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.

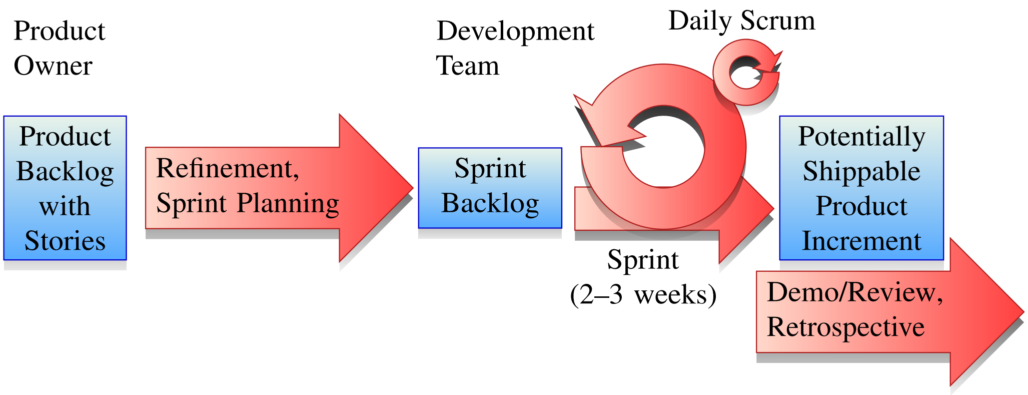

Product Owner
Development Team
Daily Scrum
Product Backlog with Stories
Refinement, Sprint Planning
Sprint Backlog
Potentially Shippable Product Increment
Sprint (2–3 weeks)
Demo/Review, Retrospective

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.

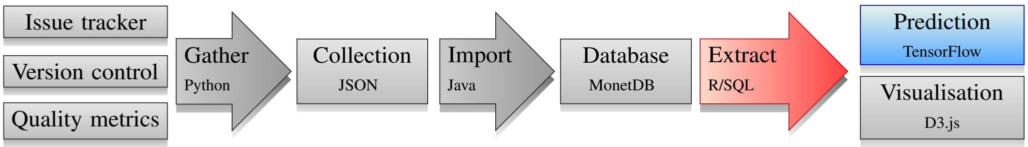


Figure 4.2: Overview of the Grip on Software pipeline, with the data analysis components highlighted.

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.

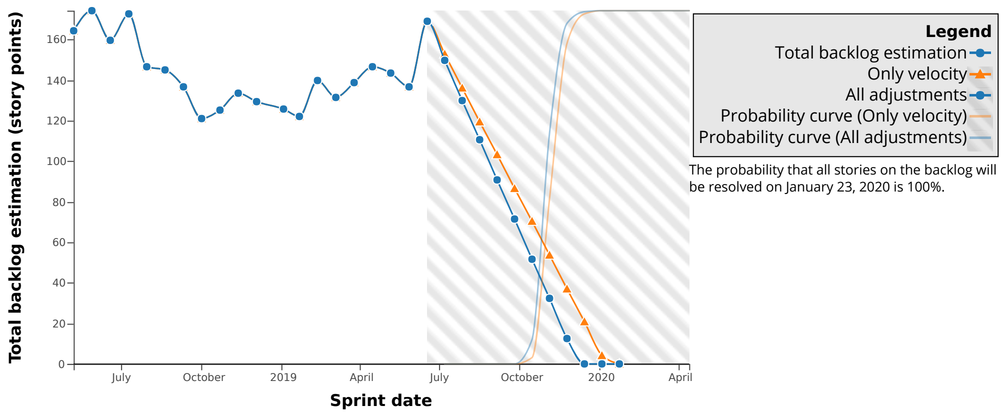


Figure 4.3: Graph of a project's backlog including estimated sizes using the linear regression model and multiple scenarios, where only changes in completed story points after each sprint (continued blue line with circles) or also other additions and removals to the backlog (orange line with triangles)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

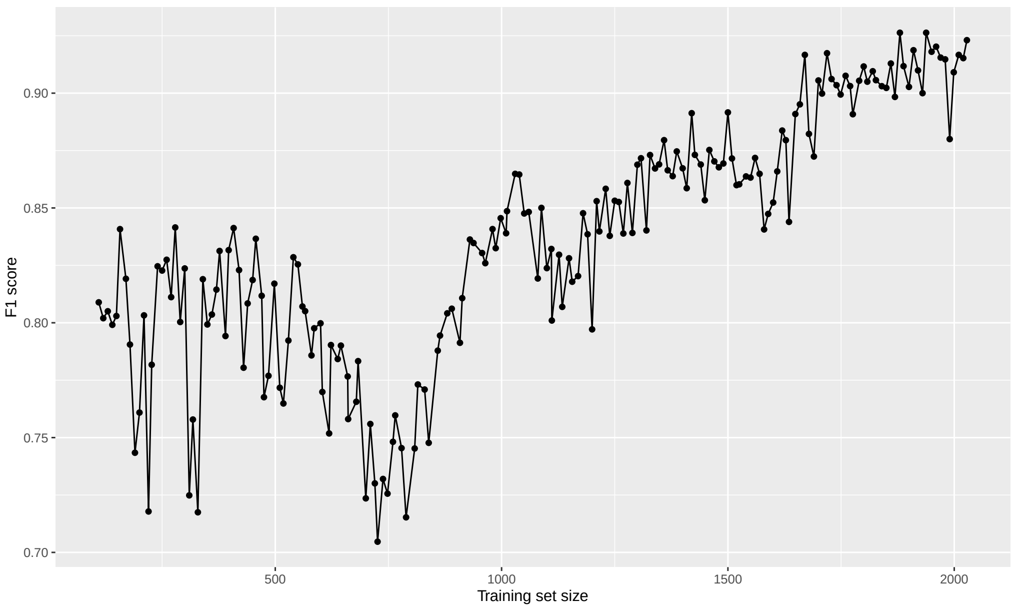


Figure 4.4: F1 score of the neural network when trained with different training set sizes.

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 (± 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.

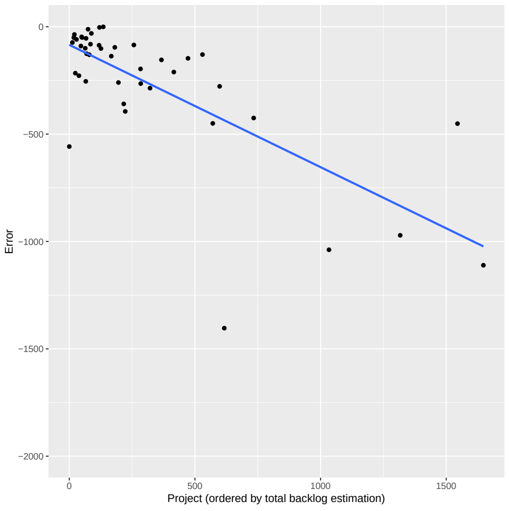
Simulation	Scenario	One third	Two thirds
Linear regression (Lin)	Velocity	-261.4 ± 316.5	-279.2 ± 355.1
Linear regression (Lin)	Mutation	-274.1 ± 326.8	-285.3 ± 359.1
Monte Carlo (MC)	Velocity	-65.0 ± 176.0	-109.1 ± 258.2
Monte Carlo (MC)	Mutation	2.7 ± 172.3	-23.2 ± 211.1

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.

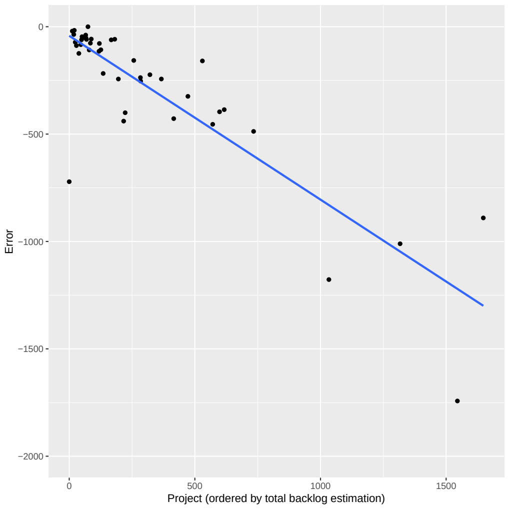
The validation error results for the backlog size predictions of individual teams and projects in each simulation can be found in Figures 4.5 and 4.6. In the plots of these figures, the projects and teams which had enough sprints with backlog data (35 total) are points which are ordered on the x axis 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

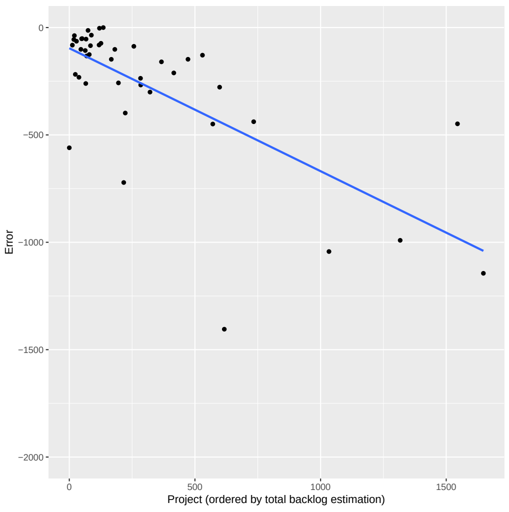
Monte Carlo, the errors show less deviation than the linear regression scenarios, but it is still biased, especially for projects with many stories on the backlog. The mutation scenario has the fewest outliers when using one third of the backlogs for seeding the distribution, but only in the case of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



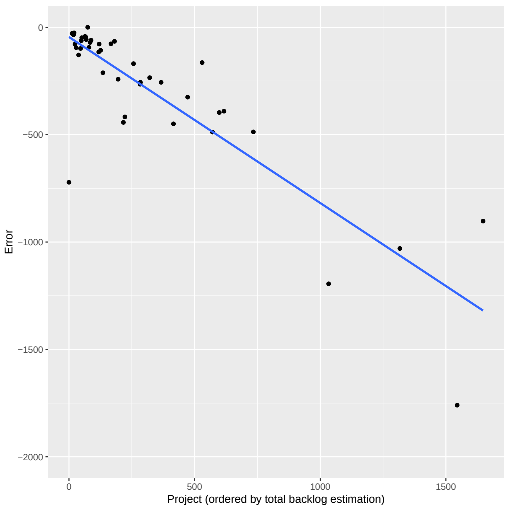
(a) Linear velocity scenario, one third ($r^2 = 0.49$)



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

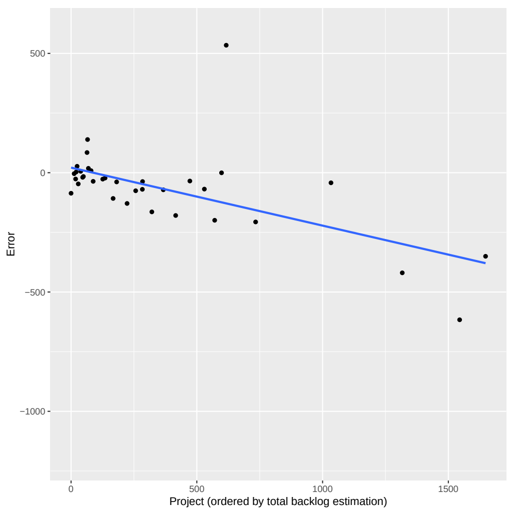


(c) Linear mutation scenario, one third ($r^2 = 0.47$)

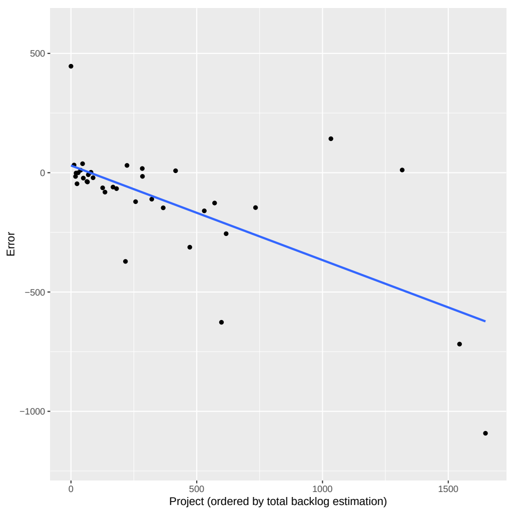


(d) Linear mutation scenario, two thirds ($r^2 = 0.79$)

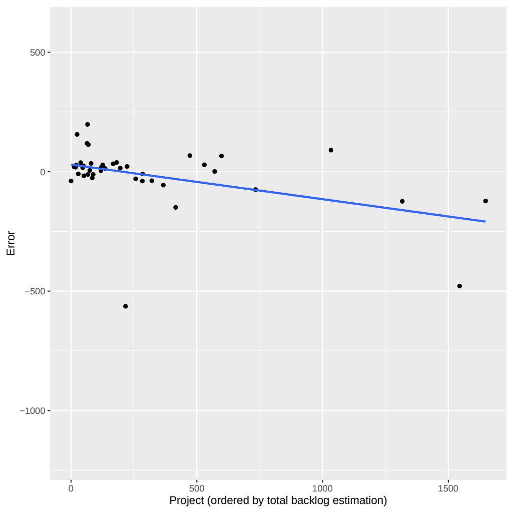
Figure 4.5: Differences in individual project/team backlog size estimations using linear regression, along with a linear trend of the validation error along the estimated most recent sizes of each backlog of the projects and teams (35 total). The reported coefficient of determination (lower is better)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



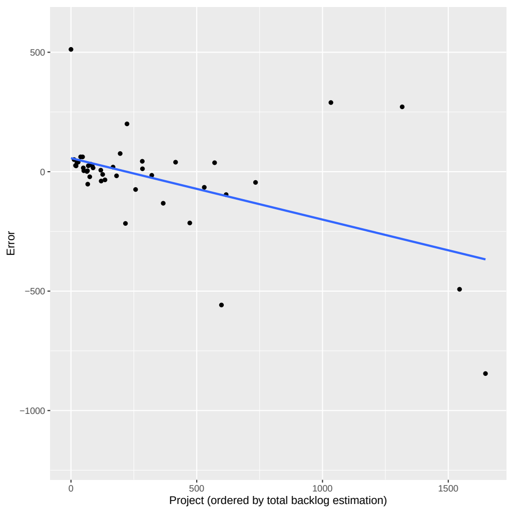
(a) MC velocity scenario, one third ($r^2 = 0.36$)



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



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

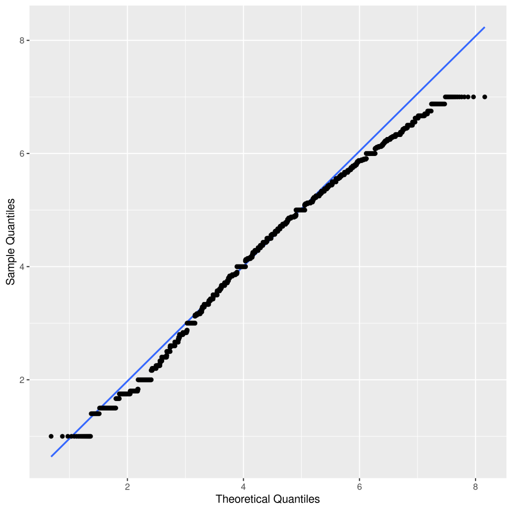


(d) MC mutation scenario, two thirds ($r^2 = 0.29$)

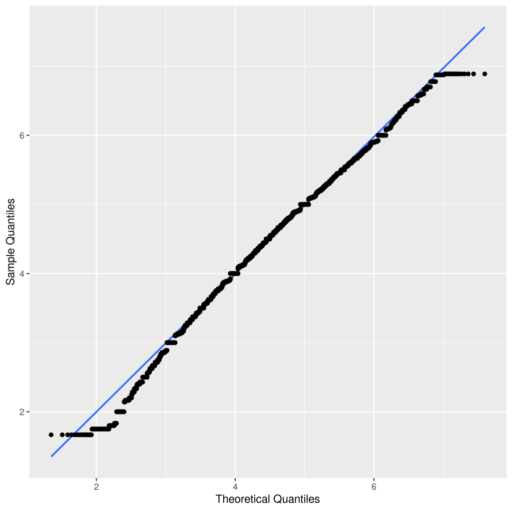
Figure 4.6: Differences in individual project/team backlog size estimations using Monte Carlo simulation, along with a linear trend of the validation error along the estimated most recent sizes of each backlog of the projects and teams (35 total). The reported coefficient of determination (lower is better)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

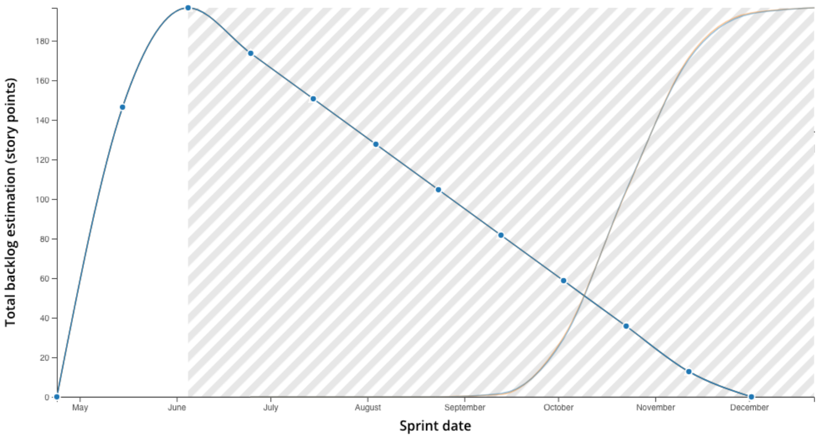


Figure 4.8: Estimation for a subset of a backlog related to a specific milestone version, showing the known and predicted backlog size (with hatch-filled background) at each sprint point in blue and the probability density curve in gray.