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.

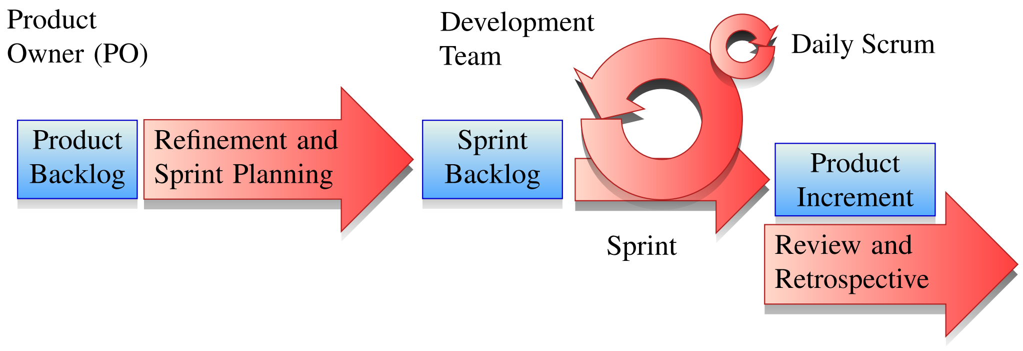

Figure 1.1: Workflow of a sprint in the SCRUM framework, showing events (red arrows), artifacts (blue) and the broad scope of some of the roles.

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/`

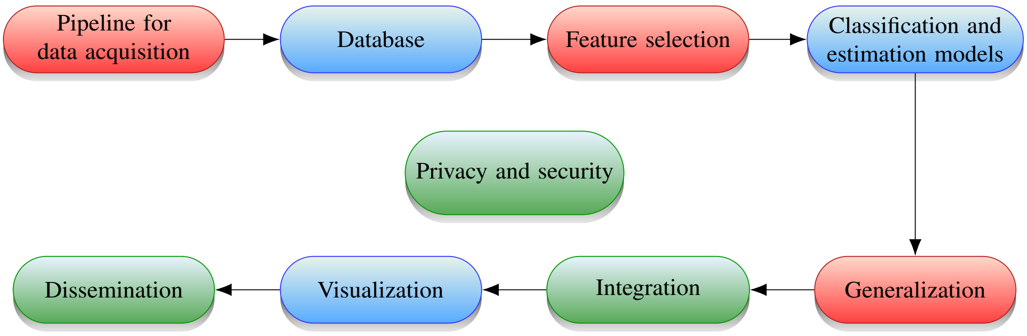

Figure 1.2: Aims and requirements as steps that are part of the design scope for the Grip on Software research, showing data mutation processes (red), technical artifacts (blue), external considerations (green) and some dependency relations.

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.

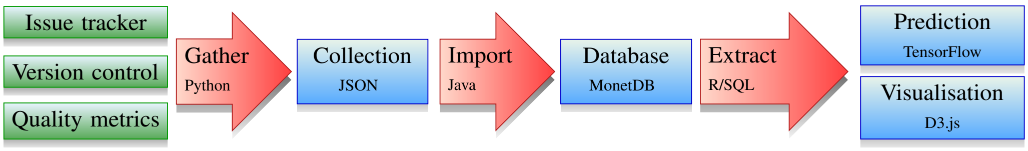

Figure 1.3: High-level overview of the full Grip on Software pipeline, with the primary SCRUM data sources (green), active components (red) and resulting artifacts (blue) highlighted.

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		PERFORMANCE	DOC	INTEGRITY	PRIVACY
		Coverage				
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.