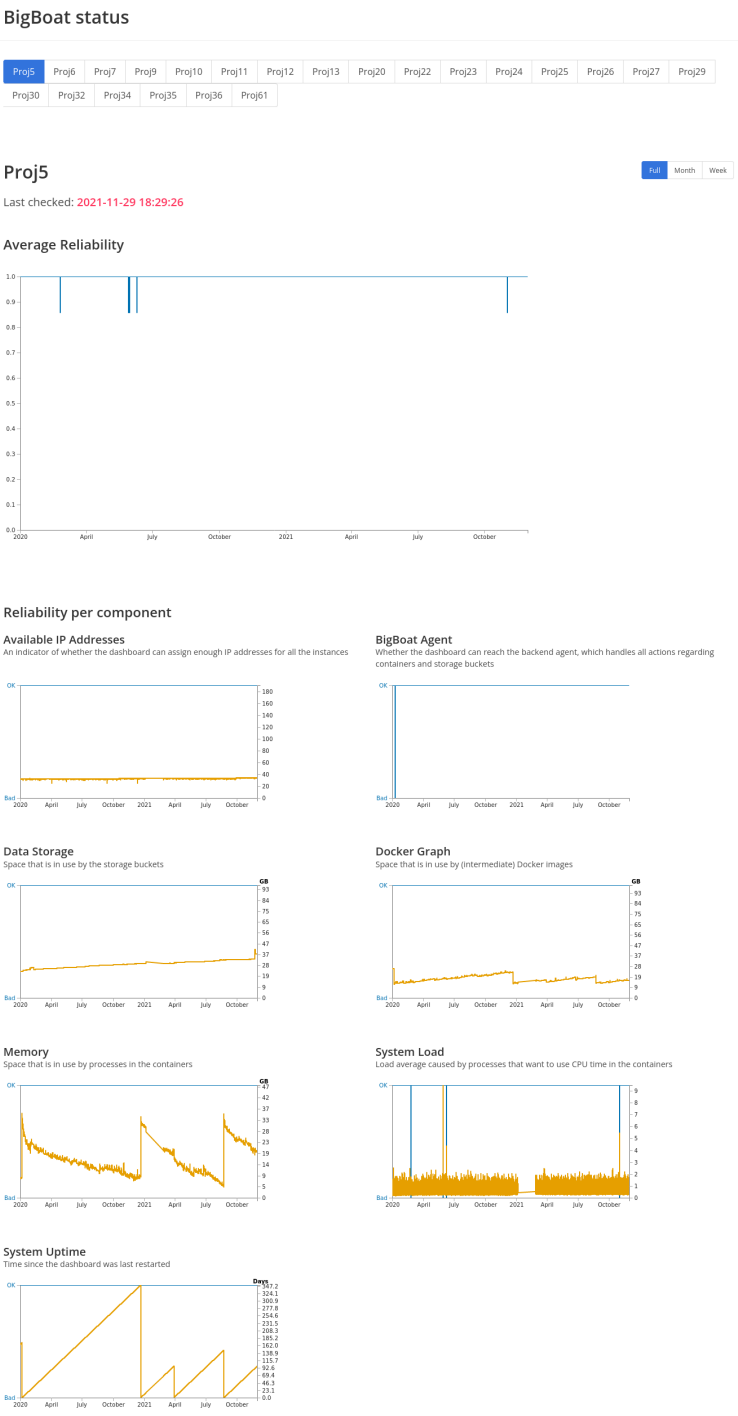


Figure 5.24: Platform status for a project.

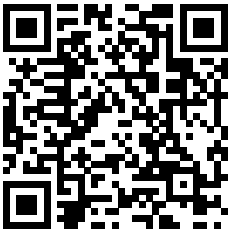


Figure 5.25: QR code of demonstration video of the platform status charts.

5.7 Novel backlog visualizations

Based on conversations and interviews, three additional visualization objectives emerged [117]: (1) to estimate the end date of a project, (2) to reveal patterns in the progress of the software development process and (3) to visualize relationships different types of tasks. We design three new visualizations to integrate further with an issue tracker, in this case as Jira plugins. These visualizations depend on React [XXIX], Redux Toolkit [XXX], Bootstrap [XXXI] and D3.js [111].

5.7.1 Product backlog burndown chart

One goal that team members needed help with was to determine end date estimates for the entire product backlog. During interviews, existing burndown charts for a single sprint, for epics and for releases were discussed. The latter two indicate the likely number of sprints left to work on the user stories that the epic or release links to, based on earlier velocity. A similar chart for an entire backlog is not provided by the issue tracker.

Because the product backlog is an intricate artifact, we define segments of the backlog size during a sprint based on the story point mutations involved with actions on the product backlog. The project's issue tracker provides most of the input data for the burndown chart:

- **Completed work**: The total number of story points completed during a sprint.
- **Discarded work**: The total number of story points on the product backlog from stories that have been closed abnormally and thus have no more work that needs to be carried out.
- **New estimations**: The change in story points from stories on the product backlog whose estimate has been adjusted.
- **Remaining work**: The total number of story points from open stories on the product backlog with no estimate change.
- **Unestimated work**: An automated estimation of the total number of story points from stories on the product backlog that were not estimated by the team by the end of a sprint. Our estimation is based on the average number of story points on the backlog at that time.
- **Added work**: The total number of story points from stories added to the product backlog.

The data collected for each sprint is rendered as a bar chart, where the bars are segmented based on these categories. Later estimations are further split up into higher and lower adjustments compared

to the earlier story points. This gives users a more detailed overview of the reasons for story point mutations. The color scheme is a derivative of the color scheme offered by Jira, taking into account users with color blindness. The sizes of the bar segments are displayed when hovering. Selecting a time span on the horizontal axis results in a view which zooms in on that period in time. The visualization also features a project selection input field and guidelines on the axes.

In order to actually show a forecast for the end date of a project, we provide two ways to estimate and visualize the number of sprints that the development team will likely have to complete after the current date. The first option allows users to extrapolate the total number of future sprints by letting them tinker with the expected average segment sizes. The representation calculates the mean values for the six story point mutation categories. The user can edit these values to determine the effects on the number of future sprints. The second option displays results of simulation of the Monte Carlo estimation algorithm mentioned in Section 4.4.3, as in the example in Figure 5.26.

Figure 5.26: Product backlog burndown chart showing a Monte Carlo simulation of future sprints.

5.7.2 Product backlog progression chart

We find that projecting information regarding past work onto the current situation helps users identify development patterns. In the progression chart plugin, we wish to display the evolution of the product backlog from its inception up to the current sprint. The data for the progression chart is based on the stages that user stories typically go through within the SCRUM framework:

1. The story is added to the product backlog with an initial description.
2. The story is refined. The team estimates the effort and the Product Owner prioritizes it.
3. The story is selected for development.
4. The team works on the story, progressing through the stages of the sprint backlog.
5. The story is completed according to a definition of “done”.

The progression chart displays both real time and earlier project information about these stages. Specific data mutations are conveyed by means of animation through a playback feature. For example, an addition to the backlog is shown as spawning a circle within a sprint, which itself is a larger circle directly contained in the project’s circle. The circle is then inflated up to the size corresponding to its story points compared to other stories. Circles are automatically positioned relative to each other in such a way that they generate the smallest overarching circle possible. The position of a circle within the diagram changes when the story moves to another stage of development. Sprints are given a color based on their recent activity; blue sprints are active. The sprints contain nested light blue, yellow, green and dark gray circles depicting status categories for stories: “to do”, “in progress”, “done” and no status, respectively. The stories within the categories are shaded based on whether they have a story point estimation (light gray) or not (dark gray). Figure 5.27 displays the state of the chart at a specific moment in time for a support team project.

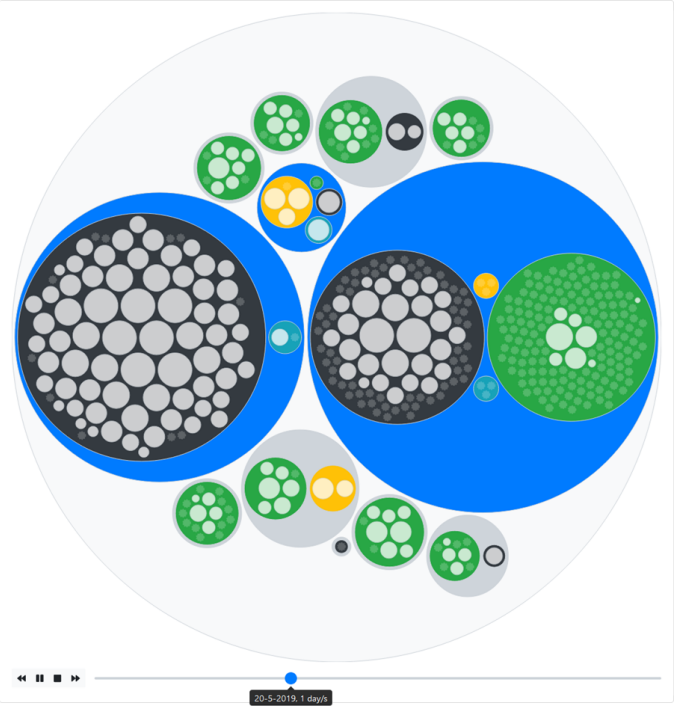


Figure 5.27: Example of the product backlog progression chart.

The playback buttons enable the user to view these changes similar to viewing a timelapse video, including normal playback, pausing to inspect the current state and adjustments to playback speed. Moving the cursor over a circle shows a tooltip with the title of the object and, in the case of a larger circle, the total number of story points it represents.

5.7.3 Product backlog relationship chart

Another purpose of visualizations of product backlogs is to reveal relationships of user stories. The complexity of these relationships is helpful to learn for the development team, as it provides an indication whether there are conflicting dependency relationships. It also encourages refinement, by putting the story in context of other tasks.

The data used for the relationship chart comes from the project's issue tracker. We collect several attributes of the recent user stories and other issues: the key, the type of item, a short description, the story points, the links and the sprints in which they are planned to be worked on.

We visually represent the data using a directed graph. The user stories and tasks are nodes, sized according to their story points in the rendering. Some nodes are connected by edges based on relations made by members of the team. The colors of the nodes correspond to the glyphs in Jira that indicate different types of items: purple items are user stories, light blue items are tasks or subtasks, red items are bugs, green items are improvements, yellow items are spikes and light gray items are technical tasks. Undirected, light gray edges indicate parent–child relations with a subtask, while directed, black edges use other dependencies, such as "blocks" or "duplicates". Figure 5.28 shows the chart for a support team along with a configuration panel.

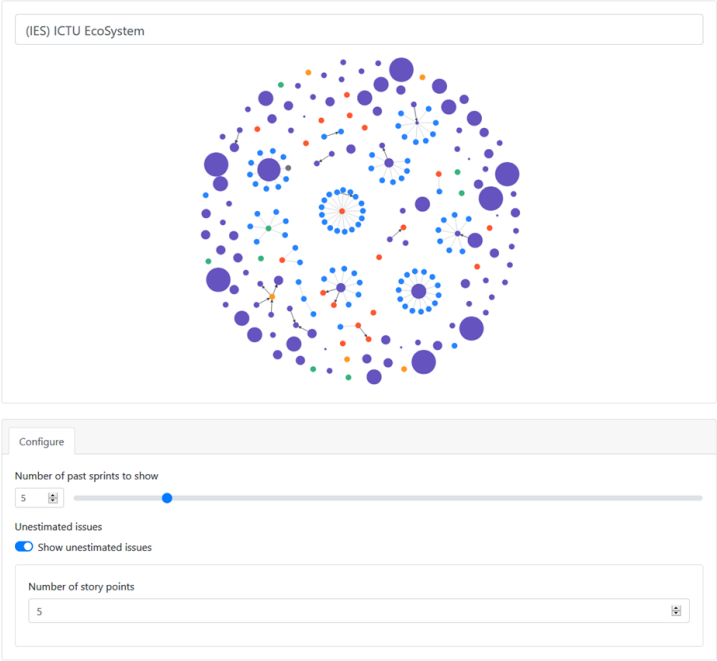


Figure 5.28: Example of the product backlog relationship chart.

The graph layout is force-directed, meaning attractive and repulsive forces between nodes cause dynamic object positioning [XXIII]. In the configuration panel, a slider allows filtering issues to only show those from a number of most recent sprints. If the user enables a toggle to display issues without estimations, a configurable number of story points is used for their node size. The annotation details show up in a tooltip when hovering over a node. When the user clicks on a node, the corresponding item is opened in the issue tracker.

5.8 Evaluation

The information visualizations that are described in this chapter offer an integrated solution to our research questions, by providing the stakeholders with a means to inspect and learn from analyses and patterns from the Grip on Software research. We wish to understand further how to assess the effectiveness that the introduction of these visualizations have on the software development process, in particular how it is adopted by the different teams and roles. We apply objective and automated measurements for the visualizations themselves in Section 5.8.1. Then, we discuss interviews and surveys with relevant parties in Section 5.8.2. Finally, we summarize our review of the information visualizations in Section 5.8.3.

5.8.1 Assessment

Based on the purposes mentioned in Section 5.2 and the design principles that we established in Section 5.4, we determine what kinds of objective aspects an information visualization should adhere to in order to properly be able to use it within a software development organization. Usually, a more verbose specification of requirements would be a starting point, but in our case the visualizations are provided as a proof of principle that our approach of making curated data and results accessible to the stakeholders has merit. The dashboard and visualizations are not market-ready products where such studies would be a non-functional requirement. Still, we consider the following objective criteria when evaluating our visualizations:

- **Similarity**: Functions that a visualization provides to enable access to certain controls or subselections of data, should behave the same was as the function that does this in another visualization on the dashboard. Examples of these functions are: selecting a certain project to display, comparing it to other projects, zooming in on temporal or coordinate data, displaying additional data via tooltips or exporting the data.

- **Traceability**: It should be possible to find out how a certain data point was determined, by providing information on how it was calculated or by adding a link to the source of the data, sometimes referred to as data provenance.

- **Tests**: Various portions of the visualization should be able to be found and interacted with using an integrated test, where a browser is controlled via an automated system which attempts to visit a web page on an isolated HTTPS website. It is then instructed to click specific buttons or other controls. The test then validates if certain HTML elements or other content is found on the page. Being able to emulate user input is also an indicator of perceptive difficulty, as visualizations that require fewer steps to work with are also easier to test this way.

- **Mobile and large screens**: The visualization should be able to adjust to devices with different screen sizes and input methods, such that the views and controls are still visible and operable. We should also account for setups where a mouse and keyboard are replaced with other input methods, such as touching or with a remote.
- **Accessibility**: The visualization should be usable for people who require the use of screen readers or other aids in order to understand the contents of a web page, such as our visualizations.
- **Color blindness**: By extension of the accessibility aspect, people who see differences between certain colors less clearly should be able to interpret a visualization to obtain knowledge from it. Information conveyed by color should have another means of description, or a color scheme adapted for color blindness should be selected.

The aspect of similarity is partially covered by the integrated dashboard with common UI elements, such as the navigation. Many other elements within the visualizations also share a common ground, e.g., a project selection control and tooltips for longer descriptions. Still, some of the controls are left out or given different functionalities across the visualizations, so for a number of relevant controls we compare their use, with the results displayed in Table 5.2.

Visualization	Selection	Compare	Zoom	Tooltips	Export
Sprint report	✓	✓	✓	✓	✓
Prediction results	✓	✗	✗	✓	✓
Timeline	✗	✓	✓	✓	✗
Leaderboard	✓	✓	✗	✗	✗
Collaboration graph	✗	≈	✓	≈	✗
Process flow	✓	≈	≈	✓	✗
Heat map	✓	✗	✓	✓	✗
Platform status	✓	✗	✓	✓	✗
Burndown chart	✓	✗	✓	✗	✗
Progression chart	✓	✗	✗	✓	✗
Relationship chart	✓	✗	✗	✓	✗

Table 5.2: Comparison of common functionalities of the visualizations (✓ is good, ≈ almost, ✗ bad/missing).

We note that some visualizations do not share a common control because they do not require some functionality. For example, the timeline and collaboration graph show all projects from the data set, thus they do not have a means to select a project. A function to filter projects could however be a desired feature, so it is relevant to denote its absence. There are slight differences in how the projects are selected or filtered as supported by metadata. The product backlog visualizations use a different control to search and select a project. Overall, there is consistency and we prevent the controls from being overcomplicated. Many visualizations are limited in their capability to compare projects, often because the focus is on one project at a time.

An important design principle that we started out with is to primarily display data relevant to the patterns that we wish to present. Many of our visualizations are built around this principle.

Still, some users want to find more details, either temporally or within another coordinate space of the visual form. To do so, a number of visualizations offer a control which zooms the view, leading to a different selection being shown. Some visualizations use buttons or scroll wheels to zoom in and out. Another method is to select or adjust a rectangular area to zoom into using a "brush" or within the area itself, with another gesture to zoom out. The collaboration graph only zooms in when a search takes place. Each control is different, but is often indicated using a button, a mouse cursor or a search field. This addresses the need for this functionality.

Another method of making both information and controls more insightful and clearly labeled, while avoiding clutter on the page, is to place these details or guides in tooltips. Many visualizations use this, albeit in slightly different formats and situations. Most visualizations label the controls with longer descriptions. Some have larger tooltips that indicate more features for a project or sprint. However, in the collaboration graph and process flow, only small tooltips are used for some details, which is less usable and accessible for many users.

Only the sprint report and the prediction results have controls that export that data shown in the visualization in different formats. The integrated dashboard optionally shows a download link for each visualization. An exported version is helpful for offline inspection, reporting and standalone presentation. Specific export controls are buttons which open the selected format in another browser tab or provides a direct download, depending on browser settings.

Most controls and rendering approaches will feel familiar to users, such as buttons, checkboxes, sliders, graphs and charts. We consider these effects of similarity when it comes to adoption in Section 5.8.2. The other objective criteria are laid out in Table 5.3 and we discuss further how some visualizations fulfill these or not.

Visualization	Traceability	Tests	Mobile	Accessibility	Color
Sprint report	✓	✓	✓	✓	✓
Prediction results	✓	✓	✓	✓	✓
Timeline	≈	✓	✗	✓	✓
Leaderboard	✓	✓	✓	✓	≈
Collaboration graph	✗	✓	✓	✓	✓
Process flow	✗	✓	≈	✓	✓
Heat map	≈	✓	✓	✓	✓
Platform status	≈	✓	✗	✓	≈
Burndown chart	≈	✗	✓	✗	≈
Progression chart	≈	✗	✓	✗	✓
Relationship chart	✓	✗	✓	✗	✗

Table 5.3: Comparison of assessment criteria regarding the visualizations (✓ is good, ≈ almost, ✗ bad/missing).

With regard to traceability, we have shown that some of the visualizations provide details on how features were collected and extracted, as well as providing links to where they come from when the visualization is deployed within the organization where access to the sources is possible. Often, there is only a link for specific entities, such as a sprint, a code repository or a Docker dashboard. Only the collaboration graph and process flow do not have links to their sources, making it harder to trace back their metrics.

All visualizations have tests that ensure that controls and expected rendering function properly. The only exceptions are the Jira plugins, which do not have automated test cases to perform these actions mechanically. The large majority of actions and gestures have been tested in this way and shown to be reproducible in this integrated setting.

When it comes to different screens and input devices, most visualizations work well in both small, mobile formats as well as in a larger lay-out. Additionally, we ensure that gestures that are important for the functioning of the visualization remain possible with touch or remote, although a small number of touch methods are not available when they conflict with normal scrolling behavior. For example, the leaderboard, where the cards properly fit a device size, cannot make use of dragging to create new normalization factors. The collaboration graph also provides slightly less interaction. The main issue for the nonconforming visualizations—the timeline, process flow and platform status—is the size of the charts or graphs that are displayed, which become too small when scaled down.

As part of our integration tests, we also use an automated accessibility testing engine [XXXII] which checks for common practices regarding web page structure and is able to find the majority of issues that could lead to failure compared to the relevant accessibility guidelines [XXXIII, XXXIV]. Any failures are aggregated and reported based on the tests. Meanwhile, browser extensions track them in development environments.

To check for potential problems for color blindness, the accessibility tests perform contrast ratio measurements between foreground and background elements, but this is not a sufficient test. We find that most visualizations use proper color schemes and/or glyphs with textual labels to differentiate colored data. The ranks used in the leaderboard use green, yellow and red, which is technically not enough to differentiate the different categories, although the rank numbers provide an indication. The platform status indicates collection dates with green and red. The Jira plugins use some colors that do not go well with another in the same rendering, despite an attempt to improve upon the original color scheme.

Overall, the visualizations provide an integrated, familiar, accessible and clear experience, putting the focus on the information being displayed. By sharing interface controls and applying them in similar ways, users are able to learn how to use a new visualization more quickly and to customize it according to their needs. Through tests, we are able to ensure that the visualizations remain available according to these criteria. This also clarifies goals of further development of a data hub for developers. Improvements to the functionality easily fits along with the existing situation.

5.8.2 Adoption

During the Grip on Software research project, we performed a limited amount of evaluation of usability and adoption among representative users. For three early-developed visualizations, namely the timeline, collaboration graph and heat map, we conducted a survey including a System Usability Scale (SUS) questionnaire. This provided some insight in the potential of these visualizations and steered the focus afterward.

The questionnaire showed that the respondents found the visualizations to be easy to learn and easy to use. One point of contention was that it many users would not be frequently accessing any of the three visualizations during their usual work routine. A likely factor that the questionnaire accentuated was that, at the time, the three visualizations did not make clear what certain elements were meant to do [118].

Based on more meetings, discussions and presentations, we focused on integration of new visualizations, predictive patterns and situation reports. The resulting visualizations are an iterative effort, where prototypes were displayed to stakeholders for further feedback and desires. This included exploring which kinds of attributes were more relevant to them. We often enrich the result rather than leave out the data that was found less interesting, so that it is still accessible through a tooltip, for example. In order to ensure that features exhibited data that users felt familiar with, we collaboratively looked for reasons where data did not adhere to expectations and adjusted filters and other conditions. This also improved the attributes for the prediction and estimation algorithms, as the input data more accurately follows the events that they described.

Specific care was taken for the information visualizations with regards to usability. Users tested whether they could acquire the knowledge provided in them, including users with color blindness. The novel structuring of data allows some users to extract knowledge that they could not easily find before. Providing existing data in fresh formats through intuitive interactions helps users find their way more quickly [119]. In particular, the sprint report and prediction results were used by various Product Owners, software delivery managers and quality managers for multiple teams. The collaboration graph's overview and timelapse mode was seen as helpful at an organizational level, as well as a nice way to showcase the organization's own history.

The predictive power of the sprint report's future sprints as well as the backlog burndown chart has helped team members in leading roles to find out when a project would finish if no additional stories are added to the backlog, or when additional features need to be suggested by the client in order to maintain enough volume. Based on exploratory interviews with users that have various roles in the organization, an inventory of missing insights was formulated. These ideas further influenced the design of the novel backlog visualizations.

5.8.3 Conclusion

In this thesis, we study patterns and analyses regarding the aspects and iterative results from SCRUM software development processes. This chapter focuses on delivering the research output to the relevant stakeholders, including development team members, Scrum Masters and Product Owners. We aim to do so in an intuitive way that integrates with the existing development ecosystem. We create a dashboard of multiple information visualizations to make our results accessible to the aforementioned stakeholders, highlight different types of data and ease the discovery of useful patterns. We provide sprint reports, prediction results, timelines, leaderboard ranking, network graphs, flow graphs, heat map calendars and platform status monitoring. Finally, we design various backlog visualizations that integrate with Jira, an often-used issue tracker.

These visualizations are designed in a way that they meet certain requirements when it comes to applicability within the software development framework. The goal is to improve the process, by supporting the stakeholders in finding out what went well and what can be improved. The information is provided with clear labels that describe what it means. Focus is placed on the important factors, but it is still possible to inspect details and trace sources through zooming, clicking links or expanding elements [109].

The view and interaction of each visualization is based on considerations for colors, icons, glyphs, gestures, interactions and controls. The rendering of the visualization is based on a visual form, such as a graph or a collection with relationships between certain fields. The visual form is defined by the type of data that we select and filter from the data pool, which includes the Grip on Software database, results from prediction algorithms and external data sets.

The information visualizations produced by the Grip on Software project are available as code repositories and a live website is publicly accessible for research purposes[††]. The visualizations on this dashboard use anonymized data; teams, projects and people have no identifying information. Moreover, there are no links to the systems where the data is collected from. Versions of the dashboard with descriptive identifiers and source links are available to the organizations that were involved in the research, namely ICTU and Wigo4it. The Jira plugins with novel backlog charts are also deployed at ICTU. We consider the application of these visualizations in two organizations to be part of our case studies into their specific workflows, while the produced results and evaluations make them relevant for wider application [120].

We introduced automated metrics in our assessments through unit tests and accessibility tests, in addition to other objective metrics and user feedback [121]. Together, the are a part of our iterative design and construction process for the visualization phase of the GROS pipeline.

Selecting a proper visualization to support the user in making an informed decision within a process is sometimes difficult, especially when there are many potentially relevant metrics and features to provide to them [122]. For our visualizations, we have taken some design principles in mind, which include familiarity, relatability, simplicity and intuitiveness. These principles mostly revolve around keeping the default view for the user focused and uncluttered, while providing details when a clearly labeled control is interacted with. Each visualization has different formats that were chosen to fit with the type of data.

The application of information visualization within Agile software development methods is an area of research that has not been extensively explored. Most earlier work focuses on visualizing specific aspects or abstract flows. The integrated dashboard provides new opportunities for stakeholders to obtain the knowledge to improve their processes.

In conclusion, we demonstrate that it is worthwhile to provide results from predictive models regarding SCRUM software development in the form of information visualization to those involved in the process. Further research could show how the adoption of the visualizations affects long-term decision making, for example regarding backlog planning. Finally, a full usability specification, study and survey—with an updated SUS questionnaire involving more of our visualizations—helps position the use of visual interaction systems within Agile software development frameworks even better.

[††] `https://gros.liacs.nl/visualization/?lang=en`

Chapter 6

Discussion

Contributions of the Grip on Software research toward predictability of the SCRUM *software development process*

Abstract of Chapter 6

We finalize the Grip on Software research by reflecting on the problem statement, research questions, implementations and results discussed in earlier chapters. Particular consideration is given to the technical pipeline that made our studies possible. This includes components for agent-based data acquisition and database storage with a query template compiler for feature extraction. We also implemented state-of-the-art prediction models and validation methods that take temporal aspects of the data set into account. The research output is established by novel, integrated information visualizations.

We consider the design and architecture of our distributed research system to be part of our main contributions to the field of predictive software engineering. This is because they form a feasible approach to help explain how a SCRUM software development process is progressing. Our problem statement involves understanding how the analysis of events from this Agile process can be used to significantly improve predictability of sprint completion and backlog progression. We address these themes and problems through several research questions and sub-questions.

We conclude the research by answering these questions. The acquisition and storage of data relevant to the SCRUM process becomes reliable using our pipeline components, including MonetDB as a column-based relational database, for which we introduce a query compiler. For pattern recognition, we introduce several models for sprint workload classification and estimation as well as project backlog size estimation with reasonably accurate results. This analysis stimulates short-term and long-term planning efforts during the development process. Through information visualization, we provide understandable and usable visualizations which we evaluate through hybrid testing and survey assessments, showing promising adoption of several integrated visualizations that support analytical decision making and development ecosystem management.

The Grip on Software research project shows potential for future additions and follow-up studies, both in improving the quality of data and expected results for analyzing particular situations in SCRUM, but also through generalization toward other software development methods or reusability of implementations in future data-oriented research.

6.1 Retrospective

This chapter provides a recap of the Grip on Software research outlined in full detail across this thesis. The research focuses on understanding how to extract, analyze and explain events that take place during a software development process, which kinds of predictive algorithms are applicable to this kind of data and whether this allows teams to learn from factors that indicate the progress thus far. Specifically, we consider development cycles based on the SCRUM framework [6], with a total of 60 projects and teams from two Dutch governmental development organizations, ICTU and Wigo4it. Our focus of this chapter is to provide an overview of the results of this research.

As software development is a complex process, we have found that it is not easy to provide a single, encompassing conclusion about how development teams should conduct their work. In our research, we specifically focus on SCRUM, while remaining impartial on its efficacy compared to other software development methods. Keeping in mind the values posed by the Manifesto for Agile Software Development [3], we consider that there is no immediate need to outline the method's strengths and weaknesses, but rather to focus on potential usability. Rules that seem inherently contradictory on the surface in fact reinforce each other; for example, without processes and supporting systems, team interactions would become chaotic. In a similar spirit, additional predictability of the process—by planning ahead—should allow teams to respond to change.

Our main objective is to improve this predictability. This leads to a scientific approach in a situation where data is available, but scattered and not immediately ready for in-depth analysis. Through novel implementations of state-of-the-art models, we design a generalizable pipeline for secure data acquisition, analysis and explainable output. This is a cornerstone for providing accurate results through different methods of predicting short-term and long-term effort in SCRUM.

In this chapter, we address the results of the Grip on Software (GROS) research in different phases. In Section 6.1.1, we first discuss the GROS pipeline from a technical perspective. We then describe the main contributions per chapter in Section 6.1.2. Next, we reflect on the conclusions in Section 6.2, with the research problem addressed in Section 6.2.1. Specific points regarding the research questions are detailed in Section 6.2.2.

We wrap up this chapter and the body of this thesis with suggestions for potential future work that extends upon our study in Section 6.3. This includes possible research directions in Section 6.3.1 as well as generalizability of the approach—including applicability to other development methods and research endeavors—in Section 6.3.2.

6.1.1 Technical overview

Throughout our research, we have created and used a unified pipeline which processes data that we collect from software development projects. The steps that we perform in this pipeline are important to our needs and relevant to discuss on their own.

In Figure 6.1, we display a schematic overview of the components of the pipeline. In gray, we show some typical systems used in software development: an issue tracker to manage project backlogs with tasks like user stories, a version control system that holds code repositories and a quality metrics dashboard for checking coding practices. These are part of a software development ecosystem present at each organization. The red arrows denote our components that acquire data from these systems, import the data in a structured manner and extract features for further research purposes. Blue blocks in between the arrows depict intermediate, persistent artifacts of our pipeline. Finally, the green blocks represent the research output at the end of the pipeline.

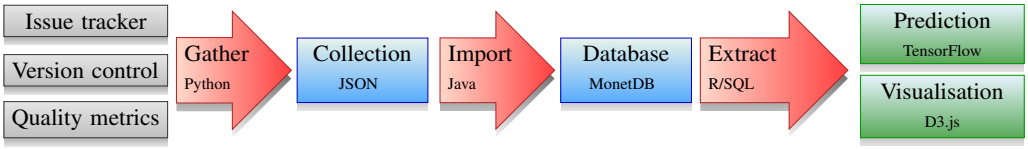


Figure 6.1: The full Grip on Software pipeline. The primary data sources (gray), processing components (red), intermediate artifacts (blue) and resultant components (green) are shown here. The components are constructed using various frameworks, modules and programming languages.

Each component is built with its own objectives in mind, with an overarching idea that the functionality of all the components is directed towards our data needs in the GROS research project. We describe the components in each chapter from a scientific usage perspective and clarify how we are able to employ the data in our studies after processing and transformation by the component. Even though the technical details of the components are not the primary subject of this thesis, they do pose a relevant topic when we augment the components with new techniques. Some of the novelty of the components is mentioned in Section 2.4.4 of Chapter 2. In Table 6.1, we indicate the chapters of this thesis and recap on what each of them describes. The component and intermediate artifacts are discussed from an applied, scientific point of view. In addition to these chapters, we briefly reflect on the technological improvements in this section.

PART	GATHER *Collection*	IMPORT *Database*	EXTRACT	PREDICTION	VISUALIZATION
Chapter 2	✓	≈	≈	≈	≈
Chapter 3	✗	✓	✓	✗	✗
Chapter 4	✗	✗	✓	✓	✗
Chapter 5	✗	✗	✓	✗	✓

Table 6.1: The chapters that each component or artifact of the pipeline is described in (✓) or not (✗). Some components are featured in multiple chapters or are only discussed briefly from a technical perspective (≈).

We summarize the technical work as part of our research, focusing again on the aspect of innovative approaches, e.g., in the field of data acquisition, pattern recognition and information visualization, in particular when it comes to handling data from Agile software development. We highlight specific aspects of each component in this section and describe why they are relevant for our research.

In the data acquisition component, our agent-based design supports the deployment of this portion of the pipeline in software development ecosystems that we encounter at different organizations, taking in account virtualization and networking. We configure the agents to access systems containing data from the development process. Each agent independently handles processing, including early encryption of personally identifiable information.

We model the collected data to fit in a consistent database model. We store the entities and relationships in a persistent, column-based tabular format in a MonetDB [38] instance. Through encrypted exports, we safely bring data together from multiple organizations at a central analysis location.

Feature extraction takes place using a set of query templates and reusable definitions that describe how to calculate velocity and other factors. We compile the templates into efficient SQL that MonetDB is able to further optimize. This also enables us to keep the templates agnostic to portions of our database model that are similar to each other. Configurations for each use case determines the relevant entities to select, based on their presence in the respective development ecosystems. This way, we obtain a consistent data set where sample features allow comparison and augmentation of models.

The prediction component provides novel methods that track the temporal data which was used for learning, while implementing state-of-the-art as well as well-founded methods, such as neural networks (NNs) and the analogy-based effort estimation algorithm (ABE), using TensorFlow [XI] for efficient, vectorized operations that are accelerated using GPU devices.

We finally produce multiple information visualizations accessible through an integrated dashboard plus several plugins for an issue tracker. We use a common basis of frameworks, such as D3.js [111], which allows us to build data-driven visualizations. We implement newly-designed controls for navigation, selection and full-page localization rendering. This leads us to the creation of various interactive visualizations. Each of them focuses on particular topics in software development. Meanwhile, their shared resources mean that it is easy to learn and switch between the individual views, leading to quicker adoption by stakeholders.

6.1.2 Main contributions

Our contributions to the scientific field extend beyond technical components of a pipeline for data acquisition in the Agile software development domain. One of our main objectives is to maintain a generalized approach in the Grip on Software research project. The modular design and data-driven method should work in other situations as well. We further consider the generalizability of the research in Section 6.3.2.

Through the individual sub-projects in GROS, we achieve and report on results regarding feasibility, performance, accuracy and adoption on progressive checkpoints of our research. This leads to the eventual result of providing traceable, interactive and explainable outcomes from predictive models. These models focus on classification and effort estimation of SCRUM sprint commitments as well as long-term product backlogs.

Before we consider the final results, we reflect on the advances of each chapter of this thesis on its own. We discussed the GROS data acquisition pipeline from a technical perspective in Section 6.1.1, but its introduction in Chapter 2 also concerns design patterns, modeling and performance, among others. The distributed, agent-based pipeline is a standalone contribution, but is inspired by work on distributed data systems, Petri nets and business analytics. We design the computational part based on these concepts. We also assert that a clear communication protocol is relevant, and introduce schema-based modeling to aid this. Meanwhile, we retain flexibility of placement of agents in the development ecosystem. Reusability of this component is also improved through continuous integration of development and documentation. The observed performance of the pipeline indicates its suitability for existing development platforms, without hindrance to the development team's usual process.

We continue our model-based principles in Chapter 3, where we introduce the GROS DB as our database system. Parts of the eventual database model that are thematically related due to their source system are explained. We further describe new relationships between entities from different systems. This also includes linking separate profiles when those profiles refer to the

same person, while preserving privacy and security of encrypted data. The introduction of a query template compilation system is followed by a performance analysis of the chosen system. The experiments show that our decision criteria—which led us to select MonetDB—turned out to provide a feasible, practical application.

Chapter 4 focuses on the pattern recognition problem, including extracting relevant features, training machine learning models and estimating short-term and long-term outcomes of SCRUM life cycles. This is the core of our research contribution. We discuss methods of selecting intuitive and essential feature sets using scoring, leading to a consolidated data set representing multiple organizations, teams, projects and sprint samples. Our new data set is split up into training, test and validation sets in a time-preserving manner. This approach allows models to learn from earlier samples while tuning and comparing accuracies based on later sprints. We introduce the tasks of sprint result classification/estimation and backlog size estimation, with models that are tuned for these purposes. The subsequent experiments with different data set sizes, parameters and scenarios show the effectiveness of applying these models in Agile software development planning.

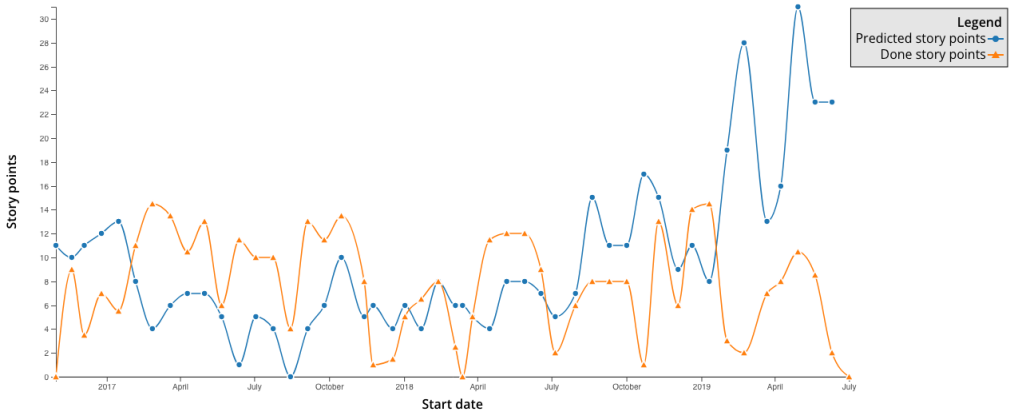


Figure 6.2: One of our information visualizations, the sprint report for a single project, showing effort estimation outcomes from predictions (blue line) compared to actual story points that the team completed (orange line). This is a way to visually represent analysis effort for experiments with limited training set sizes.

Our contributions culminate in the creation of information visualizations in Chapter 5. We make these visualizations available to the stakeholders as well as in anonymized form for further research. We produce visualizations for analytical decision support—the sprint report, prediction results, event timeline and leaderboard—as well as interactive views with an organizational ecosystem management theme, namely a collaboration graph, process flow chart, heat map and platform status dashboard. Three additional plugins are created for displaying product backlogs in new, interactive ways, namely predictive burndown charts, nested progression charts and relationship charts. Our work on extracting relevant patterns tightly connects to the visual format of the visualization hub, making the work more insightful. For example, we compare actual sprint outcomes with results from the analogy-based effort estimation across earlier sprints—by performing predictions with different training set sizes—through inclusion of both the feature and outcome in the sprint report visualization in Figure 6.2. Our methodical approach towards

improving the visualizations also leads to the conception of a hybrid assessment methods, in which we evaluate both automated tests regarding accessibility as well as input from surveys and interviews with stakeholders. This way, we validate whether our goals are met regarding the individual information visualizations.

6.2 Overall conclusion

We reflect upon our earlier introductions of the encompassing problem statement and specific research questions in this section. Based on our contributions—from a technical, scientific and organizational perspective—we conclude that our research implements state-of-the-art concepts, produces a novel data set with promising results and benefits multiple stakeholders that are active throughout the software development process.

We performed feature extraction, selection and scoring to obtain our representative, traceable and robust data set. We applied several models for the problems of classification and estimation of short-term and long-term planning in SCRUM processes. Throughout our experiments, we consider the performance of the models in this domain. We evaluate the models not only based on accuracy during the test and validation, but also by looking into effects of data set dimensions and various parameters, such as simulation scenarios.

Aside from the main problem statement in Section 6.2.1, we also discuss our specific research questions in Section 6.2.2 regarding separate focus areas of the Grip on Software research. This includes application of our pipeline in data acquisition and storage domains, the particularities of the pattern recognition analysis and finally the information visualization element.

6.2.1 Problem statement

The following problem statement is central to our research: “How can extraction and analysis of measurable events during a SCRUM software development process as well as other qualities of the product and team be used to significantly improve the predictability of practices employed at software development organizations?”

From the beginning of our research, we looked into what data would be useful to explain factors that contribute to the success of a SCRUM software development cycle. We focus on data that is made available through the use of project trackers, development platforms, quality checkers and other systems in software development. There is a constant generation of data in this development ecosystem. Due to this, team members and other stakeholders are not equipped to inspect all of the data, manually or otherwise. Thus, it is difficult to maintain an overview of the process or to spot specific problems. This is why we determine that automated data acquisition and analysis through machine learning benefits the development process as a whole.

Due to the repetitive nature of SCRUM, our underlying hypothesis is that the data exhibits relevant patterns with emerging properties of both short-term SCRUM sprints and long-term product backlogs. This would allow us to derive consistent and compact definitions for machine learning features, thus abstracting away from potential behavioral differences between teams and organizations. Meanwhile, we retain the characteristics that describe the progress in an unbiased manner, including potential alternative properties for effort estimation like number of comments

and code commits. This yields a data set that is suitable for solving classification and estimation problems, contributing to the predictability aspect of our research question.

By focusing on collecting and analyzing the measurable events of SCRUM, we ensure that the research remains objective and explainable. We note that some input is based on expert judgment, for example the assignment of story points to the work items planned for a sprint. Still, we are able to perform comparative analysis of those attributes through various feature extraction methods. Using novel approaches to existing software engineering metrics regarding sprint velocity, we obtain to a form of "exchange rate" that streamlines the input data across teams. We take much care in tracking the origin of features used in our data set to fulfill our explainability objective. When we provide our research output to stakeholders, they are able to find out what the features mean. For some estimation models, we also indicate which sprints were similar to the one being estimated. On the other hand, personal data is encrypted early and only plays a role in properly modeling relations between user profiles, while the eventual data set itself is anonymized.

An important factor in increasing the impact of the Grip on Software research project is to move beyond just providing data. Even if metrics are annotated or provided in graphical form, more steps are necessary to ensure that end users learn from observations and results through interaction. We provide different visual forms of various parts of the curated data, which allows stakeholders to view representations of focus areas that are part the development ecosystem. By interacting with these visualizations, they are able to make decisions based on configurable reports that focus on relevant factors. Transformations based on this interactivity reveal how situations compare to each other and accentuate differences between teams and organizations. At the same time, we decrease the cognitive load when the user switches between visualizations through common control elements—interacted with through simple gestures—in the interface.

6.2.2 Research questions

From our principal problem statement, we deduced several specific research questions. Each of these three questions focuses on a different facet of the GROS research, namely (1) data acquisition as well as storage, (2) pattern recognition and (3) information visualization. These themes also describe the natural flow from data extraction to usable research output while preserving explainability. The same process is embedded in our component-based pipeline shown in Section 6.1.1. Similarly, our thesis is constructed around these three research questions, each with four sub-questions, with the first research question split across two chapters. In this section, we consider each question and how we answered them in those questions, by summarizing the results obtained for each segment.

Data acquisition and storage (Chapters 2 and 3)

RQ1 How can we reliably collect data regarding SCRUM software development practices and consolidate the resulting artifacts inside a central database that constantly grows and allows adaptable queries?

The first question that we considered focuses on the technical aspect of the research. We address the problem of collecting and safeguarding large-scale, incremental data streams by means of our contribution of the pipeline. In particular, the data acquisition and database components perform

these tasks and feed the necessary input for solving the remaining objectives regarding prediction and visualization. Both steps mentioned in the question, i.e., data acquisition and consolidation, are important in this research field for a proper, curated approach. We design and construct our pipeline according to a structured, modular method for performing common quantitative and qualitative tasks to make the research work in diverse networks of systems.

We split up this question into several points regarding the two components, in order to further understand how to approach data acquisition and storage.

RQ1a Based on relevant design principles from distributed data systems and agent-based networking, how do we design a data acquisition pipeline applied to multiple organizational ecosystems?

We compose an agent-based network design, in which our distributed systems regularly check for updates of data in issue trackers, code repositories, quality dashboard and other systems that they are configured to access. This implementation works well in organizations that have multiple projects, each with their own virtual or physical networks. An agent-specific configurator allows setting the proper credentials and access settings for each data source. We implement specific technologies to access various APIs to obtain updated source-specific data, convert the data to standardized forms based on JSON exchange specification schemas and upload the new data via SSH to a centralized controller for further processing. We perform experiments of this agent-based setup in multiple projects using different development ecosystems to ensure that the activity of the pipeline components did not impact the performance of their usual process.

RQ1b Which objectives related to inspection, curation and disclosure do we achieve with a data acquisition pipeline when taking our formulated requirements in mind?

During the data collection, we monitor the incoming data through inspection of intermediate artifacts and query-based reporting of the database status. Additionally, we deploy a status dashboard that displays health information of all the distributed agents and collects logging information at this centralized location. By sampling situations from the database and looking for uncommon situations, we perform curation of the data. One example is an adjustment of how story points (SP) are counted when nonstandard values are filled in by developers to indicate that the story is not yet ready for development. Personal data is encrypted as soon as possible, while still allowing curation of duplicate accounts through links as explained for **RQ1c** hereafter.

Finally, disclosure of the data to stakeholders is provided by the later pipeline components for pattern recognition and information visualization, which integrate with the database component, making use of template-based feature extraction and data aggregation, respectively. This allows the data acquisition component to restrict its scope to just the input side of the pipeline, with the database acting as an intermediary for further access of the results. This modularity is an important factor for potential reuse of the pipeline and helps with further dissemination of all artifacts produced during the data processing.

RQ1c How can we model the relationships between different data artifacts acquired from dynamic systems regarding SCRUM software development in order to properly deduce information about their state during different sprints?

The database component is realized by MonetDB, a column-based relational database management system (RDBMS), which is well-suited for large-scale analytical information processing. Entities are stored in the database as rows that are part of tables, with properties in their columns. Relations between each entity are either additional properties or separate tables, depending on the type and cardinality of the relation. Using a column-based storage, accessing details of many entities at once is highly performant.

Based on the entities and relationships acquired from the source system, we model additional references. These link entities that already existed in the same source system as well as between different parts of the schema. Our focus here is increasing the descriptiveness of our main entities, oriented around projects and sprints. Sometimes, this is accomplished by performing inference of events that took place between a sprint's start and end dates. This allows us to uncover what has happened in a sprint through queries based on well-defined time sequences.

We further address privacy concerns during linking of the same person that exists across systems in the development ecosystem. We use project-specific and global encryption keys for personal data, allowing us to relate the same entity with different profiles without being able to read its properties—unless the original data was already available. The encryption keys remain at the organization where the sensitive data is acquired, with only the pseudonyms available during our analysis.

RQ1d Which technical challenges are relevant when deploying a database component as part of a pipeline for multiple organizational ecosystems?

The architecture of the database component is suited for different development ecosystems. We set it up as part of the pipeline at the development platform of a participating organization, while we also maintain a central instance for research purposes. The resource usage of the component is limited through the implementation of batched update statements during imports as well as optimizations of queries.

Meanwhile, we increase the flexibility of the feature extraction queries through the introduction of a template compiler. The compiler dynamically adapts the queries to the usage of systems by the organization. This means that we obtain similar sample data for machine learning features regardless of the type of project management system that tracks stories, work items and other tasks. Currently, both Jira and Azure DevOps—formerly known as Team Foundation Server (TFS) and Visual Studio Team Services (VSTS)—are fully supported in this way, while entities from GitHub and GitLab have been modeled likewise. Despite limited use of the issue tracking functionality of these two code review systems at the two organizations, their modeling in our database demonstrates potential for generalization of the technical approach.

Experimental results show that both our manual improvements to the template queries and optimizations provided by a hot-started MonetDB instance help with speeding up the queries. This enables us to perform rapid, frequent updates of data for the pattern recognition and information visualization purposes later on in the pipeline.

Pattern recognition (Chapter 4)

> **RQ2** How can we improve the predictability of SCRUM software development practices, specifically the progress of sprints and backlogs—based on analysis of data selected from the development process—and how do we validate our approach?

Our second research question concerns the core of our scientific approach, namely the machine learning models and related pattern recognition methods. From our results, it becomes clear whether each method is feasible, explainable and usable by stakeholders. We present adjustments and improvements of existing models to work on the specific problems of predicting outcomes of sprints and progression of work on backlogs. We extract features for our data set and select groups of them that hold descriptive qualities of the sprints that they originate from. By applying common means of validation on our models, we observe the accuracy of the results and understand their behavior. Stakeholders are then able to make use of these results, supported by explanations on how the machine learning models arrive at the conclusion of the provided labels. These then help with answering common questions in SCRUM, such as how many story points should be planned for the upcoming sprint or when the backlog will be completed—on time, preferably.

> **RQ2a** Which features can we select based on ongoing data from the software development process that are most indicative of the progress?

Using our query compiler, we extract features regarding progression of SCRUM sprints of multiple projects from two separate organizations in a standardized effort. The features are scored individually and as subsets through external relevance metrics as well as analogies provided by one of the models. Tuning of hyperparameters is then possible to improve accuracy and confidence of the results. The selected features are related to team size and experience, sprint and backlog planning, code version control and quality metrics. Section 4.4.1 of Chapter 4 lists the selected features.

> **RQ2b** What kinds of learning algorithms can we introduce to this problem, which learn from these features and provide feedback on what kinds of decisions can be taken?

We apply several machine learning models and statistical approaches to our problems of sprint outcomes and backlog sizes. These include state-of-the-art and well-known models, such as deep neural networks (DNN). Some methods have been proven successful in other fields, while others originate from software development already, e.g., analogy-based effort estimation (ABE). The neural networks are applied in this problem to provide a classification as to whether a sprint is at risk of finishing with non-completed stories. ABE goes a step further, synthesizing a number of story points based on similar sprint samples. We also perform estimation of future backlog sizes with two simulations, namely a linear regression algorithm and a Monte Carlo method, using three different scenarios for mutations on the backlog. Together, these models predict the progress in short-term and long-term durations, respectively in sprint commitments and backlog milestone planning. This encompasses the guidelines and provisions of the SCRUM framework, extending it with improved predictability without hindrance to self-managed task commitments.

RQ2c How do we predict the likelihood of timely and properly finishing a currently-running sprint or a longer-running period aimed at resolving a product backlog within a development project, even before it has started?

Using the DNN and ABE models mentioned in our answer to **RQ2b**, we predict the outcomes of sprints, both before a sprint has started as well as during it. The neural network provides not only a classification but also an indication of how reliable the prediction is according to the model itself. Analogies further help with understanding the second model's effort estimation, as they form the basis of the number of story points that are likely to be completed. This allows these models to help development teams through their sprint tasks and meetings—from refinement until retrospective—by providing relevant insights in their process.

For longer periods of time, the backlog size estimations demonstrate potential end dates of completing a certain collective workload by simulating changes to the product backlog across multiple future sprints. Different scenarios provide better detail on the potential changes, with potential for learning from other projects. This has allowed some of our stakeholders, particularly software delivery managers, Product Owners and analysts part of development teams, to determine whether a specific milestone is reachable. Even teams that are just starting with planning stories at the beginning of a development project or are interested in work planning for subsets of their backlog benefit from these algorithms.

RQ2d How do we validate that the predictions and recommendations are within our expectations and based on relevant, explainable factors?

Our research shows potential for the practical applications of the models, but we do not omit well-founded results regarding their accuracy and explainability. We validate the models, using the F1 score and similar metrics appropriate to each model, while comparing the sprint classification and estimation to a baseline guess of taking the previous sprint's outcome as the prediction.

Both the neural network—with an F1 score of 89.98%—and the analogy-based effort estimation, where 88.6% of the estimations of one organization are within a margin of 25%, performed better than the baseline on our temporal data set. We note that some variance is observed when the data set size is adjusted. As for explainability, we consider that feature scoring and built-in tuning of some of the models has helped to select representative subsets of features that are suitable to solving the problem at hand and helpful for stakeholders to track.

For backlog size estimations, the most likely progression is provided by the Monte Carlo method that simulates all potential changes on the backlog. A linear regression model too much variance. A scenario where only the velocity is accounted for could not properly simulate sudden changes to the backlog's scope. We validated these results by taking a fraction of previous progression of the product backlog, measured at the end of each sprint, as our data set and comparing the model's estimations to the actual progression. We measured the mean error as well as the largest deviations. We further analyzed the behavior of our backlog estimation models per project, taking into account differences in backlog sizes in each of them. Finally, we validated the distributions generated by the Monte Carlo simulations, to determine if the scenarios remain normalized despite the introduction of additional factors.

Information visualization (Chapter 5)

RQ3 How can we effectively introduce visual representations of results and recommendations from our analysis of data collected from the SCRUM development process to the involved parties?

The third and final research question of the Grip on Software project concerns the presentation of data to relevant stakeholders. This shifts our focus from purely analysis, where solely achieving an acceptable accuracy is already a plentiful outcome. For this research question, we consider how the end user benefits from the collection of previously unlinked entities in novel ways. This includes the results from our pattern recognition methods, but through an integrated visualization hub, we intuitively zoom in to various aspects of the breadth of information to showcase the possibilities that the GROS pipeline provides.

RQ3a Which concepts and goals are relevant when designing information visualizations for patterns analyzed from a software development process?

As part of our introduction of a comprehensive visual summary of the results regarding the outcomes of SCRUM sprints, among others, we outline the necessary objectives, concepts and goals regarding visualization for Agile software development, information visualization in general and the structure of our specific dashboard implementation, respectively. This includes stating the purposes regarding display of temporal, dynamic data, making data easily comparable, gaining insights and sparking new ideas.

We consider existing flows of transforming data into visual forms that are rendered in an interactive view. We map this method upon the components of our pipeline as well as the cognitive process of gaining knowledge from descriptive data. We find that an integrated approach—both in our technical component as well as regarding the process of bringing the visualizations to the involved persons—and a clearly-outlined purpose help make the data become information that is useful for stakeholders to learn from. This helps boost delivery of the product and promotes new ways of thinking in the development process.

RQ3b How do we integrate results from predictive models within existing development practices?

Our visualization hub consists of a dashboard which provides access to various information visualizations, with a similar technical and interactive implementation. The visualizations are grouped by focus area, with two main categories: analytical decision support and software development ecosystem management. The former group includes visualizations for reporting on progress of teams across earlier sprints in various formats. One of the visualizations is arranged as a distinct location for the results of our sprint predictions. A clear layout makes the main results prominent, while allowing further scrutiny through detailed tables of features, model configurations and—depending on the model—additional metrics or a list of similar sprints.

Further disclosure is achieved through an API which allows external systems to integrate with the results. One implementation to display the prediction result in a quality report was made for this purpose, helping developers see whether they should consider planning a different number of story points for the next sprint. The results from our analysis of sprint predictions across different training set sizes as well as the backlog size estimations are available in the sprint report visualization, which lets users freely make projections for planning purposes. Additionally, the Jira plugins build upon the analysis and results from the backlog size prediction, by allowing users to view a breakdown of a product backlog across time, including the future sprints simulated by the Monte Carlo method.

RQ3c Which effects do the introduction of results from predictive models and other intermediate analyses have on the development process, validated across multiple ongoing projects?

We present the information visualizations to potential users during surveys and interviews. These stakeholders include developers, quality engineers and product owners. During our study, we assess the usability of the visualization through various metrics, while asking the users how they would consider using the information contained in the visualization during their daily work. Whereas initial surveys for a limited number of visualizations showed mediocre potential usage, we later performed further interviews which indicated a significant adoption, especially regarding the use of the sprint report. Various teams use this visualization to learn from earlier sprints by selecting primary indicators that fit their team the most. This feedback was helpful for us, by confirming whether selected feature sets that we applied during the predictions are indeed relevant.

The output of the prediction models has been further integrated with the visualizations through as well as existing quality reports. In particular, earlier evaluation showed that there was a need to easily view attributes of previous, current and future sprints, thus these attributes plus the backlog size estimations were made available in the sprint report. This visualization is used by product owners and software delivery managers to gain an overview and to plan full backlogs, with planning based on milestones supported tentatively. Our predictions are considered a beneficial addition to the existing workflow. The use of predictions as a meta-metric gains adoption with an explainable backtrace to the contributing factors. By creating plugins for Jira, we brought such information back to more teams.

RQ3d What is the overall assessment of the proposed information visualizations, considering automated measurements for usability and the adoption according to interviews and surveys?

We combine integration test results with user adoption surveys to produce an assessment of the information visualizations. In general, the visualization hub is found to be usable. This is boosted by considerations on, e.g., colors, glyphs and consistent controls, but also wider topics like traceability of data, mobile-friendly design and accessibility. This allows adoption by different roles in the software development organizations, regardless of technological background or impairments such as color blindness.

By focusing our visualizations on topics at an organizational and project-specific level, both in the analytical and ecosystem management domains, they are well-received and align with

regulations regarding privacy. Finally, we remark that there is still more to explore for information visualization in software development, particularly when it comes to planning and product development, to further enhance this process.

6.3 Future work

A primary objective of our research is exploring the feasibility and performance of several pattern recognition methods in predicting potential effort in SCRUM sprints and entire project backlogs. This research has shown that we are able to acquire a representative data set to obtain accurate, explainable predictions and deliver relevant information back to the stakeholders by creating interactive, integrated visualizations. Therefore, we deduce from our answers of the research questions that we meet many of our objectives in several sub-topics, including technical implementations, data modeling and extraction, machine learning and eventual presentation.

However, there is always more research to be done, especially in the field of predictive software development. While we have provided answers to our research questions, there still remain unknowns in this domain regarding other software development methods or even other processes part of the SCRUM framework.

In other chapters, we briefly discussed possible improvements and methods to be considered. We are hopeful that our work functions as a viable means of continuing research. For Grip on Software in particular, we hypothesize that further integration of data-driven predictions in software development helps improve the process as a whole, by letting involved people find more emerging patterns. This process allow learning from past situations, current recommendations and future projections in the same place. In the end, this research makes repetitive events—which would otherwise feel ceremonial to developers—more enriched through the addition of easy-to-use visual results.

Through generalization, some of our contributions become beneficial when applied in other focus areas. Our approach toward designing and implementing modular pipeline components is translatable to other exploratory data-driven analysis. This technical foundation is available for others to build upon, since similar challenges and objectives regarding collecting, structuring, selecting, analyzing and representing large amounts of data exist elsewhere too.

Our vision of prospective research based on this thesis is therefore twofold. We consider the particular methods and models that build upon our data set in Section 6.3.1, with some limited speculation on what we imagine the end result to look like. To address possibilities beyond our current scope, we consider applicability to other software development methods and research studies in general in Section 6.3.2.

6.3.1 Further research

In several chapters, we discuss our intermediate conclusions for the topics at hand there, with suggestions regarding possible avenues of further research as well. We expand upon the earlier contemplations with several promising applications of our data in improving the predictability of SCRUM software development in more ways. We considered some of these sub-problems to become a part of our core research—with some initial work available for them as well—but some topics simply strayed from the “narrative arc” that we sought to maintain in our work.

In Chapter 4, we perform classification and estimation of SCRUM sprint outcomes and full project backlogs. One possibility is to extend this approach toward other time ranges and subsets of the backlogs. A proof of concept, where we determine when work for a specific milestone would be completed, is also shown in Section 4.7.2. Ideally, we would be able to consider how much time an early backlog will roughly take to complete, even when the development team has not awarded story points to most of the work items.

We also mentioned other models for improved accuracy and explainability. One option would be the application of Long Short-Term Memory (LSTM), a model which includes contextual information about nearby samples during training. This has shown applications in linguistic processing and other consecutive data. Because our data is temporal in nature, models that understand the relations with earlier events—e.g., when stories are moved from one sprint to the current or future sprint—are well-suited in providing more thorough labels, clusters or estimations for the samples in the GROS data set.

Based on the model of our database, we are able to extract features for other entities than just sprints or projects. Through our curation with the query template compiler, we extend our understanding to the level of teams, which benefits organizations where teams work on multiple projects or share responsibility on a larger project with multiple teams by developing individual components. Some of such multi-project teams still need some additional curation to reliably combine data from simultaneous sprints, including how story points scale the entire backlog. Project-level and team-level predictions then become more comparable.

More in-depth predictive analysis is obtainable by defining features for each story, code change or quality check, leading to an even larger data set that provides novel classification opportunities. One possible application is to provide an *early warning* to developers when something seems wrong with a specific metric, such as a missing field in a work item or an unexpected decrease in quality.

Due to the continuous nature of code quality measurements and development platform stability, we also consider *anomaly detection* to be a beneficial technique, where rare situations are found and presented for further analysis on why they take place. This leads to potential new features for use in other machine learning problems, allow early warning on its own or be used as a hint for tracing back why a feature is different than normal in the first place.

Other ways to augment our data set is to include external data, such as weather or traffic information, in order to find possible reasons for differences in development throughput day by day. Although we focused on objective data available in the software development ecosystem, we should also keep in mind the human aspect, even in our data set. For example, a developer's mood likely affects the quality and quantity of code written by that developer. It is conceivable that *sentiment analysis* on comments left in issue trackers and version control systems provides an indication of how well a sprint is going. However, more intrusive approaches—such as recording and transcribing all of the team's meetings—might encroach too much of the sensitive nature of such group events, even if the output is anonymized before combining with our current, privacy-aware pipeline.

Instead, we should focus on implementing more means of explainability in our predictive models, along with other improvements in terms of feature extraction and types of model outputs, including inherent accuracy metrics. In our case, this approach advances integration of results in the SCRUM workflow, with less potential for misunderstanding. Better tracing of where a neural network's activation comes from, regardless of the network size, leads to improvements of the perceived reliability of the algorithm, which is important to our stakeholders.

Other implementation decisions would be on-line re-runs of backlog estimations, such that users are able to try different parameters and subsets for planning. This functionality is then available through a visualization shown in Chapter 5, in particular the sprint report or through a plugin for Jira. We wish to further explore how information visualization is applicable in the Agile software development domain, with more integration of the information visualizations. This leads to a single entry point where it is possible to dive deep into specific details, while retaining a simple front-end for obtaining overviews regarding progress. Each *zoom* level then has its own presentation format applicable to the information visible there, such as tables, charts and network graph layouts.

6.3.2 Generalizability

The focus of the Grip on Software research project has been on software development processes that use the SCRUM framework. However, it is possible to expand the scope to include other areas of interest. It would be interesting to compare other software development methods to ascertain whether similar patterns are found there. Due to the repetitive nature of SCRUM, we are able to create a consistent data set of sprints as our feature-rich samples. Mapping such specific events to other frameworks is a challenge with a potential high reward, as a more broad model leads to wider applicability of the results of this extended research.

Certainly, the most sensible development methods to consider would be other Agile processes, such as extreme programming, Kanban or Lean, which share similarities with SCRUM. It is feasible that our approach also works when corresponding entities are not easily mapped. We would then focus on other levels of detail in work volume, such as milestones or individual issue items. At the organizations in our study, we also collected information on support teams. Some teams do not use SCRUM—or only for a portion of the workload—but we still provided some reports to them regarding work done in specific time frames.

Given that the two organizations are both Dutch IT implementation agencies for governments, one major opportunity for advancement is to include more diverse development companies in our data set. Each organization has its own working environment, including differences in the application of SCRUM among their development teams. Especially multinational businesses with colocated teams have widely contrasting concerns and challenges compared to a tightly-connected, local organization. The effect of developing products for-profit—instead of based on government budget—may also play a role in team workload. Overall, more breadth of data from various organizational instances leads to better models, but also more research hypotheses that we are able to test.

The technical infrastructure should be able to handle more diverse data sources. We have shown that our database model is able to handle entities that originate from different systems and provide them in a consistent format through our query template compiler. Development ecosystems with other platforms, trackers and collaboration software are easily added, as is the case for GitHub and GitLab, for example.

In general, our combined approach toward data acquisition, storage, modeling, analysis and visualization are available as building blocks for more studies in this field and even for other data-oriented research. The modular nature of the components allows adjusting the purpose of the pipeline by replacing or extending parts of it. While there is a focus on software development data, in theory the use of schema-based data exchange allows extensive abstraction of the type of incoming data.

The openness of data formats also allows better dissemination of the results of our research. Through a consistent API specification, we provide both the data set used specifically for the model as well as detailed information on model configuration and validation outcomes. This encourages not only reuse of the input data, model and labels for further study, but also sets an example on how transparency regarding analysis parameters helps to create feedback loops. Integration of such details in an intelligible manner in information visualization helps to foster interaction and discussion that exceed initial expectations. A human-centered approach in visualization leads to new possibilities, in the software ecosystem and beyond.

Glossary

Acronyms

ABE anology-based effort estimation; see pages 8, 9, 38, 76, 80, 81, 84, 111, 112, 149, 155 and 156

AI artificial intelligence; see pages 7 and 89

API application programming interface; see pages 14, 30, 32, 36, 37, 75, 80, 113, 126, 153, 158 and 162

AUC Area Under Curve; see page 111

CI continuous integration; see page 37

CPU central processing unit; see pages 39, 64–66 and 131

CSS Cascading Style Sheets; see page 100

CSV comma-separated values; see pages 61 and 106

DNN deep neural network; see pages 8, 9, 81, 111, 155 and 156

DoD Definition of Done; see pages 6, 7, 73 and 125

ER entity–relationship diagram; see pages 47–50

FTE full-time equivalent; see pages 49 and 122

GPU graphics processing unit; see pages 37, 38 and 149

HCI human-computer interaction; see page 95

HTML HyperText Markup Language; see pages 100, 106, 128 and 138

HTTPS HyperText Transfer Protocol Secure; see pages 33, 61 and 138

InfoVis information visualization; see page 97

JDBC Java Database Connectivity; see page 59

JSON JavaScript Object Notation; see pages 30, 37, 38, 60, 100, 103, 106 and 153

LDAP Lightweight Directory Access Protocol; see pages 36, 47, 48, 58 and 59

Lin linear regression algorithm; see pages 8, 9, 80, 82 and 86

LSTM Long Short-Term Memory; see pages 91 and 160

MC Monte Carlo simulation; see pages 8, 9, 76, 80, 82, 86, 103 and 135

ML machine learning; see pages 7, 8, 12, 71, 76 and 89

NN neural network; see pages 8, 9, 80, 81 and 149

PB product backlog; see pages 6, 7, 10, 72 and 73

PDF Portable Document Format; see pages 106 and 110

PG product goal; see pages 6 and 7

PO Product Owner; see pages 5–7, 10, 52, 72–74

PR pattern recognition; see pages 8 and 71

RAM random access memory; see pages 39 and 64

RDBMS relational database management system; see pages 43–46, 63, 66 and 154

SB sprint backlog; see pages 6, 7, 10, 72 and 73

SDM software delivery manager; see pages 10 and 79

SG sprint goal; see pages 6, 7 and 52

SM Scrum Master; see pages 5, 10 and 73

SP story point; see pages 6, 10, 11, 52, 74, 83 and 153

SQL Structured Query Language; see pages 46, 59, 61, 63, 66, 77 and 149

SSH Secure Shell; see pages 31 and 153

SUS System Usability Scale; see pages 141 and 143

UDF user-defined function; see pages 44–46

UI user interface; see pages 98, 99, 101 and 139

UML Unified Modeling Language; see page 50

URL Uniform Resource Locator; see pages 32, 38, 52, 59, 106, 112 and 129

VCS version control system; see pages 58 and 59

VM virtual machine; see pages 31, 39 and 60

Software development terminology

architecture high-level structural overview of a software system, as a design specification; see page 4

artifact a document or different byproduct that specify specific requirements, parts of the design or architecture, at greater detail; see page 5

burndown chart time-based diagram that displays lines and points that refer to certain events taking place in a sprint regarding changes to the number of story points left to work on from each point onward; see pages 6, 114 and 134

code textual files containing lines with instructions written in a programming language which perform actions that are part of a software system; see pages 4, 6, 10, 34, 37, 71, 73, 91 and 125

coverage percentage of statements or lines of code that is being executed during tests of a software product, as a measurement of how likely it is that problems and edge cases are detected; see pages 4, 6, 20, 35, 37, 73 and 91

Daily Scrum short meeting in SCRUM held every working day where the development team discusses what that have done during the sprint so far, what they are working on and possible impediments that hinder their tasks; see pages 6, 10, 73, 74 and 121

deployment installation or publication of a software product so that it is available to users; see pages 4, 10, 11 and 18

ecosystem environment in which code may be written (software development ecosystem) or a deployed product may be placed, where the developed software interacts with other systems and is dependent on a platform providing support for its functionality; see page 4

epic task that explains relationships between smaller tasks, such as user stories; see pages 6, 79 and 80

feature aspect of a software product that allows the system to perform something by providing certain functionality; see pages 4, 7 and 73, not to be confused with feature (Machine learning terminology)

guild meeting of a group of people across an entire organization with an interest in a particular topic, but available for everyone, with discussions ranging from Agile development methods to testing code and improving quality; see pages 10, 11 and 121

impediment any cause of delay and hindrance in the software development progress, which needs to be resolved before developers can continue with a certain task; see pages 5, 10 and 73

increment result of a software development cycle such as a SCRUM sprint that adheres to pre-set goals, consisting of changes from all the resolved items during that period, and may become a deployment (Potentially Shippable Product Increment) or released version, even when early in development (minimum viable product); see pages 6, 7, 11, 14, 72 and 73

maintenance regular adjustment of a software product after deployment in order to keep the product functioning in the environment in which the software is placed; see page 4

milestone moment in a software development plan that indicates an important step in the progress, usually when a new version is released or a deployment is scheduled; see pages 4 and 10

product the result of software development, fulfilling a need of users; see pages 4, 10 and 14

readiness quality of a story or other task in that it has been prepared enough during refinement meetings to be detailed enough to work on, with the team agreeing that is it not too complicated (ready for selection); see pages 6, 10, 72 and 153

refinement meeting in SCRUM to improve details of planned work for an upcoming sprint development cycle; see pages 6, 10, 71, 73 and 125

requirement specification of what a system, software and entire product should do (functional requirement) or should adhere to with regards to its environment (non-functional requirement); see pages 4, 16 and 19

retrospective meeting in SCRUM where the development team discuss internally how the previous sprint progressed and improve focus on important factors; see pages 6, 10, 14, 71–73

review meeting in SCRUM where the development team presents and discusses the results of the previous sprint with representatives of the end user, usually including a display of new functionality (demo); see pages 6, 10, 14, 71–73

sprint time span in a SCRUM development process, with specific meetings and goals, which repeats itself to work on more tasks; see pages 5, 6, 10, 72 and 73

sprint planning meeting in SCRUM to select tasks to be worked on during the next sprint development cycle; see pages 6, 10, 71 and 73

stakeholder people and parties with the most interest in a software development process, including members of the development teams, managerial roles or others in the organization, but also the end users and the client, who fulfills the role of eventual owner of a product; see pages 14, 95, 100, 102, 121 and 142

story request for a task related to developing code for a new software feature in a product and other relevant work, described in a simple format, usually in a single sentence describing a desire (user story); see pages 6, 7, 10, 72, 73 and 153

technical debt projected amount of effort, time or expenses in order to resolve a current, subpar situation so that a better solution is implemented in a software product which would require less maintenance in the future, whereas if the debt is not resolved, it will become harder to address later on, often used in the context of code style; see pages 56, 75, 79, 91 and 104

test method of comparing a software product to the specified requirements at various levels of inspection, such as small components (unit test) or interaction of systems in the software ecosystem (integration test); see pages 4, 10, 11, 20, 34, 37, 71, 73, 91, 125, 138 and 141, not to be confused with test (Machine learning terminology)

velocity metric used as a guideline for the number of story points to plan for a sprint, where the sum of the story points of all stories that were done during the past three sprints is divided by 3 (three-sprint velocity); see pages 74, 77, 78, 80, 82 and 83

Machine learning terminology

classification problem where the goal is to find a label for an unlabeled sample selected from a limited set of classes using a machine learning model (classification algorithm); see pages 7, 79–81, 84 and 91

clustering problem where the goal is to group similar samples from a data set together using a machine learning model; see pages 7, 81 and 109

data set collection of (usually different) records that describe objects, situations or events that are typically from a similar domain, with various properties mmaking up each sample record; see pages 7, 71, 79 and 90

ensemble model method to compose various machine learning algorithms together and to use their output, e.g., using a majority vote to choose the result, for solving machine learning problems; see pages 8 and 83

estimation problem where the goal is to find a label for an unlabeled sample that seems to fit the features using a machine learning or statistical model; see pages 7, 80–84 and 91

explainability quality of a machine learning algorithm, either inherent to the model used or achieved through external methods, that allows tracing back how a label or estimation was generated, for example which inputs were most relevant or which samples are most similar; see pages 8, 81, 84, 90, 152, 156 and 160

feature measurable observation about a specific sample in a data set; see pages 7, 76, 81 and 83, not to be confused with feature (Software development terminology)

feature selection process where a subset of the features from a data set are chosen based on scoring or other criteria, leading to a more refined working set; see pages 7, 72, 76, 78 and 84

label description of an object in a numerical or categorical manner, which is the goal of some machine learning problems in order to understand the data better (labeling), and when already available in the data set, is the expected outcome of the model given the sample input (target label); see page 7

model algorithm used in machine learning in order to solve a problem, such as providing a label to an object; see pages 7, 71, 76, 78, 80, 81, 83, 84 and 90

regression analysis method used to perform estimation of relationships between labels and the associated features of samples in a data set, using a function that closely fits most of the observed data points; see pages 8, 76, 103 and 109

sample entries in a data set that describe a particular object, situation or event, which may be used separately or in bulk as input for a machine learning model by selecting subsets of records (sampling); see page 7

supervised learning algorithm that is able to use labeled samples and extract statistical relations in order to learn patterns and generate numerical labels; see pages 7 and 76

test process where a portion of labeled samples from the data set (test set) is used to obtain accuracy metrics of the trained model, with a similar distribution; see pages 7, 78–81 and 84, not to be confused with test (Software development terminology)

training process where a portion of labeled samples from the data set (training set) is used to learn a model what patterns and relations between features exist in order to generate better labels in the future; see pages 7, 76, 79–81 and 84

trend outcome of a regression analysis, most typically a linear regression where the overall direction of temporal data is shown as a line, allowing for an estimation of future data points; see pages 8, 89, 103 and 109

unsupervised learning algorithm that uses unlabeled samples to extract statistical relations in order to learn patterns and similarities; see pages 7 and 109

validation process where a portion of labeled samples from the data set (validation set) is used to check if the model is well-tuned and not biased toward the samples from the training set; see pages 7, 79–81, 84 and 103

Bibliography

First referenced in Chapter 1

[1] Nayan B. Ruparelia. “Software development lifecycle models”. *ACM SIGSOFT Software Engineering Notes*, vol. 35, no. 3, 2010, pp. 8–13. DOI: 10.1145/1764810.1764814.

[2] Todd Sedano, Paul Ralph and Cécile Péraire. “Software development waste”. In: *Proceedings of the 2017 IEEE/ACM 39th International Conference on Software Engineering* (ICSE 2017). 2017, pp. 130–140. DOI: 10.1109/ICSE.2017.20.

[3] Agile Alliance. *Manifesto for Agile Software Development*. 2001. URL: https://agilemanifesto.org/ (visited on May 7, 2019).

[4] James A. Highsmith. *Agile Software Development Ecosystems*. Addison-Wesley, 2002.

[5] Alistair Cockburn. *Agile Software Development: The Cooperative Game*. 2nd ed. Addison-Wesley, 2007.

[6] Ken Schwaber and Jeff Sutherland. *The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game*. 2020. URL: https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf (visited on Mar. 1, 2022).

[7] Scrum.org. *The Scrum Framework Poster*. 2020. URL: https://www.scrum.org/resources/scrum-framework-poster (visited on May 29, 2021).

[8] Ken Schwaber and Jeff Sutherland. *Software in 30 Days: How Agile Managers Beat the Odds, Delight Their Customers, And Leave Competitors In the Dust*. John Wiley & Sons, 2012. DOI: 10.1002/9781119203278.

[9] Mike Cohn. *Agile Estimating and Planning*. Prentice Hall, 2005.

[10] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 4th ed. Pearson, 2021.

[11] Trevor Hastie, Robert Tibshirani and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. 2nd ed. Springer Series in Statistics. Springer, 2009. DOI: 10.1007/978-0-387-84858-7.

[12] Laurens J. P. van der Maaten, Eric O. Postma and H. Jaap van den Herik. *Dimensionality reduction: A comparative review*. Technical Report. TR 2009-005. TiCC, 2009, pp. 1–35.

[13] Kenji Kira and Larry A. Rendell. "The feature selection problem: Traditional methods and a new algorithm". In: *Proceedings of the Tenth National Conference on Artificial Intelligence* (AAAI'92). AAAI Press, 1992, pp. 129–134.

[14] Ian Goodfellow, Yoshua Bengio and Aaron Courville. *Deep Learning*. Adaptive Computation and Machine Learning series. MIT Press, 2016.

[15] David E. Rumelhart, Geoffrey E. Hinton and Ronald J. Williams. "Learning representations by back-propagating errors". *Nature*, vol. 323, no. 6088, 1986, pp. 533–536. DOI: 10.1038/323533a0.

[16] Yoshua Bengio. "Learning deep architectures for AI". *Foundations and Trends® in Machine Learning*, vol. 2, no. 1, 2009, pp. 1–127. DOI: 10.1561/2200000006.

[17] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer, 2006.

[18] Ane Blázquez-García, Angel Conde, Usue Mori and Jose A. Lozano. "A review on outlier/anomaly detection in time series data". *ACM Computing Surveys*, vol. 54, no. 3, article 56, 2021. DOI: 10.1145/3444690.

[19] David A. Freedman. *Statistical Models: Theory and Practice*. 2nd ed. Cambridge University Press, 2009. DOI: 10.1017/CBO9780511815867.

[20] Christopher Z. Mooney. *Monte Carlo Simulation*. Quantitative Applications in the Social Sciences 116. SAGE Publications, 1997. DOI: 10.4135/9781412985116.

[21] Omer Sagi and Lior Rokach. "Ensemble learning: A survey". *WIREs Data Mining and Knowledge Discovery*, vol. 8, no. 4, article 1249, 2018. DOI: 10.1002/widm.1249.

[22] Martin Shepperd, Chris Schofield and Barbara Kitchenham. "Effort estimation using analogy". In: *Proceedings of the 18th International Conference on Software Engineering* (ICSE '96). IEEE Computer Society, 1996, pp. 170–178. DOI: 10.1109/ICSE.1996.493413.

[23] Steven Finlay. *Predictive Analytics, Data Mining and Big Data: Myths, Misconceptions and Methods*. Business in the Digital Economy. Palgrave Macmillan, 2014. DOI: 10.1057/9781137379283.

[24] Bertrand Meyer. *Agile!: The Good, the Hype and the Ugly*. Springer, 2014. DOI: 10.1007/978-3-319-05155-0.

First referenced in Chapter 2

[25] Maarten van Steen and Andrew S. Tanenbaum. *Distributed Systems*. 3rd ed. distributed-systems.net, 2017.

[26] Gianpaolo Cugola and Alessandro Margara. "Processing flows of information: From data stream to complex event processing". *ACM Computing Surveys*, vol. 44, no. 3, article 15, 2012. DOI: 10.1145/2187671.2187677.

[27] Philip Harrison Enslow. "What is a "distributed" data processing system?" *Computer*, vol. 11, no. 1, 1978, pp. 13–21. DOI: 10.1109/C-M.1978.217901.

[28] Franco Zambonelli, Nicholas R. Jennings and Michael J. Wooldridge. "Organisational abstractions for the analysis and design of multi-agent systems". In: *Agent-Oriented Software Engineering* (AOSE 2000). Lecture Notes in Computer Science, vol. 1957. Springer, 2001, pp. 235–251. DOI: 10.1007/3-540-44564-1_16.

[29] Wei Ren and Yongcan Cao. *Distributed Coordination of Multi-agent Networks*. Communications and Control Engineering. Springer, 2011. DOI: 10.1007/978-0-85729-169-1.

[30] Wolfgang Reisig. *Elements of Distributed Algorithms: Modeling and Analysis with Petri Nets*. Springer, 1998. DOI: 10.1007/978-3-662-03687-7.

[31] Wil M. P. van der Aalst. "The application of Petri nets to workflow management". *Journal of Circuits, Systems and Computers*, vol. 8, no. 1, 1998, pp. 21–66. DOI: 10.1142/S0218126698000043.

[32] MengChu Zhou and Naiqi Wu. *System Modeling and Control with Resource-Oriented Petri Nets*. CRC Press, 2010. DOI: 10.1201/9781439808856.

[33] Bruno Lopes, Mario Benevides and Edward Hermann Haeusler. "Reasoning about multi-agent systems using stochastic Petri nets". In: *Trends in Practical Applications of Agents, Multi-Agent Systems and Sustainability*. Springer, 2015, pp. 75–86. DOI: 10.1007/978-3-319-19629-9_9.

[34] Gregor Hohpe, Bobby Woolf, Kyle Brown, Conrad F. D'Cruz, Martin Fowler, Sean Neville, Michael J. Rettig and Jonathan Simon. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. A Martin Fowler signature book. Addison-Wesley, 2004.

[35] Paolo Ceravolo et al. “Big data semantics”. *Journal on Data Semantics*, vol. 7, no. 2, 2018, pp. 65–85. DOI: 10.1007/s13740-018-0086-2.

[36] Aiswarya Raj Munappy, Jan Bosch and Helena Homström Olsson. “Data pipeline management in practice: Challenges and opportunities”. In: *Product-Focused Software Process Improvement* (PROFES 2020). Lecture Notes on Computer Science (Programming and Software Engineering), vol. 12562. Springer, 2020, pp. 168–184. DOI: 10.1007/978-3-030-64148-1_11.

[37] Dirk Merkel. “Docker: Lightweight Linux containers for consistent development and deployment”. *Linux Journal*, vol. 2014, no. 239, article 2, 2014.

[38] Stratos Idreos, Fabian Groffen, Niels Nes, Stefan Manegold, K. Sjoerd Mullender and Martin L. Kersten. “MonetDB: Two decades of research in column-oriented database architectures”. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, vol. 35, no. 1, 2012, pp. 40–45.

[39] Gerald Carter. *LDAP System Administration: Putting Directories to Work*. O’Reilly Media, 2003.

First referenced in Chapter 3

[40] Mark Raasveldt, Pedro Holanda, Hannes Mühleisen and Stefan Manegold. “Deep integration of machine learning into column stores”. In: *Proceedings of the 21st International Conference on Extending Database Technology* (EDBT). 2018, pp. 473–476. DOI: 10.5441/002/edbt.2018.50.

[41] Hannes Mühleisen, Alexander Bertram and Maarten-Jan Kallen. “Database-inspired optimizations for statistical analysis”. *Journal of Statistical Software*, vol. 87, no. 4, 2018, pp. 1–20. DOI: 10.18637/jss.v087.i04.

[42] Georgios Gousios, Dominik Safaric and Joost Visser. “Streaming software analytics”. In: *Proceedings of the 2016 IEEE/ACM 2nd International Workshop on Big Data Software Engineering* (BIGDSE ’16). 2016, pp. 8–11. DOI: 10.1145/2896825.2896832.

[43] Mark Raasveldt. “Integrating analytics with relational databases”. In: *Proceedings of the VLDB 2018 PhD Workshop co-located with the 44th International Conference on Very Large Databases* (VLDB 2018). 2018.

[44] Joseph Vinish D’Silva, Florestan De Moor and Bettina Kemme. “AIDA: Abstraction for advanced in-database analytics”. *Proceedings of the VLDB Endowment*, vol. 11, no. 11, 2018, pp. 1400–1413. DOI: 10.14778/3236187.3236194.

[45] Ying Zhang, Richard Koopmanschap and Martin L. Kersten. "Love at first sight: MonetDB/TensorFlow". In: *IEEE 34th International Conference on Data Engineering* (ICDE). 2018, pp. 1672–1672. DOI: 10.1109/ICDE.2018.00208.

[46] Jonathan Lajus and Hannes Mühleisen. "Efficient data management and statistics with zero-copy integration". In: *Proceedings of the 26th International Conference on Scientific and Statistical Database Management* (SSDBM '14). ACM, 2014, article 12. DOI: 10.1145/2618243.2618265.

[47] Paul Blockhaus, David Broneske, Martin Schäler, Veit Köppen and Gunter Saake. "Combining two worlds: MonetDB with multi-dimensional index structure support to efficiently query scientific data". In: *32nd International Conference on Scientific and Statistical Database Management* (SSDBM 2020). ACM, 2020, article 29. DOI: 10.1145/3400903.3401691.

[48] Mark Raasveldt, Pedro Holanda, Tim Gubner and Hannes Mühleisen. "Fair benchmarking considered difficult: Common pitfalls in database performance testing". In: *Proceedings of the Workshop on Testing Database Systems* (DBTest'18). ACM, 2018, article 2. DOI: 10.1145/3209950.3209955.

[49] Irene Martorelli et al. "Fungal metabarcoding data integration framework for the MycoDiversity DataBase (MDDB)". *Journal of Integrative Bioinformatics*, vol. 17, no. 1, article 20190046, 2020. DOI: 10.1515/jib-2019-0046.

[50] Sirine Zaouali and Sonia Ayachi Ghannouchi. "Integrating quality assessment through metrics into Scrum software development". In: *Proceedings of the 20th International Conference on New Trends in Intelligent Software Methodologies, Tools and Techniques* (SoMeT_21). Vol. 337. Frontiers in Artificial Intelligence and Applications. IOS Press. 2021, pp. 211–223. DOI: 10.3233/FAIA210021.

[51] Florencia Vega, Guillermo Rodríguez, Fabio Rocha and Rodrigo Pereira dos Santos. "Scrum Watch: A tool for monitoring the performance of Scrum-based work teams". *Journal of Universal Computer Science*, vol. 28, no. 1, 2022, pp. 98–117. DOI: 10.3897/jucs.67593.

[52] Paulo Sérgio Santos Júnior, Monalessa Perini Barcellos, Ricardo de Almeida Falbo and João Paulo A. Almeida. "From a Scrum reference ontology to the integration of applications for data-driven software development". *Information and Software Technology*, vol. 136, article 106570, 2021. DOI: 10.1016/j.infsof.2021.106570.

[53] Mark Raasveldt and Hannes Mühleisen. "Vectorized UDFs in column-stores". In: *Proceedings of the 28th International Conference on Scientific and Statistical Database Management* (SSDBM '16). ACM, 2016, article 16. DOI: 10.1145/2949689.2949703.

[54] Mark Raasveldt. "MonetDBLite: An embedded analytical database". In: *Proceedings of the 2018 International Conference on Management of Data* (SIGMOD '18). ACM, 2018, pp. 1837–1838. DOI: 10.1145/3183713.3183722.

[55] Peter A. Boncz, Martin L. Kersten and Stefan Manegold. "Breaking the memory wall in MonetDB". *Communications of the ACM*, vol. 51, no. 12, 2008, pp. 77–85. DOI: 10.1145/1409360.1409380.

[56] David Gembalczyk, Felix Martin Schuhknecht and Jens Dittrich. "An experimental analysis of different key-value stores and relational databases". In: *Datenbanksysteme für Business, Technologie und Web* (BTW2017). Gesellschaft für Informatik, 2017, pp. 351–360.

[57] Mark Raasveldt and Hannes Mühleisen. "Don't hold my data hostage: A case for client protocol redesign". *Proceedings of the VLDB Endowment*, vol. 10, no. 10, 2017, pp. 1022–1033. DOI: 10.14778/3115404.3115408.

[58] Fred Long, Dhruv Mohindra, Robert C. Seacord, Dean F. Sutherland and David Svoboda. *Java Coding Guidelines: 75 Recommendations for Reliable and Secure Programs*. SEI Series in Software Engineering. Pearson Education, 2013.

First referenced in Chapter 4

[59] Leon Helwerda, Frank Niessink and Fons J. Verbeek. "Conceptual process models and quantitative analysis of classification problems in Scrum software development practices". In: *Proceedings of the 9th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management* (IC3K 2017 - KDIR). SCITEPRESS, 2017, pp. 357–366. DOI: 10.5220/0006602803570366.

[60] Viljan Mahnič and Tomaž Hovelja. "On using planning poker for estimating user stories". *Journal of Systems and Software*, vol. 85, no. 9, 2012, pp. 2086–2095. DOI: 10.1016/j.jss.2012.04.005.

[61] Sondra Ashmore and Kristin Runyan. *Introduction to Agile Methods*. Addison-Wesley Professional, 2014.

[62] Sergio Di Martino, Filomena Ferrucci, Carmine Gravino and Federica Sarro. "Assessing the effectiveness of approximate functional sizing approaches for effort estimation". *Information and Software Technology*, vol. 123, article 106308, 2020. DOI: 10.1016/j.infsof.2020.106308.

[63] Zainab Masood, Rashina Hoda and Kelly Blincoe. "Real world scrum: a grounded theory of variations in practice". *IEEE Transactions on Software Engineering*, vol. 48, no. 5, 2022, pp. 1579–1591. DOI: 10.1109/TSE.2020.3025317.

[64] Jacky Wai Keung, Barbara A. Kitchenham and David Ross Jeffery. "Analogy-X: Providing statistical inference to analogy-based software cost estimation". *IEEE Transactions on Software Engineering*, vol. 34, no. 4, 2008, pp. 471–484. DOI: 10.1109/TSE.2008.34.

[65] Mohammad Azzeh, Daniel Neagu and Peter I. Cowling. "Analogy-based software effort estimation using fuzzy numbers". *Journal of Systems and Software*, vol. 84, no. 2, 2011, pp. 270–284. DOI: 10.1016/j.jss.2010.09.028.

[66] Kenichi Ono, Masateru Tsunoda, Akito Monden and Kenichi Matsumoto. "Influence of outliers on analogy based software development effort estimation". In: *Proceedings of the IEEE/ACIS 15th International Conference on Computer and Information Science* (ICIS). 2016, pp. 1–6. DOI: 10.1109/ICIS.2016.7550865.

[67] Eliane Maria De Bortoli Fávero, Roberto Pereira, Andrey Ricardo Pimentel and Dalcimar Casanova. "Analogy-based effort estimation: A systematic mapping of literature". *INFOCOMP Journal of Computer Science*, vol. 17, no. 2, 2018, pp. 07–22.

[68] Marta Fernández-Diego, Erwin R. Méndez, Fernando González-Ladrón-de Guevara, Silvia Mara Abrahão and Emilio Insfrán. "An update on effort estimation in Agile software development: A systematic literature review". *IEEE Access*, vol. 8, 2020, pp. 166768–166800. DOI: 10.1109/ACCESS.2020.3021664.

[69] Heejun Park and Seung Baek. "An empirical validation of a neural network model for software effort estimation". *Expert Systems with Applications*, vol. 35, no. 3, 2008, pp. 929–937. DOI: 10.1016/j.eswa.2007.08.001.

[70] Fatima Boujida, Fatima Amazal and Ali Idri. "Neural networks based software development effort estimation: A systematic mapping study". In: *Proceedings of the 16th International Conference on Software Technologies* (ICSOFT). SCITEPRESS, 2021, pp. 102–110. DOI: 10.5220/0010603701020110.

[71] Pantjawati Sudarmaningtyas and Rozlina Binti Mohamed. "Extended planning poker: A proposed model". In: *Proceedings of the 7th International Conference on Information Technology, Computer, and Electrical Engineering* (ICITACEE 2020). 2020, pp. 179–184. DOI: 10.1109/ICITACEE50144.2020.9239165.

[72] Simone Porru, Alessandro Murgia, Serge Demeyer, Michele Marchesi and Roberto Tonelli. "Estimating story points from issue reports". In: *Proceedings of the The 12th International Conference on Predictive Models and Data Analytics in Software Engineering* (PROMISE 2016). ACM, 2016. DOI: 10.1145/2972958.2972959.

[73] Valentina Lenarduzzi, Ilaria Lunesu, Martina Matta and Davide Taibi. "Functional size measures and effort estimation in Agile development: A replicated study". In: *Proceedings of the 16th International Conference on Agile Processes in Software*

Engineering and Extreme Programming (XP 2015). Springer, 2015, pp. 105–116. DOI: 10.1007/978-3-319-18612-2_9.

[74] Pedro Miranda, J. Pascoal Faria, Filipe F. Correia, Ahmed Fares, Ricardo Graça and João Mendes Moreira. "An analysis of Monte Carlo simulations for forecasting software projects". In: *Proceedings of the 36th Annual ACM Symposium on Applied Computing* (SAC '21). ACM, 2021, pp. 1550–1558. DOI: 10.1145/3412841.3442030.

[75] Howard Lei, Farnaz Ganjeizadeh, Pradeep Kumar Jayachandran and Pinar Ozcan. "A statistical analysis of the effects of Scrum and Kanban on software development projects". *Robotics and Computer-Integrated Manufacturing*, vol. 43, 2017. Special Issue: Extended Papers Selected from FAIM 2014, pp. 59–67. DOI: 10.1016/j.rcim.2015.12.001.

[76] Martin Shepperd, David Bowes and Tracy Hall. "Researcher bias: The use of machine learning in software defect prediction". *IEEE Transactions on Software Engineering*, vol. 40, no. 6, 2014, pp. 603–616. DOI: 10.1109/TSE.2014.2322358.

[77] Maria Paasivaara, Sandra Durasiewicz and Casper Lassenius. "Using Scrum in distributed Agile development: A multiple case study". In: *Proceedings of the 4th IEEE International Conference on Global Software Engineering*. 2009, pp. 195–204. DOI: 10.1109/ICGSE.2009.27.

[78] Minna Pikkarainen, Jukka Haikara, Outi Salo, Pekka Abrahamsson and Jari Still. "The impact of Agile practices on communication in software development". *Empirical Software Engineering*, vol. 13, no. 3, 2008, pp. 303–337. DOI: 10.1007/s10664-008-9065-9.

[79] Reni Kurnia, Ridi Ferdiana and Sunu Wibirama. "Software metrics classification for Agile Scrum process: A literature review". In: *Proceedings of the 2018 International Seminar on Research of Information Technology and Intelligent Systems* (ISRITI 2018). IEEE, 2018. DOI: 10.1109/isriti.2018.8864244.

[80] Wilhelm Meding. "Effective monitoring of progress of Agile software development teams in modern software companies: An industrial case study". In: *Proceedings of the 27th International Workshop on Software Measurement and 12th International Conference on Software Process and Product Measurement*. 2017, pp. 23–32. DOI: 10.1145/3143434.3143449.

[81] Prabhat Ram, Pilar Rodriguez, Markku Oivo and Silverio Martínez-Fernández. "Success factors for effective process metrics operationalization in Agile software development: A multiple case study". In: *Proceedings of the 2019 IEEE/ACM International Conference on Software and System Processes* (ICSSP 2019). IEEE, 2019, pp. 14–23. DOI: 10.1109/icssp.2019.00013.

[82] Ezequiel Scott and Dietmar Pfahl. "Using developers' features to estimate story points". In: *Proceedings of the 2018 International Conference on Software and System Process* (ICSSP '18). ACM, 2018, pp. 106–110. DOI: 10.1145/3202710.3203160.

[83] Luis Almeida, Adriano Albuquerque and Plácido Pinheiro. "A multi-criteria model for planning and fine-tuning distributed Scrum projects". In: *Proceedings of the 6th IEEE International Conference on Global Software Engineering*. 2011, pp. 75–83. DOI: 10.1109/ICGSE.2011.36.

[84] Marko Robnik-Šikonja and Igor Kononenko. "Theoretical and empirical analysis of ReliefF and RReliefF". *Machine Learning*, vol. 53, 2003, pp. 23–69. DOI: 10.1023/A:1025667309714.

[85] Martin Tomanek and Jan Juricek. "Project risk management model based on PRINCE2 and Scrum frameworks". *International Journal of Software Engineering & Applications*, vol. 6, no. 1, 2015, pp. 81–88. DOI: 10.5121/ijsea.2015.6107.

[86] Cyril Goutte and Eric Gaussier. "A probabilistic interpretation of precision, recall and F-score, with implication for evaluation". In: *Advances in Information Retrieval*. Springer, 2005, pp. 345–359. DOI: 10.1007/978-3-540-31865-1_25.

[87] Shashank Mouli Satapathy and Santanu Kumar Rath. "Empirical assessment of machine learning models for Agile software development effort estimation using story points". *Innovations in Systems and Software Engineering*, vol. 13, no. 2, 2017, pp. 191–200. DOI: 10.1007/s11334-017-0288-z.

[88] Subhra Sankar Dhar, Biman Chakraborty and Probal Chaudhuri. "Comparison of multivariate distributions using quantile-quantile plots and related tests". *Bernoulli*, vol. 20, no. 3, 2014, pp. 1484–1506. DOI: 10.3150/13-BEJ530.

[89] Sebastian Baltes and Paul Ralph. "Sampling in software engineering research: a critical review and guidelines". *Empirical Software Engineering*, vol. 27, article 94, 2022. DOI: 10.1007/s10664-021-10072-8.

First referenced in Chapter 5

[90] Edward R. Tufte. *Envisioning Information*. 2nd ed. Graphics Press, 1998.

[91] Min Chen, David Ebert, Hans Hagen, Robert S. Laramee, Robert van Liere, Kwan-Liu Ma, William Ribarsky, Gerik Scheuermann and Deborah Silver. "Data, information, and knowledge in visualization". *IEEE Computer Graphics and Applications*, vol. 29, no. 1, 2008, pp. 12–19. DOI: 10.1109/MCG.2009.6.

[92] Usama Fayyad, Georges G. Grinstein and Andreas Wierse. *Information Visualization in Data Mining and Knowledge Discovery*. Morgan Kaufmann Publishers Inc., 2001.

[93] David P. Tegarden. "Business information visualization". *Communications of the Association for Information Systems*, vol. 1, article 4, 1999. DOI: 10.17705/1cais.00104.

[94] Ben Shneiderman, Catherine Plaisant, Maxine S. Cohen, Steven Jacobs, Niklas Elmqvist and Nicholas Diakopoulos. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. 6th ed. Pearson, 2016, pp. 66–82.

[95] Iris Vessey and Dennis Galletta. "Cognitive fit: An empirical study of information acquisition". *Information Systems Research*, vol. 2, no. 1, 1991, pp. 63–84. DOI: 10.1287/isre.2.1.63.

[96] Joseph K. Nuamah, Younho Seong, Steven Jiang, Eui Park and Daniel Mountjoy. "Evaluating effectiveness of information visualizations using cognitive fit theory: A neuroergonomics approach". *Applied Ergonomics*, vol. 88, article 103173, 2020. DOI: 10.1016/j.apergo.2020.103173.

[97] Anshul Vikram Pandey, Anjali Manivannan, Oded Nov, Margaret Satterthwaite and Enrico Bertini. "The persuasive power of data visualization". *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 12, 2014, pp. 2211–2220. DOI: 10.1109/TVCG.2014.2346419.

[98] Julia Paredes, Craig Anslow and Frank Maurer. "Information visualization for Agile software development teams". In: *Proceedings of the Second IEEE Working Conference on Software Visualization* (VISSOFT 2014). IEEE, 2014, pp. 157–166. DOI: 10.1109/vissoft.2014.32.

[99] Antonio González-Torres, Francisco J. García-Peñalvo, Roberto Therón-Sánchez and Ricardo Colomo-Palacios. "Knowledge discovery in software teams by means of evolutionary visual software analytics". *Science of Computer Programming*, vol. 121, 2016. Special Issue on Knowledge-based Software Engineering, pp. 55–74. DOI: 10.1016/j.scico.2015.09.005.

[100] Nesib Tekin, Mehmet Kosa, Murat Yilmaz, Paul Clarke and Vahid Garousi. "Visualization, monitoring and control techniques for use in Scrum software development: An Analytic Hierarchy Process approach". In: *Systems, Software and Services Process Improvement*. Springer, 2020, pp. 45–57. DOI: 10.1007/978-3-030-56441-4_4.

[101] Martin J. Eppler and Sabrina Bresciani. "Visualization in management: From communication to collaboration. A response to Zhang". *Journal of Visual Languages & Computing*, vol. 24, no. 2, 2013, pp. 146–149. DOI: 10.1016/j.jvlc.2012.11.003.

[102] Evanthia Dimara and Charles Perin. "What is interaction for data visualization?" *IEEE Transactions on Visualization and Computer Graphics*, vol. 26, no. 1, 2020, pp. 119–129. DOI: 10.1109/TVCG.2019.2934283.

[103] Riccardo Mazza. *Introduction to Information Visualization*. Springer, 2009. DOI: 10.1007/978-1-84800-219-7.

[104] Juuso Koponen and Jonatan Hildén. *Data Visualization Handbook*. Art + Design + Architecture. Aalto University, 2019.

[105] Bang Wong. "Points of view: Color blindness". *Nature Methods*, vol. 8, no. 6, 2011, pp. 441–441. DOI: 10.1038/nmeth.1618.

[106] Georges Grinstein, Alfred Kobsa, Catherine Plaisant and John T. Stasko. "Which comes first, usability or utility?" In: *Proceedings of the IEEE Conference on Visualization* (VIS 2003). IEEE, 2003, pp. 605–606. DOI: 10.1109/visual.2003.1250426.

[107] Jeffrey Heer, Stuart K. Card and James A. Landay. "prefuse: A toolkit for interactive information visualization". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '05). ACM, 2005, pp. 421–430. DOI: 10.1145/1054972.1055031.

[108] Stuart K. Card, Jock D. Mackinlay and Ben Shneiderman. *Readings in Information Visualization: Using Vision to Think*. Interactive Technologies. Morgan Kaufmann, 1999.

[109] Ben Shneiderman. "The eyes have it: A task by data type taxonomy for information visualizations". In: *The Craft of Information Visualization*. Morgan Kaufmann, 2003, pp. 364–371. DOI: https://doi.org/10.1016/B978-155860915-0/50046-9.

[110] Jakob Nielsen. *Usability Engineering*. Morgan Kaufman, 1993. DOI: 10.1016/C2009-0-21512-1.

[111] Michael Bostock, Vadim Ogievetsky and Jeffrey Heer. "$\mathbb{D}^3$: Data-Driven Documents". *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 12, 2011, pp. 2301–2309. DOI: 10.1109/TVCG.2011.185.

[112] Patrick Riehmann, Manfred Hanfler and Bernd Froehlich. "Interactive Sankey diagrams". In: *IEEE Symposium on Information Visualization* (INFOVIS 2005). 2005, pp. 233–240. DOI: 10.1109/INFVIS.2005.1532152.

[113] Amy N. Langville and Carl D. Meyer. *Who's #1?: The Science of Rating and Ranking*. Princeton University Press, 2012. DOI: 10.1515/9781400841677.

[114] Josh Barnes and Piet Hut. "A hierarchical $O(N \log N)$ force-calculation algorithm". *Nature*, vol. 324, no. 6096, 1986, pp. 446–449. DOI: 10.1038/324446a0.

[115] Loup Verlet. "Computer "experiments" on classical fluids. I. Thermodynamical properties of Lennard-Jones molecules". *Physical Review*, vol. 159, no. 1, article 98, 1967, pp. 98–103. DOI: 10.1103/PhysRev.159.98.

[116] Emden R. Gansner and Stephen C. North. "An open graph visualization system and its applications to software engineering". *Software: Practice and Experience*, vol. 30, no. 11, 2000. Special Issue: Discrete algorithm engineering, pp. 1203–1233. DOI: 10.1002/1097-024X(200009)30:11<1203::AID-SPE338>3.0.CO;2-N.

[117] Cas H. J. Dekkers. "Designing information visualizations for generating business value in Agile software development". Master's thesis. LIACS, Leiden University, 2021.

[118] Laurens C. Groeneveld. "Visalization of patterns in Scrum software development". Bachelor's thesis. LIACS, Leiden University, 2017.

[119] Donald A. Norman. *The Design of Everyday Things*. Revised and Expanded Edition. Originally published as *The Psychology of Everyday Things*. Perseus Books, 2013.

[120] Catherine Plaisant. "The challenge of information visualization evaluation". In: *Proceedings of the Working Conference on Advanced Visual Interfaces* (AVI '04). ACM, 2004, pp. 109–116. DOI: 10.1145/989863.989880.

[121] Michael Behrisch et al. "Quality metrics for information visualization". *Computer Graphics Forum*, vol. 37, no. 3, 2018, pp. 625–662. DOI: 10.1111/cgf.13446.

[122] Evanthia Dimara, Anastasia Bezerianos and Pierre Dragicevic. "Conceptual and methodological issues in evaluating multidimensional visualizations for decision support". *IEEE Transactions on Visualization and Computer Graphics*, vol. 24, no. 1, 2018, pp. 749–759. DOI: 10.1109/TVCG.2017.2745138.

Technical resources

[I] Atlassian. *Jira: Issue & project tracking software*. URL: https://www.atlassian.com/software/jira.

[II] Scott Chacon et al. *Git*. URL: https://git-scm.com/.

[III] GitLab. *The most-comprehensive AI-powered DevSecOps platform*. URL: https://about.gitlab.com/.

[IV] Continuous Delivery Foundation. *Jenkins*. URL: https://www.jenkins.io/.

[V] SonarSource. *Code quality, security & static analysis tool with SonarQube*. URL: https://www.sonarsource.com/products/sonarqube/.

[VI] ICTU. *Quality-time: Software quality monitoring for teams and projects*. URL: https://github.com/ICTU/quality-time.

[VII] ICTU. *BigBoat: An open-source container and CI/CD ecosystem*. URL: https://github.com/bigboat-io.

[VIII] Microsoft. *Azure DevOps Server*. URL: https://azure.microsoft.com/en-us/products/devops/server/.

[IX] The GnuPG Project. *The GNU Privacy Guard*. URL: https://www.gnupg.org/.

[X] The R Foundation. *The R project for statistical computing*. URL: https://www.r-project.org/.

[XI] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[XII] Docker. *Docker Compose overview*. URL: https://docs.docker.com/compose/.

[XIII] GitHub. *GitHub: Let's build from here*. URL: https://github.com/.

[XIV] Apache Software Foundation. *Apache Subversion*. URL: https://subversion.apache.org/.

[XV] TOPdesk. *IT Service Management Platform*. URL: https://www.topdesk.com/en/.

[XVI] Oracle. *MySQL Workbench*. URL: https://dev.mysql.com/doc/workbench/en/.

[XVII] Lance Andersen. *JDBC 4.2 Specification*. Oracle, 2014. URL: https://download.oracle.com/otn-pub/jcp/jdbc-4_2-mrel2-spec/jdbc4.2-fr-spec.pdf.

[XVIII] Transaction Processing Performance Council. *TPC Benchmark H (Decision Support) Standard Specification*. Apr. 28, 2022. URL: https://www.tpc.org/tpc_documents_current_versions/pdf/tpc-h_v3.0.1.pdf.

[XIX] Leon Helwerda. *Grip on Software sprint features*. 2024. DOI: 10.5281/zenodo.10878529. URL: https://gros.liacs.nl/combined/prediction/api/v1/dataset.

[XX] Eibe Frank et al. *ARFF Format*. URL: https://waikato.github.io/weka-wiki/formats_and_processing/arff/.

[XXI] Jeremy Thomas. *Bulma: Free, open source, and modern CSS framework based on Flexbox*. URL: https://bulma.io/.

[XXII] Orit Golowinski. *Understand how your teams adopt DevOps with DevOps reports*. Dec. 15, 2021. URL: https://about.gitlab.com/blog/2021/12/15/devops-adoption/.

[XXIII] Mike Bostock. *d3-force: Force-directed graph layout using velocity Verlet integration*. URL: https://github.com/d3/d3-force.

[XXIV] The Graphviz Authors. *DOT Language*. URL: https://www.graphviz.org/doc/info/lang.html.

[XXV] Magnus Jacobsson. *d3-graphviz: Graphviz DOT rendering and animated transitions using D3*. URL: https://github.com/magjac/d3-graphviz.

[XXVI] WHATWG. *HTML Standard: Web workers*. Mar. 17, 2022. URL: https://html.spec.whatwg.org/multipage/workers.html.

[XXVII] Justin Palmer. *Introducing Contributions*. Jan. 7, 2013. URL: https://github.blog/2013-01-07-introducing-contributions/.

[XXVIII] Koninklijk Nederlands Meteorologisch Instituut (KNMI). *Meteo data - daily quality controlled climate data*. URL: https://dataplatform.knmi.nl/dataset/etmaalgegevensknmistations-1.

[XXIX] Meta. *React*. URL: https://react.dev/.

[XXX] Dan Abramov. *Redux - A predictable state container for JavaScript apps*. URL: https://redux.js.org/.

[XXXI] Mark Otto, Jacob Thorton and Bootstrap contributors. *Bootstrap*. URL: https://getbootstrap.com/.

[XXXII] Deque Systems. *axe: Accessiblity Testing Tools and Software*. URL: https://www.deque.com/axe/.

[XXXIII] Andrew Kirkpatrick, Joshue O'Connor, Alastair Campbell and Michael Cooper. *Web Content Accessibility Guidelines (WCAG) 2.1*. W3C Recommendation. June 5, 2018. URL: https://www.w3.org/TR/WCAG21/.

[XXXIV] Joanmarie Diggs, James Nurthen and Michael Cooper. *Accessible Rich Internet Applications (WAI-ARIA) 1.2*. W3C Candidate Recommendation Draft. Dec. 8, 2021. URL: https://www.w3.org/TR/wai-aria-1.2/.

Appendices

Appendix A

Code repositories of the Grip on Software pipeline

Accompanying Chapter 2

The references listed here are supplemental to the technical resources found in the bibliography. We provide these separate from the bibliography, given their nature of being contributions in addition to—and in support of—our research. The references indicate locations of code repositories that contain implementations, documentation and tests for the components of the GROS pipeline used throughout our research. In Section 2.3.2, we provide descriptions and further details for each of the code repositories.

[a] Leon Helwerda. *Modules used to gather data from different data sources in software development processes*. ICTU and Leiden University. DOI: 10.5281/zenodo.10911862. URL: https://github.com/grip-on-software/data-gathering.

[b] Leon Helwerda. *Web-based data gathering agent configuration*. ICTU and Leiden University. DOI: 10.5281/zenodo.11115708. URL: https://github.com/grip-on-software/agent-config.

[c] Leon Helwerda. *Data gathering agent status web application*. ICTU and Leiden University. DOI: 10.5281/zenodo.12533335. URL: https://github.com/grip-on-software/status-dashboard.

[d] Leon Helwerda, Enrique Larios Vargas, Thomas Helling and Thomas Prikkel. *Importer of gathered data into a MonetDB database*. ICTU and Leiden University. DOI: 10.5281/zenodo.12583196. URL: https://github.com/grip-on-software/monetdb-import.

[e] Leon Helwerda. *Export tables from a MonetDB database for backups or exchanges*. ICTU and Leiden University. DOI: 10.5281/zenodo.12723675. URL: https://github.com/grip-on-software/monetdb-dumper.

[f] Leon Helwerda. *Tools for securely uploading files to a remote server via HTTPS and GPG*. ICTU and Leiden University. DOI: 10.5281/zenodo.12773659. URL: https://github.com/grip-on-software/export-exchange.

[g] Leon Helwerda. *Encrypted file upload server*. ICTU and Leiden University. DOI: 10.5281/zenodo.12784820. URL: https://github.com/grip-on-software/upload.

[h] Leon Helwerda. *Requesting (anonymized/aggregate) data from a filled MonetDB database and processing for data mining or visualization*. ICTU and Leiden University. DOI: 10.5281/zenodo.12935240. URL: https://github.com/grip-on-software/data-analysis.

[i] Leon Helwerda. *Algorithms to predict, classify and analyze features and labels of Scrum data*. ICTU and Leiden University. DOI: 10.5281/zenodo.12942716. URL: https://github.com/grip-on-software/prediction.

[j] Leon Helwerda. *Integrated visualization hub*. ICTU and Leiden University. DOI: 10.5281/zenodo.13208936. URL: https://github.com/grip-on-software/visualization-site.

[k] Leon Helwerda. *Common visualization UI fragments*. ICTU and Leiden University. URL: https://github.com/grip-on-software/visualization-ui.

[l] Leon Helwerda. *Dynamic sprint report generator in comparison visualization formats*. ICTU and Leiden University. DOI: 10.5281/zenodo.13208969. URL: https://github.com/grip-on-software/sprint-report.

[m] Leon Helwerda. *Human-readable output of sprint predictions*. ICTU and Leiden University. DOI: 10.5281/zenodo.13209623. URL: https://github.com/grip-on-software/prediction-site.

[n] Leon Helwerda. *Interactive visualization of temporal data from a software development process*. ICTU and Leiden University. DOI: 10.5281/zenodo.13220620. URL: https://github.com/grip-on-software/timeline.

[o] Leon Helwerda and Laurens C. Groeneveld. *Project statistics as a leaderboard*. ICTU and Leiden University. DOI: 10.5281/zenodo.13220623. URL: https://github.com/grip-on-software/leaderboard.

[p] Leon Helwerda and Laurens C. Groeneveld. *Graph of relations between project members and the projects they work on*. ICTU and Leiden University. DOI: 10.5281/zenodo.13220626. URL: https://github.com/grip-on-software/collaboration-graph.

[q] Leon Helwerda. *Flowchart display of story status*. ICTU and Leiden University. DOI: 10.5281/zenodo.13220648. URL: https://github.com/grip-on-software/process-flow.

[r] Leon Helwerda and Laurens C. Groeneveld. *Visualization of project commit activity over time*. ICTU and Leiden University. DOI: 10.5281/zenodo.13220681. URL: https://github.com/grip-on-software/heatmap.

[s] Leon Helwerda and Laurens C. Groeneveld. *BigBoat platform reliability graphs*. ICTU and Leiden University. DOI: 10.5281/zenodo.13220696. URL: https://github.com/grip-on-software/bigboat-status.

[t] Cas H. J. Dekkers. *Effort burndown chart for product backlogs*. ICTU and Leiden University. URL: https://github.com/grip-on-software/backlog-burndown.

[u] Cas H. J. Dekkers. *Progression inspection chart for product backlogs*. ICTU and Leiden University. URL: https://github.com/grip-on-software/backlog-progression.

[v] Cas H. J. Dekkers. *Issue relationship chart for product backlogs*. ICTU and Leiden University. URL: https://github.com/grip-on-software/backlog-relationship.

[w] Leon Helwerda. *Deployment web application*. ICTU and Leiden University. DOI: 10.5281/zenodo.12571035. URL: https://github.com/grip-on-software/deployer.

[x] Leon Helwerda. *Web application framework for building authenticated services with templating support*. ICTU and Leiden University. DOI: 10.5281/zenodo.11580150. URL: https://github.com/grip-on-software/server-framework.

[y] Leon Helwerda. *Cleanup of Docker, Jenkins and SonarQube services based on build state*. ICTU and Leiden University. URL: https://github.com/grip-on-software/jenkins-cleanup.

[z] Leon Helwerda. *Collect JavaScript coverage information during a test run*. ICTU and Leiden University. URL: https://github.com/grip-on-software/coverage-collector.

Appendix B

Queries used in database performance experiments

Accompanying Chapter 3

```
SELECT ${f(join_cols, "sprint_metrics")}, COUNT(*) AS
    num_metrics
FROM (
    SELECT DISTINCT ${f(join_cols, "metric_value")},
        metric_value.metric_id
    FROM gros.metric_value
    JOIN gros.${t("sprint")} ON ${j(join_cols,
        "metric_value", "sprint")}
    WHERE metric_value.value <> -1
) AS sprint_metrics
${g(join_cols, "sprint_metrics")}
```

(a) Template

```
SELECT sprint_metrics.project_id,
    sprint_metrics.sprint_id, COUNT(*) AS num_metrics
FROM (
    SELECT DISTINCT metric_value.project_id,
        metric_value.sprint_id, metric_value.metric_id
    FROM gros.metric_value
    JOIN gros.sprint ON metric_value.project_id =
        sprint.project_id AND metric_value.sprint_id =
        sprint.sprint_id
    WHERE metric_value.value <> -1
) AS sprint_metrics
GROUP BY sprint_metrics.project_id,
    sprint_metrics.sprint_id
```

(b) Compiled

Figure B.1: All metrics (original query)

```
1 SELECT ${f(join_cols, "sprint_metrics")}, COUNT(*) AS
      num_metrics
2 FROM (
3     SELECT DISTINCT ${f(join_cols, "metric_value")},
          metric_value.metric_id
4     FROM gros.metric_value
5     WHERE metric_value.value <> -1 AND
          metric_value.sprint_id <> 0
6 ) AS sprint_metrics
7 ${g(join_cols, "sprint_metrics")}
```

(a) Template

```
1 SELECT sprint_metrics.project_id,
      sprint_metrics.sprint_id, COUNT(*) AS num_metrics
2 FROM (
3     SELECT DISTINCT metric_value.project_id,
          metric_value.sprint_id, metric_value.metric_id
4     FROM gros.metric_value
5     WHERE metric_value.value <> -1 AND
          metric_value.sprint_id <> 0
6 ) AS sprint_metrics
7 GROUP BY sprint_metrics.project_id,
      sprint_metrics.sprint_id
```

(b) Compiled

Figure B.2: All metrics (refined query)

```
SELECT ${f(join_cols, "sprint_metrics")}, COUNT(*) AS
    num_red_metrics
FROM (
    SELECT DISTINCT ${f(join_cols, "metric_value")},
        metric_value.metric_id
    FROM gros.metric_value
    JOIN gros.${t("sprint")} ON ${j(join_cols,
        "metric_value", "sprint")}
    WHERE metric_value.category = 'red'
) AS sprint_metrics
${g(join_cols, "sprint_metrics")}
```

(a) Template

```
SELECT sprint_metrics.project_id, sprint_metrics.sprint_id,
    COUNT(*) AS num_red_metrics
FROM (
    SELECT DISTINCT metric_value.project_id,
        metric_value.sprint_id, metric_value.metric_id
    FROM gros.metric_value
    JOIN gros.sprint ON metric_value.project_id =
        sprint.project_id AND metric_value.sprint_id =
        sprint.sprint_id
    WHERE metric_value.category = 'red'
) AS sprint_metrics
GROUP BY sprint_metrics.project_id, sprint_metrics.sprint_id
```

(b) Compiled

Figure B.3: Red metrics (original query)

```
SELECT ${f(join_cols, "sprint_metrics")}, COUNT(*) AS
    num_red_metrics
FROM (
    SELECT DISTINCT ${f(join_cols, "metric_value")},
        metric_value.metric_id
    FROM gros.metric_value
    WHERE metric_value.category = 'red' AND
        metric_value.sprint_id <> 0
) AS sprint_metrics
${g(join_cols, "sprint_metrics")}
```

(a) Template

```
SELECT sprint_metrics.project_id, sprint_metrics.sprint_id,
    COUNT(*) AS num_red_metrics
FROM (
    SELECT DISTINCT metric_value.project_id,
        metric_value.sprint_id, metric_value.metric_id
    FROM gros.metric_value
    WHERE metric_value.category = 'red' AND
        metric_value.sprint_id <> 0
) AS sprint_metrics
GROUP BY sprint_metrics.project_id, sprint_metrics.sprint_id;
```

(b) Compiled

Figure B.4: Red metrics (refined query)

```
SELECT ${f(join_cols, "team_spirit")},
    AVG(metric_value.value) AS team_spirit
FROM gros.metric_value, (
    SELECT ${f(join_cols, "metric_value")},
        metric_value.metric_id, MAX(metric_value.date) AS
        max_date
    FROM gros.metric_value
    JOIN gros.metric
    ON metric_value.metric_id = metric.metric_id
    JOIN gros.${t("sprint")}
    ON ${j(join_cols, "metric_value", "sprint")}
    WHERE metric_value.value <> -1
    AND metric.base_name = 'TeamSpirit'
    ${g(join_cols, "metric_value")}, metric_value.metric_id
) AS team_spirit
WHERE metric_value.date = team_spirit.max_date AND
    metric_value.metric_id = team_spirit.metric_id
${g(join_cols, "team_spirit")}
```

(a) Template

```
SELECT team_spirit.project_id, team_spirit.sprint_id,
    AVG(metric_value.value) AS team_spirit
FROM gros.metric_value, (
    SELECT metric_value.project_id, metric_value.sprint_id,
        metric_value.metric_id, MAX(metric_value.date) AS
        max_date
    FROM gros.metric_value
    JOIN gros.metric
    ON metric_value.metric_id = metric.metric_id
    JOIN gros.sprint
    ON metric_value.project_id = sprint.project_id AND
        metric_value.sprint_id = sprint.sprint_id
    WHERE metric_value.value <> -1
    AND metric.base_name = 'TeamSpirit'
    GROUP BY metric_value.project_id,
        metric_value.sprint_id, metric_value.metric_id
) AS team_spirit
WHERE metric_value.date = team_spirit.max_date and
    metric_value.metric_id = team_spirit.metric_id
GROUP BY team_spirit.project_id, team_spirit.sprint_id
```

(b) Compiled

Figure B.5: Team spirit (original query)

```
SELECT ${f(join_cols, "team_spirit")}, MAX(value) AS team_spirit
FROM (
    SELECT ${f(join_cols, "metric_value")}, metric_value.value,
        MAX(metric_value.date) AS end_date, ROW_NUMBER() OVER (
        PARTITION BY ${f(join_cols, "metric_value")}
        ORDER BY ${f(join_cols, "metric_value")},
            MAX(metric_value.date) DESC
    ) AS rev_row FROM gros.metric_value
    JOIN gros.metric
    ON metric_value.metric_id = metric.metric_id
    WHERE metric.base_name = 'TeamSpirit' AND metric.domain_name
        <> '' AND metric_value.sprint_id <> 0
    AND metric_value.value > -1
    ${g(join_cols, "metric_value")}, metric_value.value
) AS team_spirit
WHERE rev_row = 1
${g(join_cols, "team_spirit")}
```

(a) Template

```
SELECT team_spirit.project_id, team_spirit.sprint_id, MAX(value)
    AS team_spirit
FROM (
    SELECT metric_value.project_id, metric_value.sprint_id,
        metric_value.value, MAX(metric_value.date) AS end_date,
        ROW_NUMBER() OVER (
        PARTITION BY metric_value.project_id,
            metric_value.sprint_id
        ORDER BY metric_value.project_id, metric_value.sprint_id,
            MAX(metric_value.date) DESC
    ) AS rev_row FROM gros.metric_value
    JOIN gros.metric
    ON metric_value.metric_id = metric.metric_id
    WHERE metric.base_name = 'TeamSpirit' AND metric.domain_name
        <> '' AND metric_value.sprint_id <> 0
    AND metric_value.value > -1
    GROUP BY metric_value.project_id, metric_value.sprint_id,
        metric_value.value
) AS metric_team_spirit
WHERE rev_row = 1
GROUP BY metric_team_spirit.project_id,
    metric_team_spirit.sprint_id
```

(b) Compiled

Figure B.6: Team spirit (refined query)

```
SELECT ${f(join_cols, "issue", mask=1)}, ${s(issue_key)} AS key,
    MAX(${f(join_cols, "sprint", mask=2, alias=T,
    sprint="interval_sprint")}) AS ${f(join_cols, "", mask=2, alias=F)},
    MAX(${s(story_points)}) AS story_points, MAX(${s(fix_version)}) AS
    fixversion
FROM gros.${t("issue")}
LEFT JOIN gros.${t("issue")} AS older_issue
ON ${j(issue_next_changelog, "issue", "older_issue")}
LEFT JOIN gros.${t("sprint")}
ON ${j(join_cols, "issue", "sprint")}
JOIN gros.${t("sprint")} AS interval_sprint
ON ${j(join_cols, "issue", "interval_sprint", 1)}
WHERE (${f(join_cols, "sprint", mask=2, alias="alias")} IS NULL
    OR ${s(sprint_open)} >= ${s(sprint_open, sprint="interval_sprint")}
)
AND ${s(issue_not_done)}
AND ${s(issue_backlog)}
AND ${t("issue")}.updated > ${s(sprint_open, sprint="interval_sprint")}
AND (${t("older_issue")}.changelog_id IS NULL ${s(filter_inverse,
    issue="older_issue", cond_op="OR")})
${g(join_cols, "issue", f("issue_key"), mask=1)}
```

(a) Template

```
SELECT issue.project_id, issue.key AS key, MAX(interval_sprint.sprint_id)
    AS sprint_id, MAX(CASE WHEN issue.story_points IN (-5, -1, 99, 100,
    122, 999) THEN 0 ELSE issue.story_points END) AS story_points,
    MAX(issue.fixversion) AS fixversion
FROM gros.issue
LEFT JOIN gros.issue AS older_issue
ON issue.issue_id = older_issue.issue_id AND issue.changelog_id =
    older_issue.changelog_id + 1
LEFT JOIN gros.sprint
ON issue.project_id = sprint.project_id AND issue.sprint_id =
    sprint.sprint_id
JOIN gros.sprint AS interval_sprint
ON issue.project_id = interval_sprint.project_id
WHERE (sprint.sprint_id IS NULL
    OR COALESCE(CAST(sprint.start_date AS TIMESTAMP), CURRENT_TIMESTAMP())
        >= COALESCE(CAST(interval_sprint.start_date AS TIMESTAMP),
        CURRENT_TIMESTAMP())
)
AND COALESCE(issue.resolution, 0) NOT IN (1, 10000) AND
    COALESCE(issue.status, 0) NOT IN (6, 10008)
AND issue."type" = 7
AND issue.updated > COALESCE(CAST(interval_sprint.start_date AS TIMESTAMP),
    CURRENT_TIMESTAMP())
AND (older_issue.changelog_id IS NULL)
GROUP BY issue.project_id, issue.issue_id, issue.key
```

(b) Compiled

Figure B.7: Backlog added points (original query)

```
1  SELECT ${f(join_cols, "issue", mask=1)}, ${s(issue_key)} AS
       key, MAX(${f(join_cols, "sprint", mask=2, alias=T,
       sprint="interval_sprint")}) AS ${f(join_cols, "",
       mask=2, alias=F)}, MAX(${s(story_points)}) AS
       story_points, MAX(${s(fix_version)}) AS fixversion
2  FROM gros.${t("issue")}
3  LEFT JOIN gros.${t("issue")} AS older_issue
4  ON ${j(issue_next_changelog, "issue", "older_issue")}
5  JOIN gros.${t("sprint")} AS interval_sprint
6  ON ${j(join_cols, "issue", "interval_sprint", 1)}
7  AND interval_sprint.sprint_id IN (${filter_sprint_ids})
8  AND ${t("issue")}.updated > ${s(sprint_open,
       sprint="interval_sprint")}
9  WHERE ${s(issue_not_done)}
10 AND ${s(issue_backlog)}
11 AND (${t("older_issue")}.changelog_id IS NULL
       ${s(filter_inverse, issue="older_issue", cond_op="OR")})
12 ${g(join_cols, "issue", f("issue_key"), mask=1)}
```

(a) Template

```
1  SELECT issue.project_id, issue.key AS key,
2      MAX(interval_sprint.sprint_id) AS sprint_id,
3      MAX(CASE WHEN issue.story_points IN (-5, -1, 99, 100,
           122, 999) THEN 0 ELSE issue.story_points END) AS
           story_points,
4      MAX(issue.fixversion) AS fixversion
5  FROM gros.issue
6  LEFT JOIN gros.issue AS older_issue
7  ON issue.issue_id = older_issue.issue_id AND
       issue.changelog_id = older_issue.changelog_id + 1
8  JOIN gros.sprint AS interval_sprint
9  ON issue.project_id = interval_sprint.project_id
10 AND interval_sprint.sprint_id IN (...)
11 AND issue.updated > COALESCE(CAST(interval_sprint.start_date
       AS TIMESTAMP), CURRENT_TIMESTAMP())
12 WHERE COALESCE(issue.resolution, 0) NOT IN (1, 10000) AND
       COALESCE(issue.status, 0) NOT IN (6, 10008)
13 AND issue."type" = 7
14 AND (older_issue.changelog_id IS NULL)
15 GROUP BY issue.project_id, issue.issue_id, issue.key
```

(b) Compiled

Figure B.8: Backlog added points (refined query)

```
SELECT ${f(join_cols, "sprint", alias=T, sprint="in_sprint")},
    ${t("issue")}.epic AS key, COUNT(*) AS epic_children,
    SUM(${s(story_points)}) AS story_points
FROM gros.${t("issue")}
LEFT JOIN gros.${t("issue")} AS newer_issue
ON ${j(issue_next_changelog, "newer_issue", "issue")}
LEFT JOIN gros.${t("sprint")} ON ${j(join_cols, "issue", "sprint")}
JOIN gros.${t("sprint")} AS in_sprint
ON ${j(join_cols, "issue", "in_sprint", 1)}
WHERE ${t("issue")}.epic IS NOT NULL
AND (${f(join_cols, "sprint", mask=2, alias="alias")} IS NULL OR
    ${s(sprint_open)} >= ${s(sprint_close, sprint="in_sprint")})
AND ${s(issue_story)} AND ${s(issue_not_done)}
AND ${t("issue")}.updated <= ${s(sprint_close, sprint="in_sprint")}
AND (newer_issue.updated IS NULL OR newer_issue.updated > ${s(sprint_close,
    sprint="in_sprint")})
${g(join_cols, "sprint", sprint="in_sprint")}, ${t("issue")}.epic
```

(a) Template

```
SELECT in_sprint.project_id, in_sprint.sprint_id, issue.epic AS key,
    COUNT(*) AS epic_children, SUM(CASE WHEN issue.story_points IN (-5, -1,
    99, 100, 122, 999) THEN 0 ELSE issue.story_points END) AS story_points
FROM gros.issue
LEFT JOIN gros.issue AS newer_issue
ON newer_issue.issue_id = issue.issue_id AND newer_issue.changelog_id =
    issue.changelog_id + 1
LEFT JOIN gros.sprint ON issue.project_id = sprint.project_id AND
    issue.sprint_id = sprint.sprint_id
JOIN gros.sprint AS in_sprint
ON issue.project_id = in_sprint.project_id
WHERE issue.epic IS NOT NULL
AND (sprint.sprint_id IS NULL OR COALESCE(CAST(sprint.start_date AS
    TIMESTAMP), CURRENT_TIMESTAMP()) >= CASE WHEN in_sprint.complete_date
    IS NOT NULL AND CAST(in_sprint.complete_date AS DATE) <
    CAST(in_sprint.end_date AS DATE) THEN in_sprint.complete_date ELSE
    in_sprint.end_date END)
AND issue."type" = 7 AND COALESCE(issue.resolution, 0) NOT IN (1, 10000)
    AND COALESCE(issue.status, 0) NOT IN (6, 10008)
AND issue.updated <= CASE WHEN in_sprint.complete_date IS NOT NULL AND
    CAST(in_sprint.complete_date AS DATE) < CAST(in_sprint.end_date AS
    DATE) THEN in_sprint.complete_date ELSE in_sprint.end_date END
AND (newer_issue.updated IS NULL OR newer_issue.updated > CASE WHEN
    in_sprint.complete_date IS NOT NULL AND CAST(in_sprint.complete_date AS
    DATE) < CAST(in_sprint.end_date AS DATE) THEN in_sprint.complete_date
    ELSE in_sprint.end_date END)
GROUP BY in_sprint.project_id, in_sprint.sprint_id, issue.epic
```

(b) Compiled

Figure B.9: Backlog epic points (original query)

```
SELECT ${f(join_cols, "sprint", alias=T, sprint="in_sprint")},
    ${t("issue")}.epic AS key, COUNT(*) AS epic_children,
    SUM(${s(story_points)}) AS story_points
FROM gros.${t("issue")}
LEFT JOIN gros.${t("issue")} AS newer_issue
ON ${j(issue_next_changelog, "newer_issue", "issue")}
LEFT JOIN gros.${t("sprint")} ON ${j(join_cols, "issue", "sprint")}
JOIN gros.${t("sprint")} AS in_sprint
ON ${j(join_cols, "issue", "in_sprint", 1)}
AND in_sprint.sprint_id IN (${filter_sprint_ids})
AND ${t("issue")}.updated <= ${s(sprint_close, sprint="in_sprint")}
AND COALESCE(newer_issue.updated, ${s(sprint_close, sprint="in_sprint")})
    >= ${s(sprint_close, sprint="in_sprint")}
WHERE ${t("issue")}.epic IS NOT NULL AND (${f(join_cols, "sprint", mask=2,
    alias="alias")} IS NULL OR ${s(sprint_open)} >= ${s(sprint_close,
    sprint="in_sprint")}) AND ${s(issue_story)} AND ${s(issue_not_done)}
${g(join_cols, "sprint", sprint="in_sprint")}, ${t("issue")}.epic
```

(a) Template

```
SELECT in_sprint.project_id, in_sprint.sprint_id, issue.epic AS key,
    COUNT(*) AS epic_children, SUM(CASE WHEN issue.story_points IN (-5, -1,
    99, 100, 122, 999) THEN 0 ELSE issue.story_points END) AS story_points
FROM gros.issue
LEFT JOIN gros.issue AS newer_issue
ON newer_issue.issue_id = issue.issue_id AND newer_issue.changelog_id =
    issue.changelog_id + 1
LEFT JOIN gros.sprint ON issue.project_id = sprint.project_id AND
    issue.sprint_id = sprint.sprint_id
JOIN gros.sprint AS in_sprint
ON issue.project_id = in_sprint.project_id
AND in_sprint.sprint_id IN (...)
AND issue.updated <= CASE WHEN in_sprint.complete_date IS NOT NULL AND
    CAST(in_sprint.complete_date AS DATE) < CAST(in_sprint.end_date AS
    DATE) THEN in_sprint.complete_date ELSE in_sprint.end_date END
AND COALESCE(newer_issue.updated, CASE WHEN in_sprint.complete_date IS NOT
    NULL AND CAST(in_sprint.complete_date AS DATE) <
    CAST(in_sprint.end_date AS DATE) THEN in_sprint.complete_date ELSE
    in_sprint.end_date END) >= CASE WHEN in_sprint.complete_date IS NOT
    NULL AND CAST(in_sprint.complete_date AS DATE) <
    CAST(in_sprint.end_date AS DATE) THEN in_sprint.complete_date ELSE
    in_sprint.end_date END
WHERE issue.epic IS NOT NULL AND (sprint.sprint_id IS NULL OR
    COALESCE(CAST(sprint.start_date AS TIMESTAMP), CURRENT_TIMESTAMP()) >=
    CASE WHEN in_sprint.complete_date IS NOT NULL AND
    CAST(in_sprint.complete_date AS DATE) < CAST(in_sprint.end_date AS
    DATE) THEN in_sprint.complete_date ELSE in_sprint.end_date END) AND
    issue."type" = 7 AND COALESCE(issue.resolution, 0) NOT IN (1, 10000)
    AND COALESCE(issue.status, 0) NOT IN (6, 10008)
GROUP BY in_sprint.project_id, in_sprint.sprint_id, issue.epic
```

(b) Compiled

Figure B.10: Backlog epic points (refined query)

```
SELECT ${f(join_cols, "sprint", alias=T, sprint="in_sprint")},
    ${s(issue_key)} AS key, MAX(${s(story_points)}) AS story_points,
    MAX(${s(fix_version)}) AS fixversion
FROM gros.${t("issue")}
LEFT JOIN gros.${t("issue")} AS newer_issue
ON ${j(issue_next_changelog, "newer_issue", "issue")}
LEFT JOIN gros.${t("sprint")}
ON ${j(join_cols, "issue", "sprint")}
JOIN gros.${t("sprint")} AS in_sprint
ON ${j(join_cols, "issue", "in_sprint", 1)}
WHERE (${s(issue_open)} OR ${s(sprint_open)} >= ${s(sprint_open,
    sprint="in_sprint")})
AND ${s(issue_backlog)}
AND ${t("issue")}.updated <= ${s(sprint_open, sprint="in_sprint")}
AND (newer_issue.updated IS NULL OR newer_issue.updated > ${s(sprint_open,
    sprint="in_sprint")})
${g(join_cols, "sprint", f("issue_key"), sprint="in_sprint")}
```

(a) Template

```
SELECT in_sprint.project_id, in_sprint.sprint_id, issue.key AS key,
    MAX(CASE WHEN issue.story_points IN (-5, -1, 99, 100, 122, 999) THEN 0
    ELSE issue.story_points END) AS story_points, MAX(issue.fixversion) AS
    fixversion
FROM gros.issue
LEFT JOIN gros.issue AS newer_issue
ON newer_issue.issue_id = issue.issue_id AND newer_issue.changelog_id =
    issue.changelog_id + 1
LEFT JOIN gros.sprint
ON issue.project_id = sprint.project_id AND issue.sprint_id =
    sprint.sprint_id
JOIN gros.sprint AS in_sprint
ON issue.project_id = in_sprint.project_id
WHERE (issue.status NOT IN (5,6,10008) OR COALESCE(CAST(sprint.start_date
    AS TIMESTAMP), CURRENT_TIMESTAMP()) >=
    COALESCE(CAST(in_sprint.start_date AS TIMESTAMP), CURRENT_TIMESTAMP()))
AND issue."type" = 7 AND issue.story_points IS NOT NULL
AND issue.updated <= COALESCE(CAST(in_sprint.start_date AS TIMESTAMP),
    CURRENT_TIMESTAMP())
AND (newer_issue.updated IS NULL OR newer_issue.updated >
    COALESCE(CAST(in_sprint.start_date AS TIMESTAMP), CURRENT_TIMESTAMP()))
GROUP BY in_sprint.project_id, in_sprint.sprint_id, issue.issue_id,
    issue.key
```

(b) Compiled

Figure B.11: Backlog story points (original query)

```
SELECT ${f(join_cols, "sprint", alias=T, sprint="in_sprint")},
    ${s(issue_key)} AS key, MAX(${s(story_points)}) AS story_points,
    MAX(${s(fix_version)}) AS fixversion
FROM gros.${t("issue")}
LEFT JOIN gros.${t("issue")} AS newer_issue
ON ${j(issue_next_changelog, "newer_issue", "issue")}
LEFT JOIN gros.${t("sprint")}
ON ${j(join_cols, "issue", "sprint")}
JOIN gros.${t("sprint")} AS in_sprint
ON ${j(join_cols, "issue", "in_sprint", 1)}
AND in_sprint.sprint_id IN (${filter_sprint_ids})
AND ${t("issue")}.updated <= ${s(sprint_close, sprint="in_sprint")}
AND COALESCE(newer_issue.updated, ${s(sprint_close, sprint="in_sprint")})
    >= ${s(sprint_close, sprint="in_sprint")}
WHERE (${s(issue_open)} OR ${s(sprint_close)} >= ${s(sprint_close,
    sprint="in_sprint")}) AND ${s(issue_backlog)}
${g(join_cols, "sprint", f("issue_key"), sprint="in_sprint")}
```

(a) Template

```
SELECT in_sprint.project_id, in_sprint.sprint_id, issue.key AS key,
    MAX(CASE WHEN issue.story_points IN (-5, -1, 99, 100, 122, 999) THEN 0
    ELSE issue.story_points END) AS story_points, MAX(issue.fixversion) AS
    fixversion
FROM gros.issue
LEFT JOIN gros.issue AS newer_issue
ON newer_issue.issue_id = issue.issue_id AND newer_issue.changelog_id =
    issue.changelog_id + 1
LEFT JOIN gros.sprint
ON issue.project_id = sprint.project_id AND issue.sprint_id =
    sprint.sprint_id
JOIN gros.sprint AS in_sprint
ON issue.project_id = in_sprint.project_id
AND in_sprint.sprint_id IN (...)
AND issue.updated <= CASE WHEN in_sprint.complete_date IS NOT NULL AND
    CAST(in_sprint.complete_date AS DATE) < CAST(in_sprint.end_date AS
    DATE) THEN in_sprint.complete_date ELSE in_sprint.end_date END
AND COALESCE(newer_issue.updated, CASE WHEN in_sprint.complete_date IS NOT
    NULL AND CAST(in_sprint.complete_date AS DATE) <
    CAST(in_sprint.end_date AS DATE) THEN in_sprint.complete_date ELSE
    in_sprint.end_date END) >= CASE WHEN in_sprint.complete_date IS NOT
    NULL AND CAST(in_sprint.complete_date AS DATE) <
    CAST(in_sprint.end_date AS DATE) THEN in_sprint.complete_date ELSE
    in_sprint.end_date END
WHERE (issue.status NOT IN (5,6,10008) OR CASE WHEN sprint.complete_date IS
    NOT NULL AND CAST(sprint.complete_date AS DATE) < CAST(sprint.end_date
    AS DATE) THEN sprint.complete_date ELSE sprint.end_date END >= CASE
    WHEN in_sprint.complete_date IS NOT NULL AND
    CAST(in_sprint.complete_date AS DATE) < CAST(in_sprint.end_date AS
    DATE) THEN in_sprint.complete_date ELSE in_sprint.end_date END) AND
    issue."type" = 7 AND issue.story_points IS NOT NULL
GROUP BY in_sprint.project_id, in_sprint.sprint_id, issue.issue_id,
    issue.key
```

(b) Compiled

Figure B.12: Backlog story points (refined query)

Summary

Grip on Software: Understanding development progress of SCRUM *sprints and backlogs*

Complexity is a common factor in software development processes. Software developers, quality engineers and other leading roles often face challenges during development, in particular when balancing the effort spent in various tasks surrounding the implementation and maintenance of features in software products. These experts deal with an ever-changing digital ecosystem as well as priority shifts in wishes of the user representatives, i.e., the client.

In order to remain focused on implementing requirements that are described in user stories during a development phase—not rigid plans formulated early on in the life cycle—software development teams often choose to work according to an Agile software development method such as SCRUM. We wish to improve the predictability of the short-term SCRUM sprint cycle and the long-term product backlog planning, while retaining the flexibility of the process.

The essence of SCRUM is that repeated series of meetings and events take place. This allows us to extract structured data regarding the progress from several systems that each team uses as a digital support. After data acquisition and database modeling, we describe and select relevant metrics and features for a data set, aimed at improving and evaluating machine learning algorithms.

We introduce several algorithms, firstly for predicting work that a team could commit to finishing before the end of a SCRUM sprint, which takes a few weeks. Additionally, we explore forecasting algorithms that estimate the amount of work on the backlog over a longer period of time. The results from these models demonstrate that we are able to indicate whether selected user stories—with expert effort estimations in the form of story points—will be finished during a sprint, at a reasonable accuracy. We also show that it is more difficult to determine if certain milestones in the future are reachable, e.g., due to unforeseen scope changes. Still, these algorithms and data sets allow us to highlight patterns from a software development project's life cycle. This way, we provide novel, interactive information visualizations that aid in decision making.

In order to increase the extent of the data set and thus feed our algorithms with fresh data, we construct a data acquisition pipeline for our Grip on Software (GROS) research. The GROS pipeline is designed to be generalizable, so that it is able to collect data from different systems that teams and organizations work with. At two governmental software development organizations based in the Netherlands—Stichting ICTU and Wigo4it—we apply the components of the pipeline to acquire and augment the data sets through frequent runs of distributed agents at the organization. A centralized instance receives updates as well for a combined data set. We consider privacy and security aspects by (pseudo-)anonymizing project-sensitive data and personally identifying information.

After fresh data is acquired from systems related to project tracking (Jira and Azure DevOps), code versioning (such as Git and including review systems like GitHub and Gitlab), quality control (SonarQube and Quality-time), build platforms and so on, we model the data based on similarities between systems and establish new connections across those systems, so that we are able to derive emergent metrics based on intersections of the software development process data. This model leads to the construction of a MonetDB database, where large numbers of records for entities and relations from the software development process are easily inserted, updated and retrieved in batches using column-based storage.

The actual extraction and feature selection of features for the data set takes place using a novel query template compilation system. The templates are usable for various sources of data that are similar to each other, regardless of whether the data is modeled slightly differently, thus supporting diverse development ecosystems at the organizations that are involved. Common variables are shared between queries, simplifying the task to define, e.g., effort estimation values from story points while reducing noise from real-life data.

We apply these query templates to our GROS research to construct novel features, based on input from developers, SCRUM coaches and quality engineers, to describe various metrics, team properties and events that take place during a sprint in numeric form. We split up the samples of sprints in our data set into training, test and validation sets, taking into account the temporal aspect of the data. By only using older sprints of a development project during training, we avoid problems where a team's effort in a later sprint was influenced by the result of an earlier sprint which should thus not show up in test or validation.

As part of finetuning the data set and models, the extracted features are scored to determine which ones contribute the most in estimating a target label, e.g., an indication of whether stories are done at the end of a sprint. Additionally, we apply one of our models to select a representative subset of features based on a distance measure, similar to clustering.

Due to the limitation in dimensions, we consider classification and estimation models which apply practices from deep learning, in particular architectures that are explainable to stakeholders. We use a three-layered neural network (DNN) for classification—with an F1 score of 89.98%—and analysis-based effort estimation (ABE) for the evaluation of the number of story points that the team could complete during a sprint, with the latter model providing estimations that are within a 25% margin for 88.6% of one organization's validation samples. The stability of the ABE model is however problematic, as it is unable to handle data combined from multiple organizations, whereas DNN classification exhibits better statistical measurements with larger data sets.

When it comes to forecasting backlog sizes in the longer term, we compare a linear regression with Monte Carlo simulations using various scenarios, taking more data from earlier backlog progression and other mutations into account. We find that the Monte Carlo method is able to generate a proper normal distribution of outcomes, matching the eventual end of a project most closely, with only a third of a project's life span provided to feed the simulations. Still, all models underestimate the backlog size, with a likely cause being scope changes of a development project.

The data set and prediction results form the basis for a hub of information visualizations, which provide refreshing views on patterns and situations from development processes. The interactive visualizations include customizable reports, detailed prediction dashboards, timelines, network graphs, flow charts and calendars with heat maps. These layouts are meant to showcase different aspects of the modeled processes. Based on discussions with stakeholders, we also realize controls to provide the details that they need. This service allows intuitive access to the results, with focus on the most relevant features and options to zoom into fine-grained information.

Furthermore, we integrate status metrics from prediction results into quality reports as well as create inventive backlog charts for display in project management systems, with links and references to the visualizations and the source data. Usability and accessibility tests show that the design is effective, with several stakeholders adopting the visualizations in their workflow.

Overall, these approaches allow us to provide answers to our research questions and accomplish objectives laid out in this thesis, with the central focus on understanding and improving the SCRUM software development process, in particular the predictability of delivering incremental changes during sprints and viable products at long-term milestones. Through extraction of data and analysis of features describing events, we are able to create a data set and construct models aimed at various predictive tasks. The data acquisition pipeline and pattern recognition methods are augmented with a visualization front-end, which initiates a new feedback loop involving stakeholders of the development project. Using rapid data acquisition, database storage and analysis in our integrated pipeline, we are able to expand our data set regularly, leading to new classifications, estimations and information visualizations at a daily—or more frequent—rate. Together, this paves the way to an enhanced SCRUM software development progress.

Samenvatting

Grip op Software: Voortgang van ontwikkeling van SCRUM *sprints en backlogs beter begrijpen*

Complexiteit is alledaags in softwareontwikkelprocessen. Softwareontwikkelaars, kwaliteitsengineers en andere leidinggevenden ondervinden vaak uitdagingen tijdens de ontwikkeling, in het bijzonder het balanceren van inspanning die gemoeid gaat met allerlei taken behorende bij het implementeren en onderhouden van features in softwareproducten. Deze experts moeten omgaan met een digitaal ecosysteem dat altijd in verandering is, evenals met verschuivingen van prioriteit bij de wensen van de vertegenwoordigers van de gebruiker, zijnde de cliënt.

Om tijdens een ontwikkelfase gefocust te blijven op het implementeren van behoeftes, beschreven in user stories—niet vastgelegde plannen die lang geleden in de duur van het project zijn geformuleerd—kiezen softwareontwikkelteams er vaak voor om te werken volgens een Agile ontwikkelmethode zoals SCRUM. We willen de voorspelbaarheid van de sprintcyclus binnen SCRUM op de korte termijn en de planning van de product backlog op de lange termijn verbeteren en tegelijkertijd de flexibiliteit van het proces behouden.

De essentie van SCRUM zorgt ervoor dat herhaalde opeenvolgingen van bijeenkomsten en gebeurtenissen plaatsvinden. Dit maakt het mogelijk voor ons om gestructureerde informatie over de voortgang te extraheren uit verschillende systemen die de teams gebruiken als digitaal hulpmiddel. Na het vergaren van de data en het modelleren in een database kunnen we relevante metrieken en features beschrijven en selecteren voor een dataset, met als doel het verbeteren en evalueren van algoritmes uit machine learning.

We introduceren meerdere algoritmes, ten eerste voor het voorspellen van werk waarvoor een team een inzet zou kunnen afspreken om af te hebben voor het einde van een sprint in SCRUM, welke een paar weken duurt. Daarnaast onderzoeken we algoritmes voor het inschatten van de hoeveelheid werk op de backlog voor een langere tijd. De resultaten van deze modellen laten zien dat we kunnen aangeven of gekozen user stories—die story points krijgen als inschattingen voor de inspanning door experts—met redelijke betrouwbaarheid worden voltooid tijdens een sprint. We tonen ook aan dat het moeilijker is om te bepalen of bepaalde milestones in de toekomst haalbaar zijn, bijvoorbeeld door onvoorziene veranderingen in projectdoelen. Toch stellen deze algoritmes en datasets ons in staat om patronen uit de levenscyclus van een softwareontwikkelproject te benadrukken. Hiermee maken we nieuwe, interactieve informatievisualisaties beschikbaar die beslissingen ondersteunen.

Om de omvang van de dataset te vergroten en daarmee de algoritmes van verse, nieuwe data te voorzien, bouwen we een pipeline gericht op gegevensverzameling voor ons onderzoek, Grip op Software (GROS). De GROS-pipeline is ontworpen met het uitgangspunt van generaliseerbaarheid,

zodat het mogelijk is om data uit verschillende systemen te halen waar verschillende teams en organisaties mee werken. Bij twee Nederlandse overheidsdiensten voor IT—Stichting ICTU en Wigo4it—gebruiken we de componenten van de pipeline om datasets te verwerven en uit te breiden door regelmatig gedistribueerde agent-programma's data te laten verzamelen bij de organisatie. Een centrale opstelling verkrijgt eveneens bijgewerkte gegevens voor een gecombineerde dataset. We houden rekening met belangen rondom privacy en beveiliging door gevoelige informatie van projecten en persoonsgegevens te (pseudo-)anonimiseren.

Nadat nieuwe data is verzameld uit systemen rondom projectbeheer (Jira en Azure DevOps), versies van code (zoals Git, inclusief samenwerkingsfuncties uit GitHub en GitLab), kwaliteitscontrole (SonarQube en Quality-time), bouwplatforms, enzovoort, modelleren we de data op basis van overeenkomsten tussen systemen en leggen we nieuwe verbanden tussen deze systemen, zodat we opkomende metrieken kunnen afleiden uit overlappingen binnen data uit softwareontwikkelprocessen. Dit model leidt tot de inrichting van een MonetDB-database, waar grote aantallen registraties voor entiteiten en relaties uit het proces kunnen worden ingevoegd, bijgewerkt en massaal opgehaald met behulp van kolomgebaseerde opslag.

Het daadwerkelijke afleiden en selecteren van features voor de dataset vindt plaats met behulp van een nieuw opvraagsysteem met gecompileerde sjablonen. Deze sjablonen zijn bruikbaar om verschillende, op elkaar lijkende databronnen op te vragen, ongeacht of de data iets anders is gemodelleerd, om op die manier gevarieerde ecosystemen voor ontwikkeling bij de betrokken organisaties te ondersteunen. Gemeenschappelijke variabelen worden tussen de opvraagsjablonen gedeeld, wat het eenvoudiger maakt om bijvoorbeeld de inschattingen van inspanning op basis van story points te definiëren en tegelijkertijd ruis van de dagelijkse gang van zaken te verminderen.

We passen deze opvraagsjablonen toe op ons onderzoek om nieuwe fetures te maken, gebaseerd op adviezen van ontwikkelaars, SCRUM-coaches en kwaliteitsengineers, om tot een numerieke beschrijving voor verschillende metrieken, teameigenschappen en gebeurtenissen uit een sprint te komen. We splitsen onze dataset met voorbeelden van sprints op in sets voor trainen, testen en validatie, waarbij we rekening houden met het tijdsaspect van de data. Door alleen oudere sprints van een ontwikkelproject te gebruiken tijdens het trainen voorkomen we problemen waar de inspanning van een team in een latere sprint beïnvloed was door het resultaat van een eerdere sprint; deze sprint hoort dus niet voorbij te komen tijdens het testen en valideren.

Om de dataset en modellen te verfijnen, krijgen de afgeleide features ook scores die bepalen welke features het meest bijdragen aan het inschatten van een verwacht label, bijvoorbeeld een indicatie of stories klaar zijn aan het einde van een sprint. Daarnaast gebruiken we één van onze modellen om een representatieve deelverzameling van features te kiezen gebaseerd op een afstandsmaat, vergelijkbaar met clusteren.

Door de beperkingen in dimensies gebruiken we modellen voor classificatie en inschatting die gebruik maken van praktijken uit deep learning, met in het bijzonder structuren die uitlegbaar zijn aan belanghebbenden. We gebruiken een drielaags neural netwerk (DNN) voor classificatie—met een F1-score van 89.98%—en analogie-gebaseerde inspanningsschatting (ABE) voor het bepalen van het aantal story points dat het team zou kunnen afmaken tijdens een sprint, met inschattingen van het laatstgenoemde model die binnen een 25% marge voor 88.6% van de sprints in de validatieset van één organisatie zijn. De stabiliteit van het ABE-model is echter problematisch, omdat het niet overweg kan met gecombineerde data van meerdere organisaties, terwijl de classificatie van DNN statistisch gezien verbetert met grotere datasets.

Voor het, op langere termijn, voorspellen van backloggroottes, vergelijken we een lineaire regressie met Monte Carlo-simulaties op basis van verschillende scenario's, die meer data uit

eerdere voortgang van de backlog en andere veranderingen meenemen. We zien dat de Monte Carlo-methode in staat is om een juiste normaalverdeling van uitkomsten te genereren die het best lijkt op een uiteindelijke voltooiing van een project, met slechts een derde van het verloop van een project om de simulaties op te starten. Echter onderschatten alle modellen de backloggrootte, met als vermoedelijke reden veranderingen in projectdoelen.

De dataset en resultaten van de voorspellingen zijn de basis voor een centrale locatie van informatievisualisaties, die elk een vernieuwende blik bieden op patronen en situaties gevonden in ontwikkelprocessen. De interactieve visualisaties omvatten aanpasbare rapporten, panelen met gedetailleerde voorspellingen, tijdslijnen, netwerken als grafen, stroomschema's en kalenders met activiteitsniveaus. Deze ontwerpen dienen om verschillende aspecten van het gemodelleerde proces te belichten. Op basis van gesprekken met belanghebbenden maken we extra hulpgereedschappen beschikbaar voor de details die zij nodig hebben. Zo bieden wij intuïtieve toegang tot de resultaten, met focus op de belangrijkste onderdelen en opties om in te zoomen op fijnmazige informatie.

Daarnaast integreren we statusmetrieken van voorspellingen in kwaliteitsrapporten en bouwen we nieuwe grafieken voor backlogs die getoond kunnen worden binnen systemen voor projectbeheer, met links en referenties naar de visualisaties en brongegevens. Tests voor bruikbaarheid en toegankelijkheid tonen aan dat de opzet effectief is, met meerdere belanghebbenden die de visualisaties in hun werkwijze opnemen.

Over het geheel genomen hebben deze benaderingswijzen ervoor gezorgd dat we antwoorden en oplossingen kunnen geven voor onze onderzoeksvragen en doelen uit dit proefschrift, met de centrale focus op het begrijpen en verbeteren van het SCRUM-softwareontwikkelproces, met name de voorspelbaarheid van het leveren van stapsgewijze veranderingen tijdens sprints en bruikbare producten bij milestones op de langere termijn. Met behulp van het extraheren en analyseren van features die gebeurtenissen beschrijven maken wij een dataset en bouwen wij modellen gericht op verschillende voorspellingstaken. De pipeline voor gegevensverzameling en patroonherkenningsmethodes worden aangevuld met visualisaties in een gebruikersomgeving, wat een nieuwe terugkoppeling opstart met de belanghebbenden in het ontwikkelproject. Door de gegevensverzameling, database-opslag en analyse te versnellen, vergroten we de dataset regelmatig, wat leidt tot classificaties, inschattingen en informatievisualisaties die dagelijks—of vaker—worden vernieuwd. Alles bij elkaar genomen maakt dit de weg vrij voor een verbeterde SCRUM-softwareontwikkelmethode.

Curriculum Vitae

Leon Helwerda was born on the 23rd of June, 1992 in Voorburg, the Netherlands. He graduated from the Huygenslyceum in the same locality in 2010. Following that, he enjoyed a year of Mathematics education at Leiden University before switching to Computer Science, obtaining his BSc degree in *Informatica* in 2014 with extracurricular courses included. Following that, he graduated *cum laude* with an MSc degree in Computer Science—Core Computer Science specialization—in 2016. While this education mainly took place at Leiden University, the Master's thesis included experiments performed at CWI, Amsterdam. Continuing his research efforts at the Leiden Institute of Advanced Computer Science (LIACS), Leon embarked on another journey as a PhD student in the *Grip op Software* (GROS) research project, a collaboration between LIACS and Stichting ICTU. As a part of the Imaging and Bio-informatics group, he collaborated with other staff members to set up and maintain systems and aid with research and education.

Publications

- Leon Helwerda et al. “Query compilation for feature extraction in MonetDB”, 2024. Pending submission.
- Leon Helwerda et al. “Estimation models for prediction of sprints and backlogs”. *Empirical Software Engineering*, 2024. Submitted.
- Leon Helwerda et al. “Information visualization in analytical decision support and ecosystem management in agile processes”, 2024. Pending submission.
- Leon Helwerda, Frank Niessink and Fons J. Verbeek. “Conceptual process models and quantitative analysis of classification problems in Scrum software development practices”. In: *Proceedings of the 9th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management* (IC3K 2017 - KDIR). SCITEPRESS, 2017, pp. 357–366. DOI: 10.5220/0006602803570366.

Leon also co-authored the following publications:

- Alan Zammit, Leon Helwerda, René C. L. Olsthoorn, Fons J. Verbeek and Alexander P. Gultyaev. “A database of flavivirus RNA structures with a search algorithm for pseudoknots and triple base interactions”. *Bioinformatics*, vol. 37, no. 7, Aug. 2020, pp. 956–962. DOI: 10.1093/bioinformatics/btaa759.
- Irene Martorelli, Leon Helwerda, Jesse Kerkvliet, Sofia I. F. Gomes, Jorinde Nuytinck, Chivany R. A. van der Werff, Guus J. Ramackers, Alexander P. Gultyaev, Vincent S. F. T. Merckx and Fons J. Verbeek. “Fungal metabarcoding data integration framework for the MycoDiversity DataBase (MDDB)”. *Journal of Integrative Bioinformatics*, vol. 17, no. 1, article 20190046, 2020. DOI: 10.1515/jib-2019-0046.
- K. Joost Batenburg, Leon Helwerda, Walter A. Kosters and Tim van der Meij. “Mobile radio tomography: Agent-based imaging”. In: *BNAIC 2016: Artificial Intelligence*. Communications in Computer and Information Science, vol. 765. Springer, 2017, pp. 63–77. DOI: 10.1007/978-3-319-67468-1_5.

Acknowledgments

During the extended amount of time that it took to finalize the research—in particular writing this dissertation—many people have offered support in one way or another. Without this generous encouragement, it would be hard to imagine in what state the project would be today. Throughout this process, I found that motivation is drawn not only from energy that I am able to exert from myself, but also by the positivity and confidence that others have clearly conveyed. In this section, as the author of this thesis, I would like to thank those that have proven to play an essential role in the realization of this work. I mention some of these people by group, so even if their name is not included here, they still have my greatest appreciation.

Firstly, I thank the people involved in *Grip on Software*. As a project, it has gone through various phases, with pilots and subprojects that inevitably led to people joining and leaving again. But I rather consider GROS as a team, where ambitions and visions have brought it to greater heights. Here, I focus on GROS members from LIACS, but I mention participation from ICTU later on in this section. I thank Fons for proposing the PhD position to me and providing a place where I could grow my skills, both in machine learning research and computer systems in general. Walter, who only officially joined the team late on, has been a cornerstone throughout my education and research, with supervision of both Bachelor and Master thesis as a few indicators. As such, I am glad that the recognition of the role he played all these years that led to this point is now in print as well. I will also mention Aske, who helped kick-start and oversee the project in its early phase. The pilot project—in which Enrique, Thomas and Thomas were involved—helped start this research by demonstrating the potential of data acquisition. By looking beyond data, we found that information visualization would best work toward understanding SCRUM, and work by Laurens helped a lot there. Last but not least, Cas brought along more expertise regarding visualization for software development as well as more help that I could ever know to ask for in the final stages. This has been a source of inspiration during struggles.

There are more people at Leiden University that deserve some credit in helping me through various stages of this process. Due to closeness of my initial office to other people and shared interests with them, an informal group quickly formed, with fitting, unofficial names, ranging from Coffee club—as most enjoyed this beverage during breaks—to the *Fish tank* crew, referring to the office where most of them relocated to later on. In particular, I mention Lise and Irene here, who invited me into the group in the first place, which also included Rens, Dirk, Arie-Willem, Jan, Zé and many others who have come and gone. In the later stages of my PhD track, Kees and Alexandra helped me to find a route through an entanglement of fuzzy paths. Even if some advice was difficult to put in practice for me, it was still appreciated. I would also like to thank Alice for lending a listening ear at various times. Long chats with Chivany are always a nice change of pace, with many topics passing by. I would always feel comfortable during those times with you, Shiv.

More broadly, I have enjoyed working together with staff and students from the *Bio-imaging group*, who have passion for their topics which might feel distanced from my research, but the overlaps are plentiful and collaborations were always productive. This also includes Lu, who has been very understanding of tough situations. More staff to mention here are Jeannette and Jetty, for whom I have happily assisted courses. This is also the case for Kris, where our collaboration extended into technical work as well. I also want to acknowledge the other staff that have been involved in the LIACS *Research and education lab*, namely Vian and Jur. During my final year, I joined the ISSC ALICE team to help with documentation and GPU monitoring for their cluster, and my time with the team was a constructive and enjoyable experience.

Aside from staff and students that were there during various phases of my PhD, there are also the students from earlier years that kept in touch. Of these, it is relevant to at least name Tim, Jerome and Simon, with whom I had good interactions with during our studies and beyond.

As for the GROS project itself, it could not have been anything concrete without the support and input from people over at ICTU. Foremost, Frank has been insightful and practical, with experience and knowledge to back up his advice. Additionally, our meetings and informal talks with quality managers, software delivery managers, Scrum Masters and coaches—as well as other people whose roles do not define them—have been thoughtful, fruitful and compassionate. Particularly, I want to thank the development teams that realize the software on behalf of ICTU for other government agencies. All involved parties have been considerate and permissive to allow us to use their data. I also thank the support teams for enabling a novel setup to acquire the data using decentralized agents in the development ecosystem that they maintained.

Similarly, I thank the people at Wigo4it who helped in our research. Talking with various people to understand the differences between the two governmental organizations brought extra qualitative data, but also new viewpoints to consider.

Finally, I praise my family, namely my parents, Remco and Marian, as well as my sisters, Daphne and Renate. They have shown engagement and—once I found how to ask—patience and flexibility, which has been important throughout this process.

SIKS Dissertation Series

2016
01 Syed Saiden Abbas (RUN), Recognition of Shapes by Humans and Machines
02 Michiel Christiaan Meulendijk (UU), Optimizing medication reviews through decision support: prescribing a better pill to swallow
03 Maya Sappelli (RUN), Knowledge Work in Context: User Centered Knowledge Worker Support
04 Laurens Rietveld (VUA), Publishing and Consuming Linked Data
05 Evgeny Sherkhonov (UvA), Expanded Acyclic Queries: Containment and an Application in Explaining Missing Answers
06 Michel Wilson (TUD), Robust scheduling in an uncertain environment
07 Jeroen de Man (VUA), Measuring and modeling negative emotions for virtual training
08 Matje van de Camp (TiU), A Link to the Past: Constructing Historical Social Networks from Unstructured Data
09 Archana Nottamkandath (VUA), Trusting Crowdsourced Information on Cultural Artefacts
10 George Karafotias (VUA), Parameter Control for Evolutionary Algorithms
11 Anne Schuth (UvA), Search Engines that Learn from Their Users
12 Max Knobbout (UU), Logics for Modelling and Verifying Normative Multi-Agent Systems
13 Nana Baah Gyan (VUA), The Web, Speech Technologies and Rural Development in West Africa — An ICT4D Approach
14 Ravi Khadka (UU), Revisiting Legacy Software System Modernization
15 Steffen Michels (RUN), Hybrid Probabilistic Logics — Theoretical Aspects, Algorithms and Experiments
16 Guangliang Li (UvA), Socially Intelligent Autonomous Agents that Learn from Human Reward
17 Berend Weel (VUA), Towards Embodied Evolution of Robot Organisms
18 Albert Meroño Peñuela (VUA), Refining Statistical Data on the Web
19 Julia Efremova (TU/e), Mining Social Structures from Genealogical Data
20 Daan Odijk (UvA), Context & Semantics in News & Web Search
21 Alejandro Moreno Célleri (UT), From Traditional to Interactive Playspaces: Automatic Analysis of Player Behavior in the Interactive Tag Playground
22 Grace Lewis (VUA), Software Architecture Strategies for Cyber-Foraging Systems
23 Fei Cai (UvA), Query Auto Completion in Information Retrieval
24 Brend Wanders (UT), Repurposing and Probabilistic Integration of Data; An Iterative and data model independent approach

25 Julia Kiseleva (TU/e), Using Contextual Information to Understand Searching and Browsing Behavior
26 Dilhan Thilakarathne (VUA), In or Out of Control: Exploring Computational Models to Study the Role of Human Awareness and Control in Behavioural Choices, with Applications in Aviation and Energy Management Domains
27 Wen Li (TUD), Understanding Geo-spatial Information on Social Media
28 Mingxin Zhang (TUD), Large-scale Agent-based Social Simulation — A study on epidemic prediction and control
29 Nicolas Höning (TUD), Peak reduction in decentralised electricity systems — Markets and prices for flexible planning
30 Ruud Mattheij (TiU), The Eyes Have It
31 Mohammad Khelghati (UT), Deep web content monitoring
32 Eelco Vriezekolk (UT), Assessing Telecommunication Service Availability Risks for Crisis Organisations
33 Peter Bloem (UvA), Single Sample Statistics, exercises in learning from just one example
34 Dennis Schunselaar (TU/e), Configurable Process Trees: Elicitation, Analysis, and Enactment
35 Zhaochun Ren (UvA), Monitoring Social Media: Summarization, Classification and Recommendation
36 Daphne Karreman (UT), Beyond R2D2: The design of nonverbal interaction behavior optimized for robot-specific morphologies
37 Giovanni Sileno (UvA), Aligning Law and Action — a conceptual and computational inquiry
38 Andrea Minuto (UT), Materials that Matter — Smart Materials meet Art & Interaction Design
39 Merijn Bruijnes (UT), Believable Suspect Agents; Response and Interpersonal Style Selection for an Artificial Suspect
40 Christian Detweiler (TUD), Accounting for Values in Design
41 Thomas King (TUD), Governing Governance: A Formal Framework for Analysing Institutional Design and Enactment Governance
42 Spyros Martzoukos (UvA), Combinatorial and Compositional Aspects of Bilingual Aligned Corpora
43 Saskia Koldijk (RUN), Context-Aware Support for Stress Self-Management: From Theory to Practice
44 Thibault Sellam (UvA), Automatic Assistants for Database Exploration
45 Bram van de Laar (UT), Experiencing Brain-Computer Interface Control
46 Jorge Gallego Perez (UT), Robots to Make you Happy
47 Christina Weber (UL), Real-time foresight — Preparedness for dynamic innovation networks
48 Tanja Buttler (TUD), Collecting Lessons Learned
49 Gleb Polevoy (TUD), Participation and Interaction in Projects. A Game-Theoretic Analysis
50 Yan Wang (TiU), The Bridge of Dreams: Towards a Method for Operational Performance Alignment in IT-enabled Service Supply Chains

2017
01 Jan-Jaap Oerlemans (UL), Investigating Cybercrime
02 Sjoerd Timmer (UU), Designing and Understanding Forensic Bayesian Networks using Argumentation
03 Daniël Harold Telgen (UU), Grid Manufacturing; A Cyber-Physical Approach with Autonomous Products and Reconfigurable Manufacturing Machines
04 Mrunal Gawade (CWI), Multi-core Parallelism in a Column-store
05 Mahdieh Shadi (UvA), Collaboration Behavior
06 Damir Vandic (EUR), Intelligent Information Systems for Web Product Search
07 Roel Bertens (UU), Insight in Information: from Abstract to Anomaly
08 Rob Konijn (VUA), Detecting Interesting Differences:Data Mining in Health Insurance Data using Outlier Detection and Subgroup Discovery
09 Dong Nguyen (UT), Text as Social and Cultural Data: A Computational Perspective on Variation in Text
10 Robby van Delden (UT), (Steering) Interactive Play Behavior
11 Florian Kunneman (RUN), Modelling patterns of time and emotion in Twitter #anticipointment
12 Sander Leemans (TU/e), Robust Process Mining with Guarantees
13 Gijs Huisman (UT), Social Touch Technology — Extending the reach of social touch through haptic technology
14 Shoshannah Tekofsky (TiU), You Are Who You Play You Are: Modelling Player Traits from Video Game Behavior
15 Peter Berck (RUN), Memory-Based Text Correction
16 Aleksandr Chuklin (UvA), Understanding and Modeling Users of Modern Search Engines
17 Daniel Dimov (UL), Crowdsourced Online Dispute Resolution
18 Ridho Reinanda (UvA), Entity Associations for Search
19 Jeroen Vuurens (UT), Proximity of Terms, Texts and Semantic Vectors in Information Retrieval
20 Mohammadbashir Sedighi (TUD), Fostering Engagement in Knowledge Sharing: The Role of Perceived Benefits, Costs and Visibility
21 Jeroen Linssen (UT), Meta Matters in Interactive Storytelling and Serious Gaming (A Play on Worlds)
22 Sara Magliacane (VUA), Logics for causal inference under uncertainty
23 David Graus (UvA), Entities of Interest — Discovery in Digital Traces
24 Chang Wang (TUD), Use of Affordances for Efficient Robot Learning
25 Veruska Zamborlini (VUA), Knowledge Representation for Clinical Guidelines, with applications to Multimorbidity Analysis and Literature Search
26 Merel Jung (UT), Socially intelligent robots that understand and respond to human touch
27 Michiel Joosse (UT), Investigating Positioning and Gaze Behaviors of Social Robots: People's Preferences, Perceptions and Behaviors
28 John Klein (VUA), Architecture Practices for Complex Contexts
29 Adel Alhuraibi (TiU), From IT-Business Strategic Alignment to Performance: A Moderated Mediation Model of Social Innovation, and Enterprise Governance of IT
30 Wilma Latuny (TiU), The Power of Facial Expressions
31 Ben Ruijl (UL), Advances in computational methods for QFT calculations

32 Thaer Samar (RUN), Access to and Retrievability of Content in Web Archives
33 Brigit van Loggem (OU), Towards a Design Rationale for Software Documentation: A Model of Computer-Mediated Activity
34 Maren Scheffel (OU), The Evaluation Framework for Learning Analytics
35 Martine de Vos (VUA), Interpreting natural science spreadsheets
36 Yuanhao Guo (UL), Shape Analysis for Phenotype Characterisation from High-throughput Imaging
37 Alejandro Montes Garcia (TU/e), WiBAF: A Within Browser Adaptation Framework that Enables Control over Privacy
38 Alex Kayal (TUD), Normative Social Applications
39 Sara Ahmadi (RUN), Exploiting properties of the human auditory system and compressive sensing methods to increase noise robustness in ASR
40 Altaf Hussain Abro (VUA), Steer your Mind: Computational Exploration of Human Control in Relation to Emotions, Desires and Social Support For applications in human-aware support systems
41 Adnan Manzoor (VUA), Minding a Healthy Lifestyle: An Exploration of Mental Processes and a Smart Environment to Provide Support for a Healthy Lifestyle
42 Elena Sokolova (RUN), Causal discovery from mixed and missing data with applications on ADHD datasets
43 Maaike de Boer (RUN), Semantic Mapping in Video Retrieval
44 Garm Lucassen (UU), Understanding User Stories — Computational Linguistics in Agile Requirements Engineering
45 Bas Testerink (UU), Decentralized Runtime Norm Enforcement
46 Jan Schneider (OU), Sensor-based Learning Support
47 Jie Yang (TUD), Crowd Knowledge Creation Acceleration
48 Angel Suarez (OU), Collaborative inquiry-based learning

2018 01 Han van der Aa (VUA), Comparing and Aligning Process Representations
02 Felix Mannhardt (TU/e), Multi-perspective Process Mining
03 Steven Bosems (UT), Causal Models For Well-Being: Knowledge Modeling, Model-Driven Development of Context-Aware Applications, and Behavior Prediction
04 Jordan Janeiro (TUD), Flexible Coordination Support for Diagnosis Teams in Data-Centric Engineering Tasks
05 Hugo Huurdeman (UvA), Supporting the Complex Dynamics of the Information Seeking Process
06 Dan Ionita (UT), Model-Driven Information Security Risk Assessment of Socio-Technical Systems
07 Jieting Luo (UU), A formal account of opportunism in multi-agent systems
08 Rick Smetsers (RUN), Advances in Model Learning for Software Systems
09 Xu Xie (TUD), Data Assimilation in Discrete Event Simulations
10 Julienka Mollee (VUA), Moving forward: supporting physical activity behavior change through intelligent technology
11 Mahdi Sargolzaei (UvA), Enabling Framework for Service-oriented Collaborative Networks
12 Xixi Lu (TU/e), Using behavioral context in process mining
13 Seyed Amin Tabatabaei (VUA), Computing a Sustainable Future

14 Bart Joosten (TiU), Detecting Social Signals with Spatiotemporal Gabor Filters
15 Naser Davarzani (UM), Biomarker discovery in heart failure
16 Jaebok Kim (UT), Automatic recognition of engagement and emotion in a group of children
17 Jianpeng Zhang (TU/e), On Graph Sample Clustering
18 Henriette Nakad (UL), De Notaris en Private Rechtspraak
19 Minh Duc Pham (VUA), Emergent relational schemas for RDF
20 Manxia Liu (RUN), Time and Bayesian Networks
21 Aad Slootmaker (OU), EMERGO: a generic platform for authoring and playing scenario-based serious games
22 Eric Fernandes de Mello Araújo (VUA), Contagious: Modeling the Spread of Behaviours, Perceptions and Emotions in Social Networks
23 Kim Schouten (EUR), Semantics-driven Aspect-Based Sentiment Analysis
24 Jered Vroon (UT), Responsive Social Positioning Behaviour for Semi-Autonomous Telepresence Robots
25 Riste Gligorov (VUA), Serious Games in Audio-Visual Collections
26 Roelof Anne Jelle de Vries (UT),Theory-Based and Tailor-Made: Motivational Messages for Behavior Change Technology
27 Maikel Leemans (TU/e), Hierarchical Process Mining for Scalable Software Analysis
28 Christian Willemse (UT), Social Touch Technologies: How they feel and how they make you feel
29 Yu Gu (TiU), Emotion Recognition from Mandarin Speech
30 Wouter Beek (VUA), The "K" in "semantic web" stands for "knowledge": scaling semantics to the web

---

2019 01 Rob van Eijk (UL),Web privacy measurement in real-time bidding systems. A graph-based approach to RTB system classification
02 Emmanuelle Beauxis Aussalet (CWI, UU), Statistics and Visualizations for Assessing Class Size Uncertainty
03 Eduardo Gonzalez Lopez de Murillas (TU/e), Process Mining on Databases: Extracting Event Data from Real Life Data Sources
04 Ridho Rahmadi (RUN), Finding stable causal structures from clinical data
05 Sebastiaan van Zelst (TU/e), Process Mining with Streaming Data
06 Chris Dijkshoorn (VUA), Nichesourcing for Improving Access to Linked Cultural Heritage Datasets
07 Soude Fazeli (TUD), Recommender Systems in Social Learning Platforms
08 Frits de Nijs (TUD), Resource-constrained Multi-agent Markov Decision Processes
09 Fahimeh Alizadeh Moghaddam (UvA), Self-adaptation for energy efficiency in software systems
10 Qing Chuan Ye (EUR), Multi-objective Optimization Methods for Allocation and Prediction
11 Yue Zhao (TUD), Learning Analytics Technology to Understand Learner Behavioral Engagement in MOOCs
12 Jacqueline Heinerman (VUA), Better Together
13 Guanliang Chen (TUD), MOOC Analytics: Learner Modeling and Content Generation
14 Daniel Davis (TUD), Large-Scale Learning Analytics: Modeling Learner Behavior & Improving Learning Outcomes in Massive Open Online Courses

15 Erwin Walraven (TUD), Planning under Uncertainty in Constrained and Partially Observable Environments
16 Guangming Li (TU/e), Process Mining based on Object-Centric Behavioral Constraint (OCBC) Models
17 Ali Hurriyetoglu (RUN),Extracting actionable information from microtexts
18 Gerard Wagenaar (UU), Artefacts in Agile Team Communication
19 Vincent Koeman (TUD), Tools for Developing Cognitive Agents
20 Chide Groenouwe (UU), Fostering technically augmented human collective intelligence
21 Cong Liu (TU/e), Software Data Analytics: Architectural Model Discovery and Design Pattern Detection
22 Martin van den Berg (VUA),Improving IT Decisions with Enterprise Architecture
23 Qin Liu (TUD), Intelligent Control Systems: Learning, Interpreting, Verification
24 Anca Dumitrache (VUA), Truth in Disagreement — Crowdsourcing Labeled Data for Natural Language Processing
25 Emiel van Miltenburg (VUA), Pragmatic factors in (automatic) image description
26 Prince Singh (UT), An Integration Platform for Synchromodal Transport
27 Alessandra Antonaci (OU), The Gamification Design Process applied to (Massive) Open Online Courses
28 Esther Kuindersma (UL), Cleared for take-off: Game-based learning to prepare airline pilots for critical situations
29 Daniel Formolo (VUA), Using virtual agents for simulation and training of social skills in safety-critical circumstances
30 Vahid Yazdanpanah (UT), Multiagent Industrial Symbiosis Systems
31 Milan Jelisavcic (VUA), Alive and Kicking: Baby Steps in Robotics
32 Chiara Sironi (UM), Monte-Carlo Tree Search for Artificial General Intelligence in Games
33 Anil Yaman (TU/e), Evolution of Biologically Inspired Learning in Artificial Neural Networks
34 Negar Ahmadi (TU/e), EEG Microstate and Functional Brain Network Features for Classification of Epilepsy and PNES
35 Lisa Facey-Shaw (OU), Gamification with digital badges in learning programming
36 Kevin Ackermans (OU), Designing Video-Enhanced Rubrics to Master Complex Skills
37 Jian Fang (TUD), Database Acceleration on FPGAs
38 Akos Kadar (OU), Learning visually grounded and multilingual representations

2020 01 Armon Toubman (UL), Calculated Moves: Generating Air Combat Behaviour
02 Marcos de Paula Bueno (UL), Unraveling Temporal Processes using Probabilistic Graphical Models
03 Mostafa Deghani (UvA), Learning with Imperfect Supervision for Language Understanding
04 Maarten van Gompel (RUN), Context as Linguistic Bridges
05 Yulong Pei (TU/e), On local and global structure mining
06 Preethu Rose Anish (UT), Stimulation Architectural Thinking during Requirements Elicitation — An Approach and Tool Support

07 Wim van der Vegt (OU), Towards a software architecture for reusable game components
08 Ali Mirsoleimani (UL),Structured Parallel Programming for Monte Carlo Tree Search
09 Myriam Traub (UU), Measuring Tool Bias and Improving Data Quality for Digital Humanities Research
10 Alifah Syamsiyah (TU/e), In-database Preprocessing for Process Mining
11 Sepideh Mesbah (TUD), Semantic-Enhanced Training Data AugmentationMethods for Long-Tail Entity Recognition Models
12 Ward van Breda (VUA), Predictive Modeling in E-Mental Health: Exploring Applicability in Personalised Depression Treatment
13 Marco Virgolin (CWI), Design and Application of Gene-pool Optimal Mixing Evolutionary Algorithms for Genetic Programming
14 Mark Raasveldt (CWI/UL), Integrating Analytics with Relational Databases
15 Konstantinos Georgiadis (OU), Smart CAT: Machine Learning for Configurable Assessments in Serious Games
16 Ilona Wilmont (RUN), Cognitive Aspects of Conceptual Modelling
17 Daniele Di Mitri (OU), The Multimodal Tutor: Adaptive Feedback from Multimodal Experiences
18 Georgios Methenitis (TUD), Agent Interactions & Mechanisms in Markets with Uncertainties: Electricity Markets in Renewable Energy Systems
19 Guido van Capelleveen (UT), Industrial Symbiosis Recommender Systems
20 Albert Hankel (VUA), Embedding Green ICT Maturity in Organisations
21 Karine da Silva Miras de Araujo (VUA), Where is the robot?: Life as it could be
22 Maryam Masoud Khamis (RUN), Understanding complex systems implementation through a modeling approach: the case of e-government in Zanzibar
23 Rianne Conijn (UT), The Keys to Writing: A writing analytics approach to studying writing processes using keystroke logging
24 Lenin da Nóbrega Medeiros (VUA/RUN), How are you feeling, human? Towards emotionally supportive chatbots
25 Xin Du (TU/e), The Uncertainty in Exceptional Model Mining
26 Krzysztof Leszek Sadowski (UU), GAMBIT: Genetic Algorithm for Model-Based mixed-Integer opTimization
27 Ekaterina Muravyeva (TUD), Personal data and informed consent in an educational context
28 Bibeg Limbu (TUD), Multimodal interaction for deliberate practice: Training complex skills with augmented reality
29 Ioan Gabriel Bucur (RUN), Being Bayesian about Causal Inference
30 Bob Zadok Blok (UL), Creatief, Creatiever, Creatiefst
31 Gongjin Lan (VUA), Learning better — From Baby to Better
32 Jason Rhuggenaath (TU/e), Revenue management in online markets: pricing and online advertising
33 Rick Gilsing (TU/e), Supporting service-dominant business model evaluation in the context of business model innovation
34 Anna Bon (UM), Intervention or Collaboration? Redesigning Information and Communication Technologies for Development

	35	Siamak Farshidi (UU), Multi-Criteria Decision-Making in Software Production
2021	01	Francisco Xavier Dos Santos Fonseca (TUD),Location-based Games for Social Interaction in Public Space
	02	Rijk Mercuur (TUD), Simulating Human Routines: Integrating Social Practice Theory in Agent-Based Models
	03	Seyyed Hadi Hashemi (UvA), Modeling Users Interacting with Smart Devices
	04	Ioana Jivet (OU), The Dashboard That Loved Me: Designing adaptive learning analytics for self-regulated learning
	05	Davide Dell'Anna (UU), Data-Driven Supervision of Autonomous Systems
	06	Daniel Davison (UT), "Hey robot, what do you think?" How children learn with a social robot
	07	Armel Lefebvre (UU), Research data management for open science
	08	Nardie Fanchamps (OU), The Influence of Sense-Reason-Act Programming on Computational Thinking
	09	Cristina Zaga (UT), The Design of Robothings. Non-Anthropomorphic and Non-Verbal Robots to Promote Children's Collaboration Through Play
	10	Quinten Meertens (UvA), Misclassification Bias in Statistical Learning
	11	Anne van Rossum (UL), Nonparametric Bayesian Methods in Robotic Vision
	12	Lei Pi (UL), External Knowledge Absorption in Chinese SMEs
	13	Bob R. Schadenberg (UT), Robots for Autistic Children: Understanding and Facilitating Predictability for Engagement in Learning
	14	Negin Samaeemofrad (UL), Business Incubators: The Impact of Their Support
	15	Onat Ege Adali (TU/e), Transformation of Value Propositions into Resource Re-Configurations through the Business Services Paradigm
	16	Esam A. H. Ghaleb (UM), Bimodal emotion recognition from audio-visual cues
	17	Dario Dotti (UM), Human Behavior Understanding from motion and bodily cues using deep neural networks
	18	Remi Wieten (UU), Bridging the Gap Between Informal Sense-Making Tools and Formal Systems — Facilitating the Construction of Bayesian Networks and Argumentation Frameworks
	19	Roberto Verdecchia (VUA), Architectural Technical Debt: Identification and Management
	20	Masoud Mansoury (TU/e), Understanding and Mitigating Multi-Sided Exposure Bias in Recommender Systems
	21	Pedro Thiago Timbó Holanda (CWI), Progressive Indexes
	22	Sihang Qiu (TUD), Conversational Crowdsourcing
	23	Hugo Manuel Proença (UL), Robust rules for prediction and description
	24	Kaijie Zhu (TU/e), On Efficient Temporal Subgraph Query Processing
	25	Eoin Martino Grua (VUA), The Future of E-Health is Mobile: Combining AI and Self-Adaptation to Create Adaptive E-Health Mobile Applications
	26	Benno Kruit (CWI/VUA), Reading the Grid: Extending Knowledge Bases from Human-readable Tables
	27	Jelte van Waterschoot (UT), Personalized and Personal Conversations: Designing Agents Who Want to Connect With You

	28	Christoph Selig (UL), Understanding the Heterogeneity of Corporate Entrepreneurship Programs
2022	01	Judith van Stegeren (UT), Flavor text generation for role-playing video games
	02	Paulo da Costa (TU/e), Data-driven Prognostics and Logistics Optimisation: A Deep Learning Journey
	03	Ali el Hassouni (VUA), A Model A Day Keeps The Doctor Away: Reinforcement Learning For Personalized Healthcare
	04	Ünal Aksu (UU), A Cross-Organizational Process Mining Framework
	05	Shiwei Liu (TU/e), Sparse Neural Network Training with In-Time Over-Parameterization
	06	Reza Refaei Afshar (TU/e), Machine Learning for Ad Publishers in Real Time Bidding
	07	Sambit Praharaj (OU), Measuring the Unmeasurable? Towards Automatic Co-located Collaboration Analytics
	08	Maikel L. van Eck (TU/e), Process Mining for Smart Product Design
	09	Oana Andreea Inel (VUA), Understanding Events: A Diversity-driven Human-Machine Approach
	10	Felipe Moraes Gomes (TUD), Examining the Effectiveness of Collaborative Search Engines
	11	Mirjam de Haas (UT), Staying engaged in child-robot interaction, a quantitative approach to studying preschoolers' engagement with robots and tasks during second-language tutoring
	12	Guanyi Chen (UU), Computational Generation of Chinese Noun Phrases
	13	Xander Wilcke (VUA), Machine Learning on Multimodal Knowledge Graphs: Opportunities, Challenges, and Methods for Learning on Real-World Heterogeneous and Spatially-Oriented Knowledge
	14	Michiel Overeem (UU), Evolution of Low-Code Platforms
	15	Jelmer Jan Koorn (UU), Work in Process: Unearthing Meaning using Process Mining
	16	Pieter Gijsbers (TU/e), Systems for AutoML Research
	17	Laura van der Lubbe (VUA), Empowering vulnerable people with serious games and gamification
	18	Paris Mavromoustakos Blom (TiU), Player Affect Modelling and Video Game Personalisation
	19	Bilge Yigit Ozkan (UU), Cybersecurity Maturity Assessment and Standardisation
	20	Fakhra Jabeen (VUA), Dark Side of the Digital Media — Computational Analysis of Negative Human Behaviors on Social Media
	21	Seethu Mariyam Christopher (UM), Intelligent Toys for Physical and Cognitive Assessments
	22	Alexandra Sierra Rativa (TiU), Virtual Character Design and its potential to foster Empathy, Immersion, and Collaboration Skills in Video Games and Virtual Reality Simulations
	23	Ilir Kola (TUD), Enabling Social Situation Awareness in Support Agents
	24	Samaneh Heidari (UU), Agents with Social Norms and Values — A framework for agent based social simulations with social norms and personal values
	25	Anna L.D. Latour (UL), Optimal decision-making under constraints and uncertainty
	26	Anne Dirkson (UL), Knowledge Discovery from Patient Forums: Gaining novel medical insights from patient experiences

27 Christos Athanasiadis (UM), Emotion-aware cross-modal domain adaptation in video sequences
28 Onuralp Ulusoy (UU), Privacy in Collaborative Systems
29 Jan Kolkmeier (UT), From Head Transform to Mind Transplant: Social Interactions in Mixed Reality
30 Dean De Leo (CWI), Analysis of Dynamic Graphs on Sparse Arrays
31 Konstantinos Traganos (TU/e), Tackling Complexity in Smart Manufacturing with Advanced Manufacturing Process Management
32 Cezara Pastrav (UU), Social simulation for socio-ecological systems
33 Brinn Hekkelman (CWI/TUD), Fair Mechanisms for Smart Grid Congestion Management
34 Nimat Ullah (VUA), Mind Your Behaviour: Computational Modelling of Emotion & Desire Regulation for Behaviour Change
35 Mike E.U. Ligthart (VUA), Shaping the Child-Robot Relationship: Interaction Design Patterns for a Sustainable Interaction

---

2023 01 Bojan Simoski (VUA), Untangling the Puzzle of Digital Health Interventions
02 Mariana Rachel Dias da Silva (TiU), Grounded or in flight? What our bodies can tell us about the whereabouts of our thoughts
03 Shabnam Najafian (TUD), User Modeling for Privacy-preserving Explanations in Group Recommendations
04 Gineke Wiggers (UL), The Relevance of Impact: bibliometric-enhanced legal information retrieval
05 Anton Bouter (CWI), Optimal Mixing Evolutionary Algorithms for Large-Scale Real-Valued Optimization, Including Real-World Medical Applications
06 António Pereira Barata (UL), Reliable and Fair Machine Learning for Risk Assessment
07 Tianjin Huang (TU/e), The Roles of Adversarial Examples on Trustworthiness of Deep Learning
08 Lu Yin (TU/e), Knowledge Elicitation using Psychometric Learning
09 Xu Wang (VUA), Scientific Dataset Recommendation with Semantic Techniques
10 Dennis J.N.J. Soemers (UM), Learning State-Action Features for General Game Playing
11 Fawad Taj (VUA), Towards Motivating Machines: Computational Modeling of the Mechanism of Actions for Effective Digital Health Behavior Change Applications
12 Tessel Bogaard (VUA), Using Metadata to Understand Search Behavior in Digital Libraries
13 Injy Sarhan (UU), Open Information Extraction for Knowledge Representation
14 Selma Čaušević (TUD), Energy resilience through self-organization
15 Alvaro Henrique Chaim Correia (TU/e), Insights on Learning Tractable Probabilistic Graphical Models
16 Peter Blomsma (TiU), Building Embodied Conversational Agents: Observations on human nonverbal behaviour as a resource for the development of artificial characters
17 Meike Nauta (UT), Explainable AI and Interpretable Computer Vision – From Oversight to Insight
18 Gustavo Penha (TUD), Designing and Diagnosing Models for Conversational Search and Recommendation

19 George Aalbers (TiU), Digital Traces of the Mind: Using Smartphones to Capture Signals of Well-Being in Individuals
20 Arkadiy Dushatskiy (TUD), Expensive Optimization with Model-Based Evolutionary Algorithms applied to Medical Image Segmentation using Deep Learning
21 Gerrit Jan de Bruin (UL), Network Analysis Methods for Smart Inspection in the Transport Domain
22 Alireza Shojaifar (UU), Volitional Cybersecurity
23 Theo Theunissen (UU), Documentation in Continuous Software Development
24 Agathe Balayn (TUD), Practices Towards Hazardous Failure Diagnosis in Machine Learning
25 Jurian Baas (UU), Entity Resolution on Historical Knowledge Graphs
26 Loek Tonnaer (TU/e), Linearly Symmetry-Based Disentangled Representations and their Out-of-Distribution Behaviour
27 Ghada Sokar (TU/e), Learning Continually Under Changing Data Distributions
28 Floris den Hengst (VUA), Learning to Behave: Reinforcement Learning in Human Contexts
29 Tim Draws (TUD), Understanding Viewpoint Biases in Web Search Results

2024 01 Daphne Miedema (TU/e), On Learning SQL: Disentangling concepts in data systems education
02 Emile van Krieken (VUA), Optimisation in Neurosymbolic Learning Systems
03 Feri Wijayanto (RUN), Automated Model Selection for Rasch and Mediation Analysis
04 Mike Huisman (UL), Understanding Deep Meta-Learning
05 Yiyong Gou (UM), Aerial Robotic Operations: Multi-environment Cooperative Inspection & Construction Crack Autonomous Repair
06 Azqa Nadeem (TUD), Understanding Adversary Behavior via XAI: Leveraging Sequence Clustering to Extract Threat Intelligence
07 Parisa Shayan (TiU), Modeling User Behavior in Learning Management Systems
08 Xin Zhou (UvA), From Empowering to Motivating: Enhancing Policy Enforcement through Process Design and Incentive Implementation
09 Giso Dal (UT), Probabilistic Inference Using Partitioned Bayesian Networks
10 Cristina-Iulia Bucur (VUA), Linkflows: Towards Genuine Semantic Publishing in Science
11 withdrawn
12 Peide Zhu (TUD), Towards Robust Automatic Question Generation For Learning
13 Enrico Liscio (TUD), Context-Specific Value Inference via Hybrid Intelligence
14 Larissa Capobianco Shimomura (TU/e), On Graph Generating Dependencies and their Applications in Data Profiling
15 Ting Liu (VUA), A Gut Feeling: Biomedical Knowledge Graphs for Interrelating the Gut Microbiome and Mental Health
16 Arthur Barbosa Câmara (TUD), Designing Search-as-Learning Systems
17 Razieh Alidoosti (VUA), Ethics-aware Software Architecture Design
18 Laurens Stoop (UU), Data Driven Understanding of Energy-Meteorological Variability and its Impact on Energy System Operations
19 Azadeh Mozafari Mehr (TU/e), Multi-perspective Conformance Checking: Identifying and Understanding Patterns of Anomalous Behavior

20 Ritsart Anne Plantenga (UL), Omgang met Regels
21 Federica Vinella (UU), Crowdsourcing User-Centered Teams
22 Zeynep Ozturk Yurt (TU/e), Beyond Routine: Extending BPM for Knowledge-Intensive Processes with Controllable Dynamic Contexts
23 Jie Luo (VUA), Lamarck's Revenge: Inheritance of Learned Traits Improves Robot Evolution
24 Nirmal Roy (TUD), Exploring the effects of interactive interfaces on user search behaviour
25 Alisa Rieger (TUD), Striving for Responsible Opinion Formation in Web Search on Debated Topics
26 Tim Gubner (CWI), Adaptively Generating Heterogeneous Execution Strategies using the VOILA Framework
27 Lincen Yang (UL), Information-theoretic Partition-based Models for Interpretable Machine Learning
28 Leon Helwerda (UL), Grip on Software: Understanding development progress of SCRUM sprints and backlogs