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

Helwerda, L.S.

Summary

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

Complexity is a common factor in software development processes. Software developers, quality engineers and other leading roles often face challenges during development, in particular when balancing the effort spent in various tasks surrounding the implementation and maintenance of features in software products. These experts deal with an ever-changing digital ecosystem as well as priority shifts in wishes of the user representatives, i.e., the client.

In order to remain focused on implementing requirements that are described in user stories during a development phase—not rigid plans formulated early on in the life cycle—software development teams often choose to work according to an Agile software development method such as SCRUM. We wish to improve the predictability of the short-term SCRUM sprint cycle and the long-term product backlog planning, while retaining the flexibility of the process.

The essence of SCRUM is that repeated series of meetings and events take place. This allows us to extract structured data regarding the progress from several systems that each team uses as a digital support. After data acquisition and database modeling, we describe and select relevant metrics and features for a data set, aimed at improving and evaluating machine learning algorithms.

We introduce several algorithms, firstly for predicting work that a team could commit to finishing before the end of a SCRUM sprint, which takes a few weeks. Additionally, we explore forecasting algorithms that estimate the amount of work on the backlog over a longer period of time. The results from these models demonstrate that we are able to indicate whether selected user stories—with expert effort estimations in the form of story points—will be finished during a sprint, at a reasonable accuracy. We also show that it is more difficult to determine if certain milestones in the future are reachable, e.g., due to unforeseen scope changes. Still, these algorithms and data sets allow us to highlight patterns from a software development project's life cycle. This way, we provide novel, interactive information visualizations that aid in decision making.

In order to increase the extent of the data set and thus feed our algorithms with fresh data, we construct a data acquisition pipeline for our Grip on Software (GROS) research. The GROS pipeline is designed to be generalizable, so that it is able to collect data from different systems that teams and organizations work with. At two governmental software development organizations based in the Netherlands—Stichting ICTU and Wigo4it—we apply the components of the pipeline to acquire and augment the data sets through frequent runs of distributed agents at the organization. A centralized instance receives updates as well for a combined data set. We consider privacy and security aspects by (pseudo-)anonymizing project-sensitive data and personally identifying information.

After fresh data is acquired from systems related to project tracking (Jira and Azure DevOps), code versioning (such as Git and including review systems like GitHub and Gitlab), quality control (SonarQube and Quality-time), build platforms and so on, we model the data based on similarities between systems and establish new connections across those systems, so that we are able to derive emergent metrics based on intersections of the software development process data. This model leads to the construction of a MonetDB database, where large numbers of records for entities and relations from the software development process are easily inserted, updated and retrieved in batches using column-based storage.

The actual extraction and feature selection of features for the data set takes place using a novel query template compilation system. The templates are usable for various sources of data that are similar to each other, regardless of whether the data is modeled slightly differently, thus supporting diverse development ecosystems at the organizations that are involved. Common variables are shared between queries, simplifying the task to define, e.g., effort estimation values from story points while reducing noise from real-life data.

We apply these query templates to our GROS research to construct novel features, based on input from developers, SCRUM coaches and quality engineers, to describe various metrics, team properties and events that take place during a sprint in numeric form. We split up the samples of sprints in our data set into training, test and validation sets, taking into account the temporal aspect of the data. By only using older sprints of a development project during training, we avoid problems where a team's effort in a later sprint was influenced by the result of an earlier sprint which should thus not show up in test or validation.

As part of finetuning the data set and models, the extracted features are scored to determine which ones contribute the most in estimating a target label, e.g., an indication of whether stories are done at the end of a sprint. Additionally, we apply one of our models to select a representative subset of features based on a distance measure, similar to clustering.

Due to the limitation in dimensions, we consider classification and estimation models which apply practices from deep learning, in particular architectures that are explainable to stakeholders. We use a three-layered neural network (DNN) for classification—with an F1 score of 89.98%—and analysis-based effort estimation (ABE) for the evaluation of the number of story points that the team could complete during a sprint, with the latter model providing estimations that are within a 25% margin for 88.6% of one organization's validation samples. The stability of the ABE model is however problematic, as it is unable to handle data combined from multiple organizations, whereas DNN classification exhibits better statistical measurements with larger data sets.

When it comes to forecasting backlog sizes in the longer term, we compare a linear regression with Monte Carlo simulations using various scenarios, taking more data from earlier backlog progression and other mutations into account. We find that the Monte Carlo method is able to generate a proper normal distribution of outcomes, matching the eventual end of a project most closely, with only a third of a project's life span provided to feed the simulations. Still, all models underestimate the backlog size, with a likely cause being scope changes of a development project.

The data set and prediction results form the basis for a hub of information visualizations, which provide refreshing views on patterns and situations from development processes. The interactive visualizations include customizable reports, detailed prediction dashboards, timelines, network graphs, flow charts and calendars with heat maps. These layouts are meant to showcase different aspects of the modeled processes. Based on discussions with stakeholders, we also realize controls to provide the details that they need. This service allows intuitive access to the results, with focus on the most relevant features and options to zoom into fine-grained information.

Furthermore, we integrate status metrics from prediction results into quality reports as well as create inventive backlog charts for display in project management systems, with links and references to the visualizations and the source data. Usability and accessibility tests show that the design is effective, with several stakeholders adopting the visualizations in their workflow.

Overall, these approaches allow us to provide answers to our research questions and accomplish objectives laid out in this thesis, with the central focus on understanding and improving the SCRUM software development process, in particular the predictability of delivering incremental changes during sprints and viable products at long-term milestones. Through extraction of data and analysis of features describing events, we are able to create a data set and construct models aimed at various predictive tasks. The data acquisition pipeline and pattern recognition methods are augmented with a visualization front-end, which initiates a new feedback loop involving stakeholders of the development project. Using rapid data acquisition, database storage and analysis in our integrated pipeline, we are able to expand our data set regularly, leading to new classifications, estimations and information visualizations at a daily—or more frequent—rate. Together, this paves the way to an enhanced SCRUM software development progress.