Conceptual Process Models and Quantitative Analysis of Classification Problems in Scrum Software Development Practices

Leon Helwerda[1 2], Frank Niessink[2] and Fons J. Verbeek[1]
[1]*Leiden Institute of Advanced Computer Science, Leiden University, the Netherlands*
[2]*Stichting ICTU, The Hague, the Netherlands*
l.s.helwerda@liacs.leidenuniv.nl

Keywords: Agile, Classification, Conceptual Frameworks, Prediction, Scrum, Software Development.

Abstract: We propose a novel classification method that integrates into existing agile software development practices by collecting data records generated by software and tools used in the development process. We extract features from the collected data and create visualizations that provide insights, and feed the data into a prediction framework consisting of a deep neural network. The features and results are validated against conceptual frameworks that model the development methodologies as similar processes in other contexts. Initial results show that the visualization and prediction techniques provide promising outcomes that may help development teams and management gain better understanding of past events and future risks.

1 INTRODUCTION

Software development organizations have to take many factors into account in order to stay dynamic and innovative. The people who work on producing a deliverable product to an actively participating client must have a diverse set of skills and knowledge about their development platform and associated topics, in order to collaborate with their peers and stakeholders.

We study the effectiveness of different practices within software development processes. Specifically, we investigate the use of the Scrum software development method, and observe the effects of various events and actions during the development process upon the outcome of the process as well as the successful release of the product. Moreover, we take other development aids, such as software quality assessment tools and continuous integration pipelines, into account in this research.

The research takes place from multiple viewpoints: we apply the principles from theoretical software engineering, delve into the practical aspects by following the actions made during a sprint, combine our experiences with relevant work and conceptual models from other fields, and apply machine learning on features that are extracted according to the models and definitions we have formed.

We specifically focus on the practice of the Scrum software development process as it is applied at a government-owned, non-profit organization based in the Netherlands. This organization develops and maintains specialized software for other governmental entities, and keeps close liaison contact with these offices. In this paper, we set out to investigate how Scrum manifests itself in this organization, what other social and technical practices are involved, and how these may be used as indicators that point toward the success of the process and the end result, as detailed in in the research questions in Section 3.2.

The remainder of our paper has the following structure: Section 2 presets our theoretical groundwork as well as points toward related practical studies. Section 3 provides insight into the problem statement and theoretical backgrounds, and Section 4 shows the analytical approach of finding solutions to some of the problems. Section 5 discusses the solutions and Section 6 concludes our findings thus far.

2 BACKGROUND

In this section, we introduce the foundations of the Scrum framework which provides us with a model of the interactions between the people, the code and the support tools. This helps us understand what certain properties in the collected data mean and how we can apply them in other models, such as the conceptual frameworks in Section 3.3. We show existing work which is relevant to this approach in Section 2.2.

2.1 Concepts

Scrum is a lightweight framework which describes a software development process. A self-organizing software development team works in sprint iterations of about two weeks to deliver increments of working software to the client. The client provides feedback on new features during a post-sprint review, and prioritizes desired items on a product backlog. The Scrum team commits itself as a whole to develop a certain number of the top items during the sprint, and in an optimal situation no stories are added or removed while the sprint is undergoing.

The Scrum process is meant to have a flexible implementation, such as what determines a story to be ‘done’. This definition can range from implementation to (automated) testing, documentation and client acceptance. Rules can be added and removed within the framework when the team agrees to do so during a retrospective, where team members discuss prior events and determine what practical problems they need to overcome in the future.

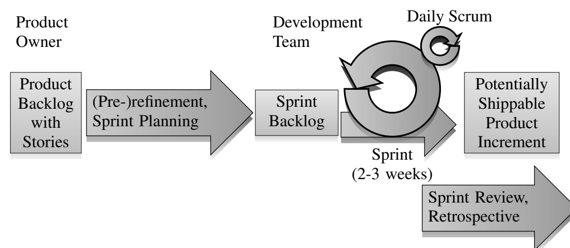


Figure 1: Workflow of a sprint in the Scrum framework.

Other events surrounding a Scrum sprint, outlined in Figure 1, are the pre-refinement, where stories are developed to become ready for selection in a sprint, the refinement where the stories are picked, and the pre-sprint planning, where the stories are outlined once more. Every workday, the team holds a Daily Scrum stand-up meeting to discuss the situation of the stories and ask each other what they did thus far, their plans to do in the remainder of the sprint and if there are any (foreseeable) problems.

Scrum is an agile software development method, which means that it adheres to principles that are set out in the Agile Manifesto (Agile Alliance, 2001). The manifesto assigns an ordering of value between pairs of software development aspects, e.g., favoring individuals and interactions over processes and tools. Even though our research makes use of such systems to collect data points, we do so to provide the team with recommendations based on data regarding their work (Highsmith, 2002). Potential conflicts resulting from putting the principles of the Agile Manifesto in practice are resolved by ensuring that there is plenty attention for higher-valued goals (Cockburn, 2007).

With regard to the individuals and their interactions, the Scrum framework defines a number of groups and roles. The shape of the organisation is outside the scope of Scrum, as it may include managers, technical leads, coaches and support teams. The main Scrum roles are as follows:

- Client: The organization which has procured for the development of the product. The client may be the end-user or the software maintainer. The client expects the product to be delivered according to their requirements. An actively involved client provides regular feedback the development team and other stakeholders such that potentially changing wishes are known.
- Software development team: A group of people that work together on a product or component. The development team shares a work ethos which drives them to not only successfully release their product in the end, but also improve their working method and the product quality.
- Scrum master: A role that might rotate between team members, who ensures that any impediments or other problems are taken care of rapidly, and verifies that the team commits to the same goals.
- Product owner (PO): A middle-man between the team and the client, who handles the bidirectional communication surrounding a Scrum sprint. The PO assesses requirements from the client and molds them into stories, with the help of the team. Additionally, the PO organizes meetings and demonstrations between the stakeholders.

2.2 Related work

Some of the longest outstanding questions in the field of software development is whether the use of so-called methodologies yields a better product that is delivered earlier than in the absence of them, and how we can compare the different practices (Wynekoop and Russo, 1995). Each in vivo study appears to differ in its scientific rigorousness (Dybå and Dingsøyr, 2008) and the topic of interest within the study. Meta-analyses of the related topic of software fault prediction using machine learning show that bias is a strong factor in the obtained results of such classifiers (Shepperd et al., 2014).

A large number of studies deal with distributed software development projects which use agile programming or management solutions. While these may provide relevant results (Paasivaara et al., 2009),

their use in on-site collaboration teams may be limited. Studies show the (successful) application of Scrum in small teams (Rising and Janoff, 2000) and in teams that have a requirement of communicating with other teams as well as external stakeholders on a frequent and documented basis (Pikkarainen et al., 2008).

We distinguish the earlier case studies into two segments: qualitative and quantitative. The qualitative studies assess the application of Scrum or another agile development practice through means of interviews, developer experiences, and scoring systems. The empirical methods used this way still help laying down new foundations for practices and anti-patterns in Scrum (Eloranta et al., 2016) and set light on new relevant factors (Lee, 2012), providing knowledge models for others to build upon.

Recent quantitative research covers topics including agile software development processes, or more specifically Scrum practices. 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 (Almeida et al., 2011). There is an analysis of the effectiveness of Scrum and Kanban on project resources management (Lei et al., 2017), and an ethnographic case study on the correlation with overtime and customer satisfaction after introduction of Scrum in an organization (Mann and Maurer, 2005).

3 DESIGN

We formulate our goals and propose our research questions related to the quantitative validation of software development processes in this section.

3.1 Goals

Different types of goals exist in the context of an analysis of software development processes. We categorize these goals by level of detail, focus area and stakeholder interest. For the benefit of the software development organization, a corporate industrial goal would be to reduce development and maintenance costs. We study various factors that influence the required effort and sprint success, i.e., whether the estimated effort is realized in time.

Tactical goals are usually high-level, with a focus on the process itself. For example, we wish to improve the software development process by means of novel standards and best practices. A research goal is then to recommend new norms based on analysis and to verify that these norms boost the progress.

At a more detailed level, we have goals that strengthen the measurable nature of the process. The software development organization management may only have a need for a single indicator of success, but some stakeholders prefer insight into the underlying factors. In a research context, we have measurable domains (projects, teams, deliverable artifacts, and so on) and we apply specific measurements to them.

Table 1: The goal, question, metric framework for Scrum software development research.

FIELD	VALUE
Object of study	Scrum board, issue tracker, version control
Purpose	Visualization, prediction, recommendation
Quality focus	Scrum sprint progress, code quality metrics, collaboration
Point of view	Team leaders, team members, management
Environment	Scrum software development organization

We summarize the purposes and context of our goals in Table 1. From this summary, we build prediction models that reduce bias toward individual domain samples, and may be generalized, applied and inspected more broadly. We extract features from artifacts and records originating from the software development process in order to better understand it and provide recommendations for stakeholders. We provide a systematic mapping from conceptual frameworks to the data set of features.

3.2 Research questions

We wish to find out how we can significantly improve software quality of products developed at software development organizations. We consider the use of various kinds of analysis tools that accept collections of measurable events as input. These events occur during the development process; they may be based upon attributes of a Scrum event, changes in the issue tracker or code, or signals of changes in the quality of the deliverable product.

From this research question, we can deduce several subquestions which form the basis of our research. Are we able to objectively determine best practices or other quality norms by means of analysis of data logs detailing the software development process? We look for indicators that point toward a successful or unsuccessful sprint period within Scrum. We take into account the viewpoints of involved stakeholders as semi-quantitative indicators.

Through this scientific analysis of process data, we may be able to deduce new, predictive norms or recommendations for software development projects. This requires research into feature extraction and model definition and validation, to support prediction of success or failure of a current Scrum sprint period. We make use of information about earlier sprints, such that we can predict the probable outcome before the sprint in question has started.

Finally, to what respect and extent, and using which kinds of measures, can the effectiveness of novel software engineering methodologies be determined scientifically? After model validation, we will apply the prediction to ongoing projects and determine the effects of recommendations on the development process and its success. The recommendation model must integrate into the current software development practices, for example by augmenting existing systems for quality reporting, project management, logistics and human resources. Such an experimental setup requires thorough verification and comparison with projects that lack this setup.

3.3 Conceptual frameworks

We describe a Scrum sprint as models which we will use to perform model validation. We present three models that relate to the linear model of a Scrum sprint, namely a factory process, a symbiotic learning network, and a predator-prey system.

In the factory model, we start at some predetermined state with a concept for something a user may want to be able to do with the product, of which the release is the eventual outcome. This leads to a use case which can be expanded into a story. The story may undergo multiple phases in which it is further detailed in terms of design and scope, after which the story is reviewed. The review determines whether the story is ready to develop into an implemented feature. This step employs programming of source code to handle the use cases. Again, this step can be reviewed to ensure code quality and agreement within the team about how the code is supposed to function. Aside from manual inspection, a test process allows the team to check if the implementation conforms to their expectations through the use of verification models (with a technical equivalent of automated regression tests and similar benchmarks).

A special twist of the Scrum factory is that the client may be involved in the quality acceptance of the product before it is released to them. This may materialize in the form of acceptance testing in a test environment, witness testing, or a demo near the end of the sprint. This external testing process brings the story closer to production. In the end, the stories that are considered to be 'done' are released in a potentially shippable increment. Again, this is slightly different from conventional product launch strategies, since not all desired functionality may have made it into the increment, but those that did are working as expected.

There are indicative moments at each step in this process: before the entire process starts, in between the subprocesses, and at the end of the production line. These moments are shown in Figure 2 and may occur during the Scrum sprint or before or after it in the case of designing and reviewing the stories. At any moment, we may determine how many of these stories are at the current step as well as how many are waiting to be pulled into the next step after a subtask is done. Thus we have separate backlogs for stories at any point of the development phase, not just before they are pulled into a sprint.

Ideally, the factory pipeline is a one-way conveyor belt with a stable speed such that the backlogs remain small and manageable. However, one additional complication is that stories may be pulled back into an earlier state, for example when review or testing uncovers problems that require redesigning, fixes in code, or other changes in an earlier process. Similarly to the intermediate backlogs, the volume of such setbacks should be limited. The practice of adding these backward flows into the model yields a value stream map, which stems from the Lean software development principles (Abdulmalek and Rajgopal, 2007).

In another context, the Scrum sprint can be seen as a symbiotic environment that encourages stakeholders to learn from past mistakes, such that known problems can be prevented in the future.

One can define a time range, such as the start of a sprint until the end of a sprint, in which the team performs actions that may improve the product and themselves. At the start of this range, we have a number of artifacts, such as code, components in the system architecture, stories in the sprint and in the backlog, and (reported) bugs. All of these artifacts may have some measurable indication of how proficient they are: is the code readable, are the stories detailed enough (but not too implementation-specific), etc.

At the end of the sprint, these artifacts have the same properties, but upon measuring them they may have improved. We can detect if the solutions were implemented in the code in such a way that it is reusable for later features and is future-proof against unknown bugs or regressions elsewhere in the code. This includes checks for code duplications or other code smells within or between components. The structure of the architecture may improve, which is

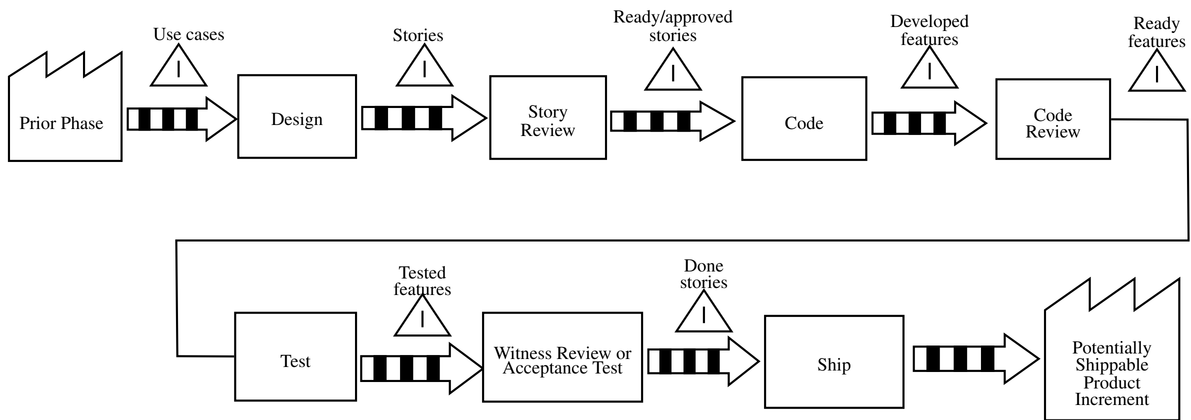


Figure 2: Factory model of the Scrum process, similar to value stream maps from Lean.

more than just aligning it with the initial design concept. Problems that were encountered with certain stories should be used as a learning moment to ensure the use case is clear enough before work commences, and to lead to fewer bugs in the future.

As a final model, the actions of software developers that complete work on a story, find and fix bugs, or create unit tests can be seen as a predator that intends to minimize the population size of a prey (Arcuri and Yao, 2008). Every time the developers get more work done within a sprint, their 'prey' should subsequently subside. However, if the quality and quantity of the actions are lower than expected, then the number of prey grows again due to the rise of bugs and undesirable features.

We define the predator size as the amount of work that the team achieves, i.e., the velocity of the team. We map the prey to volume of the product backlog that need to be actioned upon, such as stories and bugs. This makes the two population sizes more abstract than in the biological process. The main similarities are that the two volumes are inversely related to each other, and the assumption that there is enough 'food' for the prey to live from, namely the influx of ideas to improve the product and the code in the product itself that may hold – not yet known – bugs. Finally, we assume that the predator is geared toward solving these problems as the collective goal.

The powerful dynamics of predator-prey systems have been studied in depth. In general, the predator works best with a large population of prey (a definition which can additionally take into account the well-orderedness of the backlog and clarity of the stories). The predator often decreases the size of the prey to an extent that it is almost extinct. This reduces the work output of the predator, leading to a resurgence of the prey stories and bugs. There are however stable versions of the predator-prey system, where neither of the two species changes their size based on the other, or they slightly oscillate around two mean points.

What we learn from these observations is that software development processes work best when the backlog size is large enough. More importantly, the system becomes stable when each cycle does not yield tremendous changes to both volumes. Thus, a stable influx of (new) stories, as well as a stable velocity of work done per time unit, are factors in the process that help ensure that the project can continue onward. The predator-prey system obviously does not include all aspects of the development process, but it provides a mathematical concept of the major relevant properties of the Scrum cycle-based framework: input, changes, and output of story units as well as the velocity of the team itself. Responding to changes in the backlog volume and scope allows the predator team to keep the prey volume of issues and tasks at a manageable level.

4 ANALYSIS

We collect data from distributed version control systems, issue trackers and other tools used by the projects. This is a completely automated process that works via a pipeline where data flows one way. After the collection and processing steps, the data is stored in a database. The pipeline takes into account the latest state of the collection process such that only updated data is retrieved. This way, we can perform frequent analysis using the persistent database, for example feature extraction as demonstrated in Section 4.4. We do this every time a new sprint might start, e.g., weekly, and predict the outcome of new sprints as soon as possible.

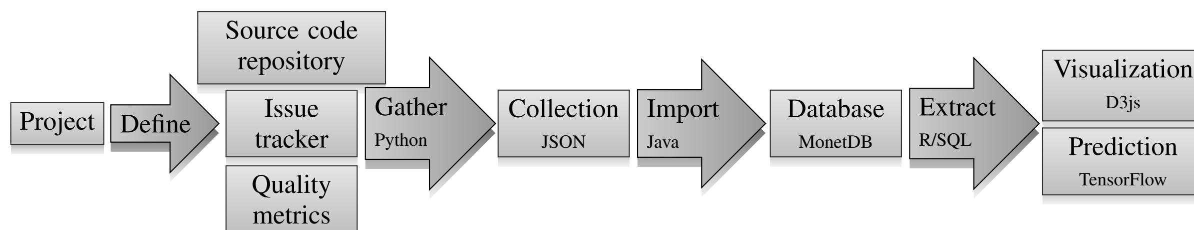


Figure 3: Pipeline of the collection of data from projects and their purposes after feature extraction.

4.1 Data sources

Each project has its own set of instances of tools used during software development, such version control systems (VCS), quality reporting tools, build automation, documentation wikis and project management systems. A project has an associated issue tracker board, which in our case is JIRA. This software provides additional functionality for Scrum boards with a backlog and sprint tracking. Projects use a VCS like Git or Subversion. In the case of Git, several repository managers with review tools are in use, in particular GitLab, GitHub and Team Foundation Server.

Quality control is achieved using SonarQube with a diverse set of profiles. The results of a SonarQube check are made available to a quality dashboard which holds current and previous values of metrics based on code quality and other sources. A metric may have details available at the source in question.

Because some of these sources are only available to the team itself for security considerations, we make use of Docker-based automated services that are deployed in the development environment of the team. These 'agents' register themselves at a central server, regularly collect fresh data and send the data to the server. Additionally, the agents perform health checks to warn if there are problems with the environment.

We process the data and where possible, automatically create relationships between data sources, such as matching a code commit with the sprint or issue it relates to. Next, user accounts in the JIRA issue tracker and the commit authors in VCS repositories are linked, with a hand-made filter when automatic matches are insufficient. Finally, the data is imported into a MonetDB database as shown in Figure 3.

4.2 Threats to validity

During our initial research, we validate the data collected so far against other sources and findings during a Scrum sprint. For example, we compare the actions taken by team members during daily stand-up meetings and find that many administrative actions in the issue tracker take place around such meetings. We also find this by comparing certain actions, such as rank changes and story point changes, with meeting reservation data from a self-service desk system.

This means that we cannot assume that the action, such as closing a story, actually took place when the task is finished or a decision is made. Detailed additions to tasks are often done during lunch breaks or near the end of the day, for some teams with up to four times as many changes in such hours compared to other moments during the sprint. This makes it more difficult to connect changes to issues with code changes made during the day, but does not immediately affect our method when aggregating data over entire sprints. Knowledge about the existence of these patterns may in fact help find other anomalies.

Sprint are administratively closed in the issue tracker as well. By default, the end date is a projection from the start date, so if nothing is done the sprint is closed automatically a few weeks later. For sprints whose stories are done in time, a date of completion is known, but it may suffer the same consequences of (delayed) administrative actions. We may use it as a middle ground in some cases, such as when sprints seem to overlap or have dubious dates.

Teams use the functionality provided by the issue tracker in different ways. Due to various definitions of 'done', inherent to the Scrum framework, an issue status may have several meanings. As another example, an impediment may indicate that the team is waiting for feedback from the client, not that the team has a problem that must be fixed by the Scrum master.

In other data sources, we may have problems with missing data, such as when a quality metric source is misconfigured. Version control systems allow team members to describe their changes in short commit messages. Quite often, developers do not make use of this, or they use a integrated development editor which fills in the latest message automatically. It is considered good practice to mention the issue that the commit relates to, but this only happens in up to 14% of all commits in our data set. About 6% of all commits are merges, which is relatively low considering that in distributed development, features are often implemented on a branch, tested and merged later on.

We intend to generalize our approach, and build a feature extraction model where we create reusable definitions of properties related to the Scrum process whose realizations take into account the unexpected patterns that exist in the data. Additionally, we take decisions about improving our coverage of certain properties across all fields, consider not using a field directly for some feature, or assume that we can interpolate or leave out a metric or event.

4.3 Reporting

We report our findings back to involved stakeholders, including team members and management, through various communication channels. We take into account that a bare number or classification for a sprint does not provide sufficient context. Many people wish to know how the report came to be and what else can be deduced from the data. For this reason, we provide as many details from the steps that we take in the feature extraction and prediction process.

Aside from the prediction results, we separately make all features available in a timeline visualization which displays and compares Scrum sprints from different teams. The timeline includes significant events that took place in each sprint. Additional visualizations of the collected data come in the form of a burn down chart, a leaderboard with project statistics, a calendar showing code commit volumes per day using a heat map and external data such as daily weather temperatures, and a network graph showing collaborations between team members on different projects with time-lapse capabilities.

We hold a system usability scale (SUS) questionnaire. The questionnaire is reachable from the visualization interface and yields 17 responses. The respondents have various roles in the organization. We found that none of the respondents disagreed with the statement that the visualizations were well integrated, and the general agreement is that the visualizations are easy to use (only the timeline has two disagreements). Most of the respondents are not yet inclined to use the visualizations frequently. Comments seem to indicate that this is due to the fact that the data shown does not directly impact their current work progress.

The classifications for a current sprint are shown on a distinct page, including a risk assessment as well as metrics that indicate the performance of the prediction algorithm and its configuration. All data is shared with other tools, including a quality reporting tool that is well-used by the teams.

Intermediate results are not only shared electronically but also presented during various meetings, which immediately provide the possibility for attendees to provide comments and questions. Similar to a Scrum review, we attempt to display an early version of a visualization such that we can update it based on feedback from these meetings.

4.4 Feature extraction

In order to create a dataset of numerical features that describe certain properties of the Scrum sprints that have taken place, we perform feature extraction on the collected record data. We use a combination of SQL statements and R programs to aggregate the data. The SQL statements may contain variables that define certain common properties, filters and formulas, such as the actual end date of a sprint, types of issues related to stories, or the calculation of the velocity in a sprint, based on the number of story points divided by the number of working days in the sprint.

This way, we define features of sprints in a generic manner, taking into account inconsistencies in the source data as mentioned in Section 4.2:

1. Sprint:
   - Sequential order of the sprint in the project lifespan.
   - Number of weekdays during the sprint.
2. Team size:
   - Number of people that made a change in the code, or on the issue tracker, during the sprint.
   - Number of sprints that each developer has made a change in before the sprint.
   - Number of new developers in the team that have not made a change before.
3. Issue tracker:
   - Mean number of watchers or people making a change on an issue.
   - Mean number of story points that are ‘done’.
   - Mean number of labels provided to an issue.
   - Number of impediments.
   - Number of changes to the order of stories on the backlog, or the number of points, before or after the sprint has started.
   - Number of stories that are not closed as ‘done’.
   - Number of workdays since the start of the sprint which is the pivot day around which the most changes are made.
   - Velocity, both for the sprint as well as the average over three sprints prior.
   - Number of issues that are closed, except stories.
   - Number of concurrent stories, and the average number of days that the stories are in progress.

4. Code version control:
   - Number of commits.
   - Average number of additions, deletions, total difference size, number of files affected.
5. Metrics:
   - The overall sentiment of the team about the sprint as indicated during the retrospective.
   - Number of metrics that are shown in the quality dashboard, and the number of metrics that are underperforming or not available.

Any of these features may take on the role of a label, indicating a single outcome of a sprint to be predicted from the remaining features. The label may be converted to binary classifications. The features are rescaled such that training models are not influenced by unrelated scales.

Because the eventual value of a feature is unknown while a sprint is in progress, we instead predict the label for this sprint using features from earlier sprints. We create such a dataset by rolling all features to the later sprint of the same project. This loses the features of the latest 'active' sprint, as well as a complete sample of the first sprint of the project.

For example, we may have 15 projects of different lifespans, with a total of 530 sprints. After the roll operation, we remove the label of first sprint of each project and stow away the latest sprints as our prediction target or validation set, leaving us with 500 sprints in the main dataset. Table 2 shows the actual dimensions and other properties of our data.

Table 2: Dimensions and related properties of the database.

PROPERTY	VALUE
Projects	15
Issues	60158
Stories	5369
Changes per issue	8.5
Code repositories	196
Code changes	140357
Metric values	71806613
Sprints	531

We then split up the dataset into training and test sets, using stratified cross-validation to avoid biased sets. We also calculate the distribution of labels across the sets and the accuracy when we take the label of previous sprint is as the new label, to better understand the data and to improve the prediction algorithm. The project identifier is never passed to the model or training algorithm to generalize its use for all teams; the label distribution may optionally be used to rebalance the training set.

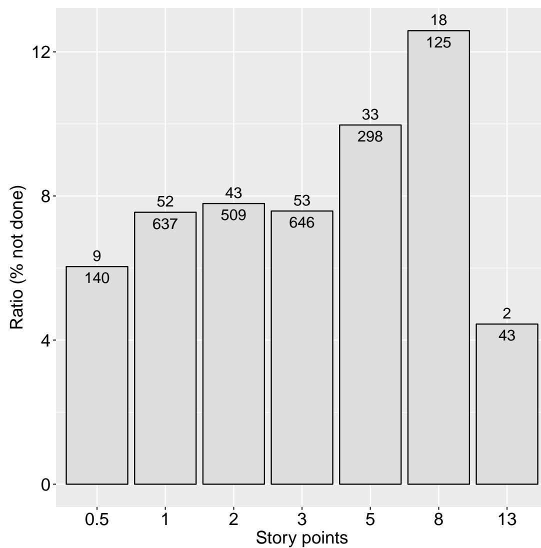
5 VALIDATION

From our thoughts on conceptual frameworks in Section 3.3, we deduce certain properties which appear to be relevant in both a Scrum process and in similar processes. One point is that there must be some added value after a period in which the most relevant actions take place. For Scrum, this means that there must be some (predetermined) number of story points reached at the end of the sprint. Certainly, when value is not realized within this period, it may need to be done in a later sprint, which is not helpful for throughput of prioritized stories. Thus, if there are stories that are not done or closed as unfixable at the end of the sprint, then this indicates a problem.

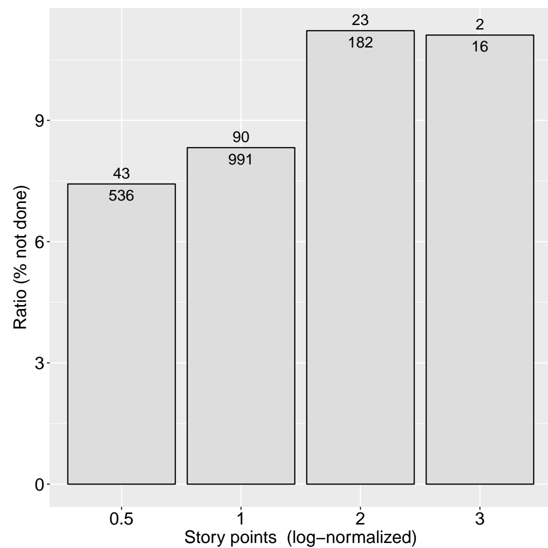
5.1 Preliminary results

During our initial research into the quality of the collected data, we create an inventory of the possible applications of the data, through discussions with developers, Scrum coaches, management, and support team members. We specifically select questions which can be answered efficiently with the database, and additionally indicate whether the results lead to unexpected results. Thus we validate the quantitative data against human expectations regarding the Scrum process. This allowed us to find some peculiarities in the data, such as the length of the sprint which is often predetermined due to a projected end date, or changes made to priorities or story points at unlikely moments.

One of these questions relates to an often-stated guideline with regard to the size of a story: If the story is considered to be large, then it is better to split it up into multiple smaller stories. We wondered whether a story which is awarded with many points during the refinement, cf. Section 2.1, is more likely to end up being 'not done' than a story with few points. Story points may not be entirely comparable across teams, or even across periods of time. Story points are awarded according to the Fibonacci scale. Therefore, we acquire a logarithmic normalization factor of the largest story of each sprint. In Figure 4(a) we aggregate stories with the same points and demonstrate the ratio of not-done stories with those points. The numbers above each bar indicate the stories that are 'not done', and the total number of stories with the same amount of points is shown in the bar. Figure 4(b) shows aggregated ratios after log-normalization. The distinct trend shows there an increased likelihood that a story with a higher number of points is not finished. This pattern remains visible when taking subsets of projects, and indicates that we are able to answer these questions efficiently.



(a) Story points and likelihood of 'not done'.



(b) Normalized story points and likelihood of 'not done'.

Figure 4: Results showing a summary of the story points and the ratio of being 'not done' over all stories with the same points.

5.2 Prediction

We identify sprints with a high risk of unsatisfactory results by training a machine learning algorithm on a dataset with 23 features and one label, which we introduce in Section 4.4. Even though many indicators of successful sprints are feasible, we use a single metric in this model for simplicity. We consider a sprint to be successful if and only if all the stories involved in the sprint are closed at the end of the sprint, with no deferrals. We convert the feature providing the number of 'not done' stories to this binary label. The class distribution is highly biased toward sprints that have no unfinished stories, making up 80% of the data set. Other features, such as the number of impediments (77%) or the number of story point changes after the sprint has started (83%) exhibit similar distributions. A weighted label combining such features at different thresholds may improve this distribution.

We apply the data set to a deep neural network with various configurations to make use of the capabilities of such architectures to handle a large number of features. We find a feasible neural network with three hidden layers of 100, 200 and 300 activation nodes, respectively. In Figure 5, we show the accuracy curve of this experiment. We reach an accuracy of 84% on the test set after training the neural network for 1000 steps. A baseline classification using the label of the previous sprint has an accuracy on the test set of 78%, indicated by the dashed line. Our trained model thus outperforms a forecasting operation. This is a promising result of our novel application of machine learning on Scrum data.

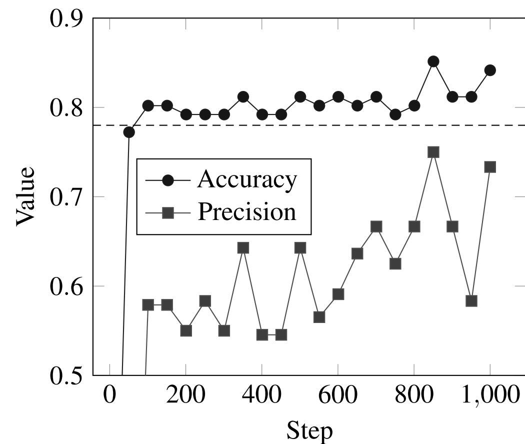


Figure 5: Accuracy and precision of the three-layer neural network on the data set, with the label 'all stories are done'.

6 DISCUSSION

Our research is in the preliminary stages of assessing the possibilities of a quantitative analysis in Scrum software development processes. Our results thus far indicate that there is a promising potential of such an analysis, which may provide us with relevant factors and risk assessments during a sprint. We wish to know which features demonstrate the impact on the progress of the sprint the most, and why they are influential. From a data mining standpoint, we show that it is possible to extract features from a collection of records regarding Scrum sprints. Further analysis will show what techniques work to improve accuracy.

In our formulation of conceptual models akin to a Scrum sprint, we find that some of these models are easier to relate than others. Further abstraction sometimes reveals the most important factors in the newly created model and thus the original Scrum model. Inspirations from other scientific fields help make the model more intelligible through recognizable properties. A model may be simplified, describe a specific feature in more detail, or contain inherent attributes that only become visible once the model is abstracted. From our proposed models, a feature of stable velocity, independent of the team size, comes to mind.

6.1 Conclusions

This quantitative study of process data from Scrum development sprints presents a novel application of data mining in the field of software engineering. The use of state-of-the-art prediction algorithms plays a huge role in this research. Analytical approaches help find the success factors of Scrum sprints.

Conceptual models present a viable method for validating a set of features and labels against the model or fractions of it. The abstractions that are made by relating one event to another help in finding features that one could not perceive beforehand.

6.2 Future research

The Scrum process provides a model which provides a set of features that indicate certain behavior during a sprint. Accurate prediction using these features remains challenging due to noisy data. Future developments, including feature selection and expansion of the data set with more variation, help solving this task.

We intend to not only predict binary classifications to our end users, but also provide recommendations to team members and management. The learning model must be able to tell why it came to a certain conclusion and what can be done to counteract the risk of a failing sprint within time constraints. This means that the model will have more introspective abilities as well as the capability to provide more than a risk assessment, leading to new norms.

ACKNOWLEDGEMENTS

We want to thank Stichting ICTU for providing the funding and data access which makes it possible to perform research and build tools for prediction and visualization. Particularly, we thank those who assist us through feedback during meetings, interviews and surveys and let us observe Scrum in practice.

REFERENCES

Abdulmalek, F. A. and Rajgopal, J. (2007). Analyzing the benefits of lean manufacturing and value stream mapping via simulation: A process sector case study. *International Journal of Production Economics*, 107(1):223–236.

Agile Alliance (2001). Manifesto for agile software development. http://agilemanifesto.org/ [2017-08-30].

Almeida, L., Albuquerque, A., and Pinheiro, P. (2011). A multi-criteria model for planning and fine-tuning distributed Scrum projects. In *Proceedings of the 6th IEEE International Conference on Global Software Engineering*, pages 75–83.

Arcuri, A. and Yao, X. (2008). A novel co-evolutionary approach to automatic software bug fixing. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 162–168.

Cockburn, A. (2007). *Agile Software Development: The Cooperative Game*. Addison-Wesley, 2nd edition.

Dybå, T. and Dingsøyr, T. (2008). Empirical studies of agile software development: A systematic review. *Information and Software Technology*, 50(9):833–859.

Eloranta, V.-P., Koskimies, K., and Mikkonen, T. (2016). Exploring ScrumBut–An empirical study of Scrum anti-patterns. *Information and Software Technology*, 74:194–203.

Highsmith, J. A. (2002). *Agile Software Development Ecosystems*. Addison-Wesley.

Lee, R. C. (2012). The success factors of running Scrum: A qualitative perspective. *Journal of Software Engineering and Applications*, 5(6):367–374.

Lei, H., Ganjeizadeh, F., Jayachandran, P. K., and Ozcan, P. (2017). A statistical analysis of the effects of Scrum and Kanban on software development projects. *Robotics and Computer-Integrated Manufacturing*, 43:59–67.

Mann, C. and Maurer, F. (2005). A case study on the impact of Scrum on overtime and customer satisfaction. In *Proceedings of the Agile Development Conference*, pages 70–79.

Paasivaara, M., Durasiewicz, S., and Lassenius, C. (2009). Using Scrum in distributed agile development: A multiple case study. In *Proceedings of the 4th IEEE International Conference on Global Software Engineering*, pages 195–204.

Pikkarainen, M., Haikara, J., Salo, O., Abrahamsson, P., and Still, J. (2008). The impact of agile practices on communication in software development. *Empirical Software Engineering*, 13(3):303–337.

Rising, L. and Janoff, N. S. (2000). The Scrum software development process for small teams. *IEEE Software*, 17(4):26–32.

Shepperd, M., Bowes, D., and Hall, T. (2014). Researcher bias: The use of machine learning in software defect prediction. *IEEE Transactions on Software Engineering*, 40(6):603–616.

Wynekoop, J. and Russo, N. (1995). Systems development methodologies: Unanswered questions. *Journal of Information Technology*, 10:65–73.