



Universiteit
Leiden

The Netherlands

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

Helwerda, L.S.

Citation

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

Version: Publisher's Version

License: [Licence agreement concerning inclusion of doctoral thesis in the Institutional Repository of the University of Leiden](#)

Downloaded from: <https://hdl.handle.net/1887/4092508>

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

Chapter 5

Information visualization

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

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

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

Abstract of Chapter 5

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

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

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

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

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

5.1 Preamble

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

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

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

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

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

RQ3a Which concepts and goals are relevant when designing information visualizations for patterns analyzed from a software development process?

RQ3b How do we integrate results from predictive models within existing development practices?

RQ3c Which effects do the introduction of results from predictive models and other intermediate analyses have on the development process, validated across multiple ongoing projects?

RQ3d What is the overall assessment of the proposed information visualizations, considering automated measurements for usability and the adoption according to interviews and surveys?

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

5.2 Purpose

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

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

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

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

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

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

5.3 Relevant concepts

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

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

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

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

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

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

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

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

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

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

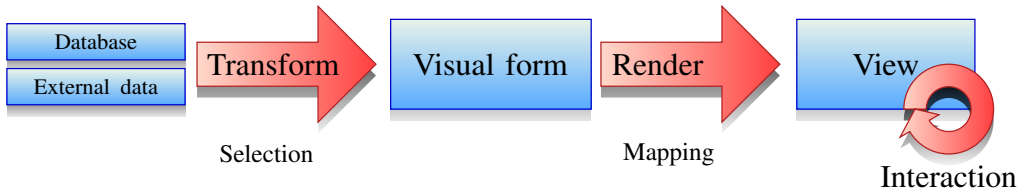


Figure 5.1: General flow of constructing an information visualization structure. Blue blocks are data formats, red arrows are translation steps.

5.4 Dashboard framework

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

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

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

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

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

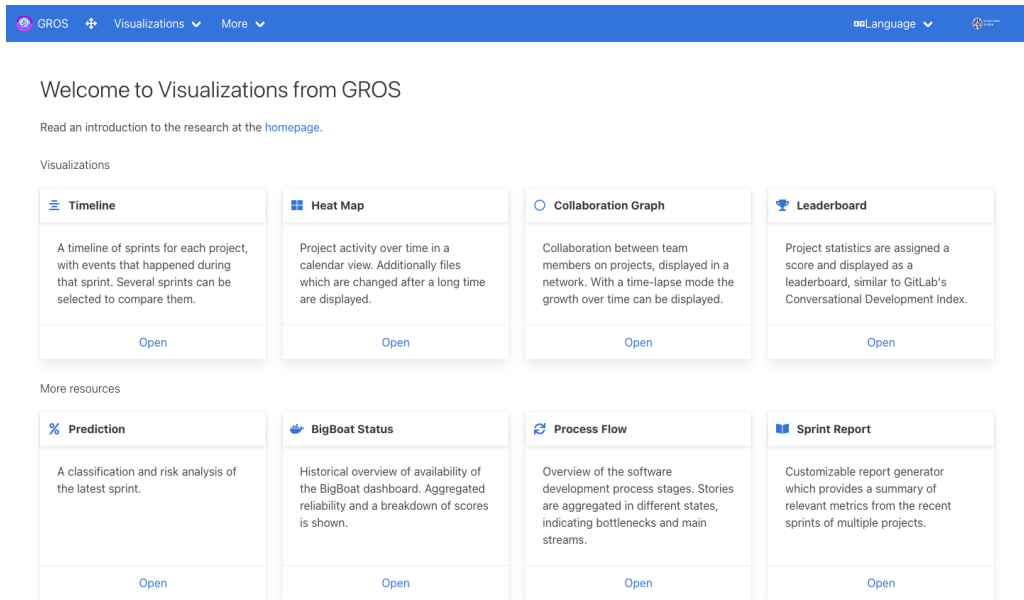


Figure 5.2: Dashboard of Grip on Software, providing access to eight information visualizations.

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

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

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

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

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

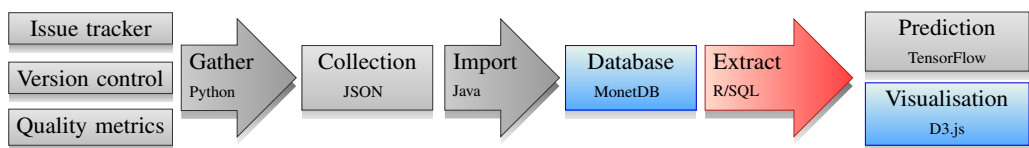


Figure 5.3: Overview of the Grip on Software pipeline, with the components related to information visualization highlighted.

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

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

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

5.5 Visualizations for analytical decision support

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

5.5.1 Sprint report

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

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

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

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

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

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

Data

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

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

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

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

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

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

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

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

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

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

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

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

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

Configuration

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

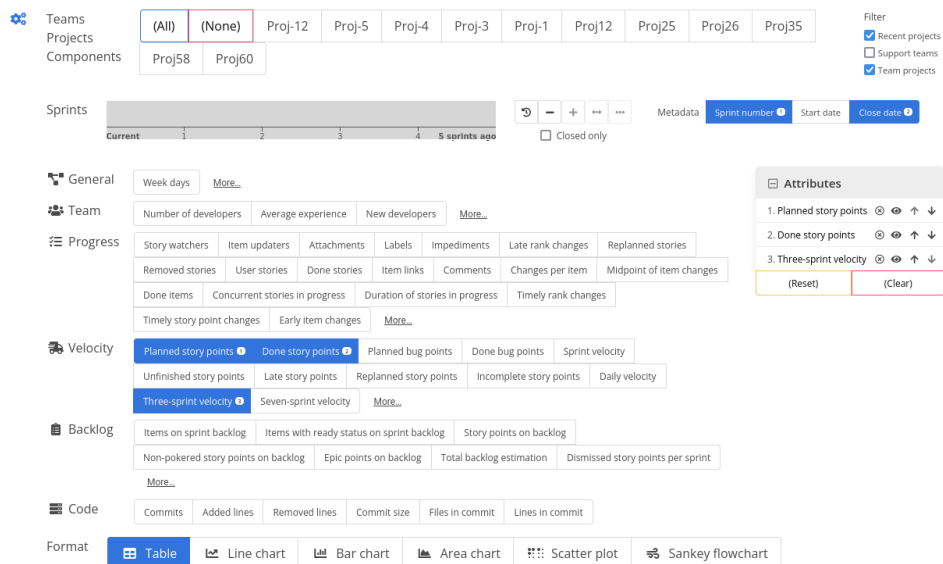


Figure 5.4: Configuration panel of the sprint report.

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

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

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

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

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

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

Table

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

Proj-12	Jan 3, 2022	Dec 6, 2021	Nov 15, 2021	Oct 25, 2021	Oct 4, 2021
Planned story points ↕	10	69	70	70	49
Done story points ↕	0	20	74	80	34
Three-sprint velocity	31 ½ points/sprint	58 points/sprint	62 ½ points/sprint	45 5/6 points/sprint	38 1/6 points/sprint

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

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

Line chart

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

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

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

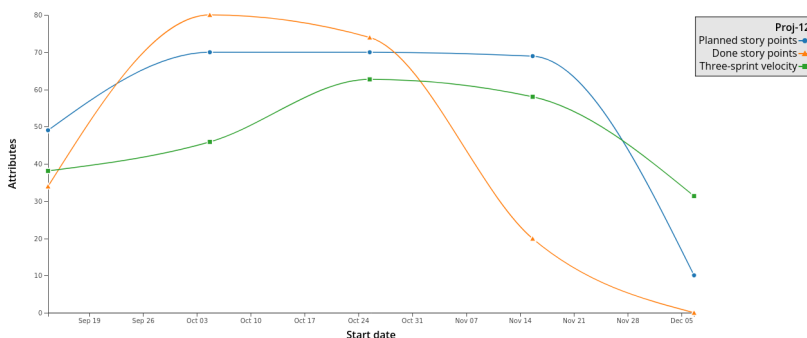


Figure 5.6: Example of the line chart format of the sprint report.

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

Bar chart

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

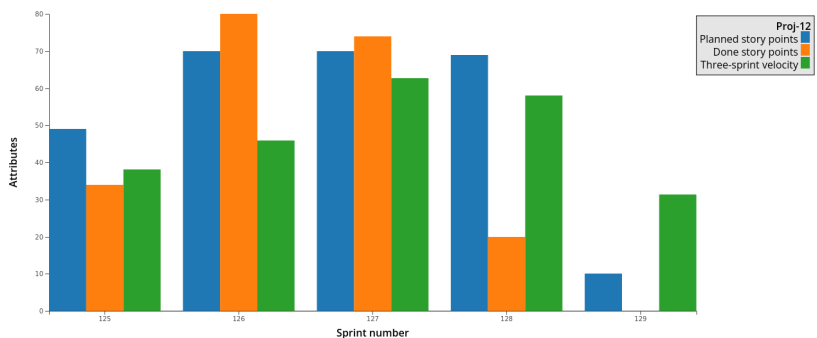


Figure 5.7: Example of the bar chart format of the sprint report.

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

Area chart

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

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

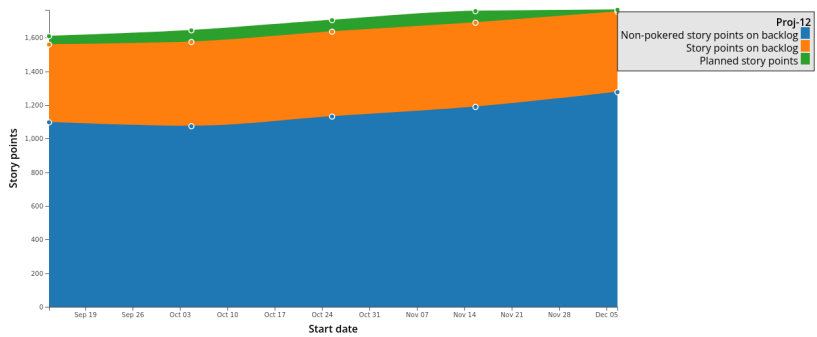


Figure 5.8: Example of the area chart format of the sprint report.

Scatter plot

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

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

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

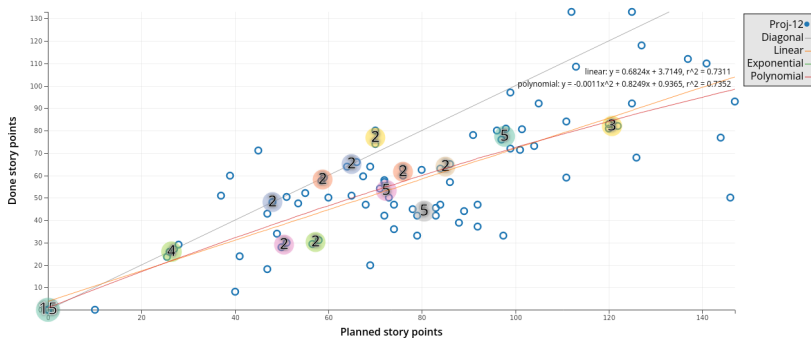


Figure 5.9: Example of the scatter plot output format of the sprint report.

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

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

Sankey flowchart

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

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

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

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

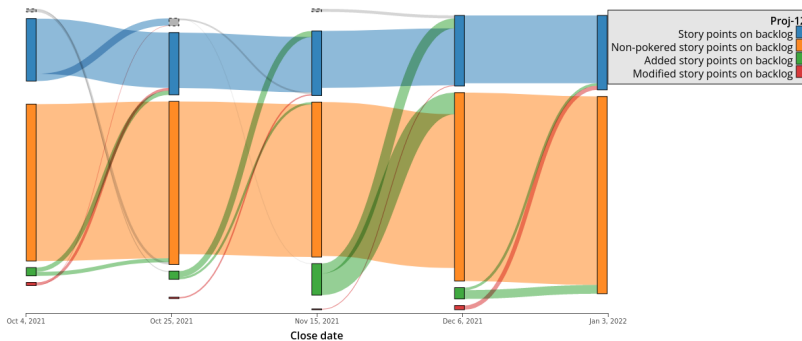


Figure 5.10: Example of the Sankey flowchart format of the sprint report.

Retrospective on interaction

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

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

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

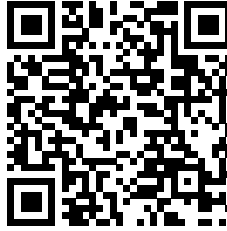


Figure 5.11: QR code of video demonstration of the sprint report.

5.5.2 Prediction results

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

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

Data

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

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

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

Rendering

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

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

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

Interaction

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

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

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

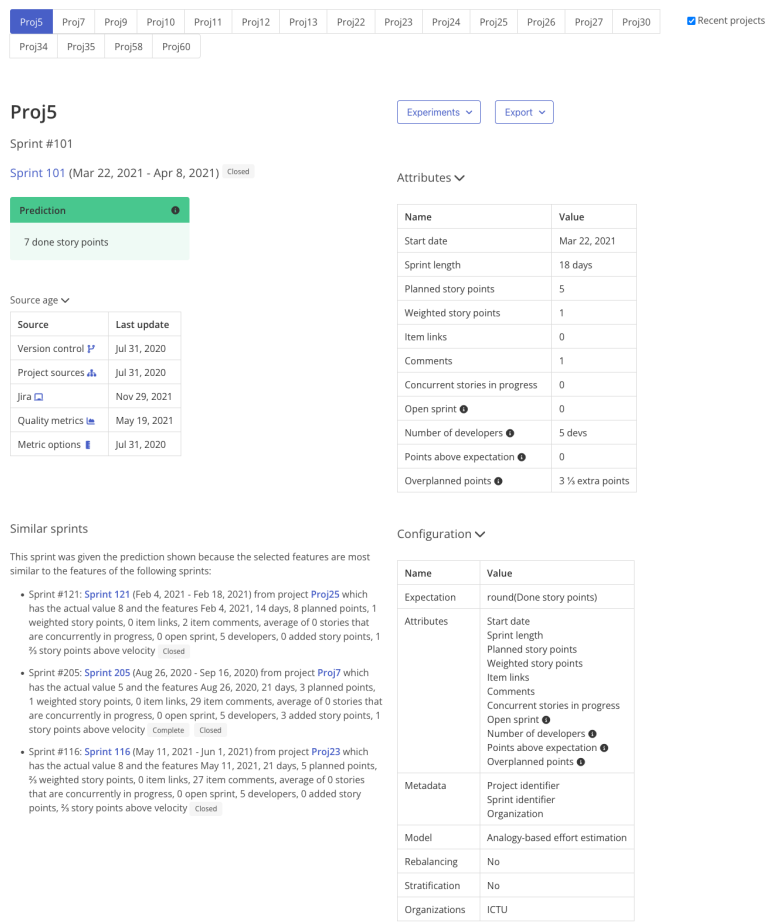


Figure 5.12: Estimated label for a sprint based on its features as shown in the prediction results.

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

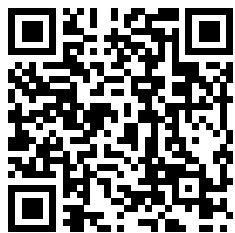


Figure 5.13: QR code of video demonstration of the prediction results.

5.5.3 Timeline

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

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

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

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

Data

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

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

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

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

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

Rendering

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

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

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

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

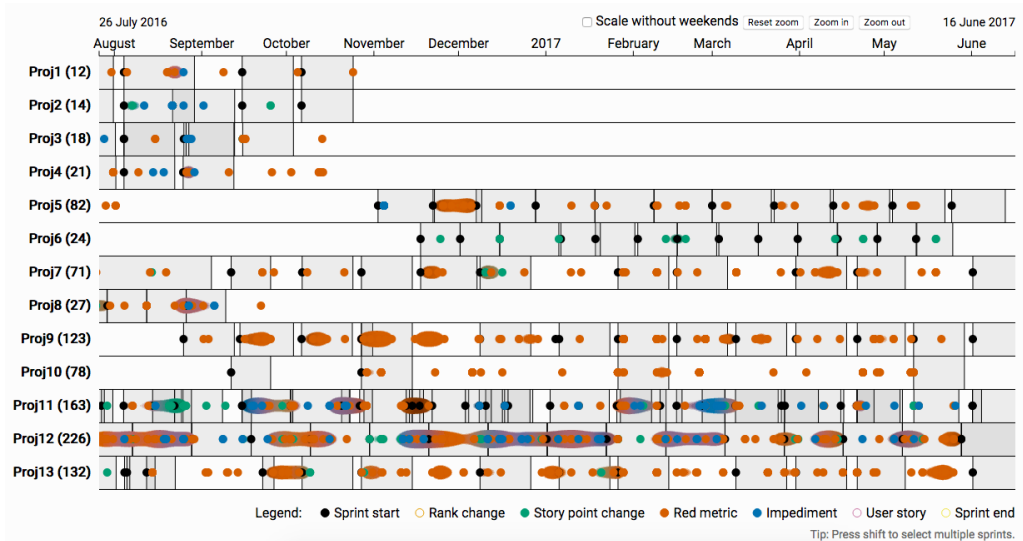


Figure 5.14: The default timeline view with colored event glyphs for projects displayed over time.

Interaction

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

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

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

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

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

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

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

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

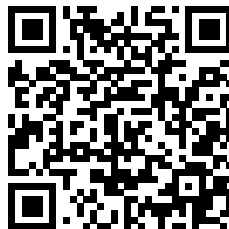


Figure 5.15: QR code of video demonstration of the timeline chart.

5.5.4 Leaderboard

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

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

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

Data

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

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

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

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

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

Rendering

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

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

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

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

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

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

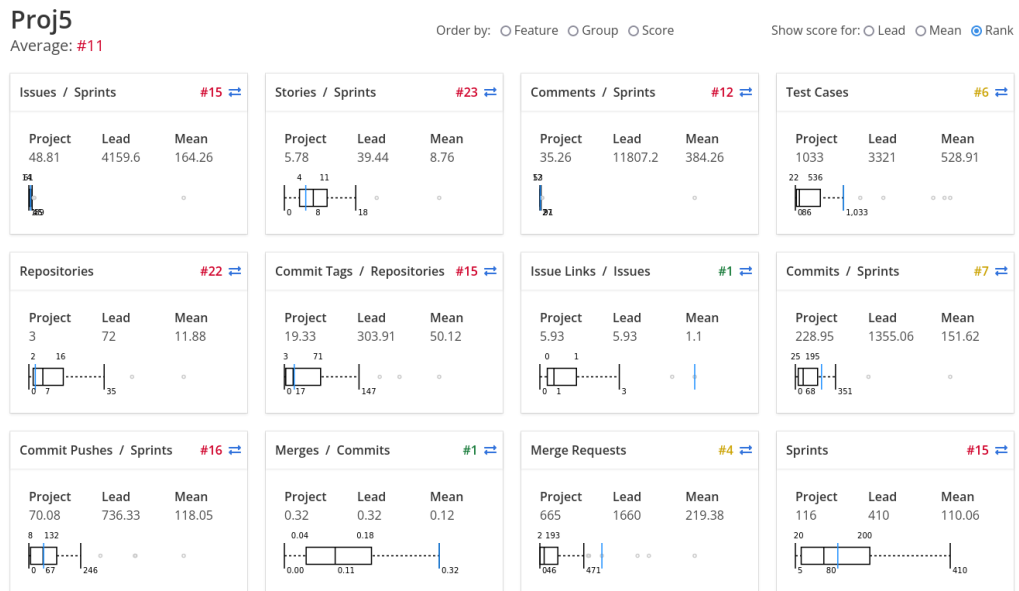


Figure 5.16: The leaderboard visualization for a project showing its ranks and cards.

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

Interaction

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

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



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

5.6 Visualizations for ecosystem management

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

5.6.1 Collaboration graph

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

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

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

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

Data

The Grip on Software database contains a number of models describing projects and people, which are collected from different sources, such as project trackers like Jira [1], version control systems, e.g., Git [11] and potentially the project registration system, which keeps track of estimations for full-time equivalents (FTEs) and seat counts that a development project has appointed.

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

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

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

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

Rendering

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

$$\forall (v, w) \in E : v \in P \wedge w \in M \quad (5.1)$$

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

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

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

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

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

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

Interactions

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

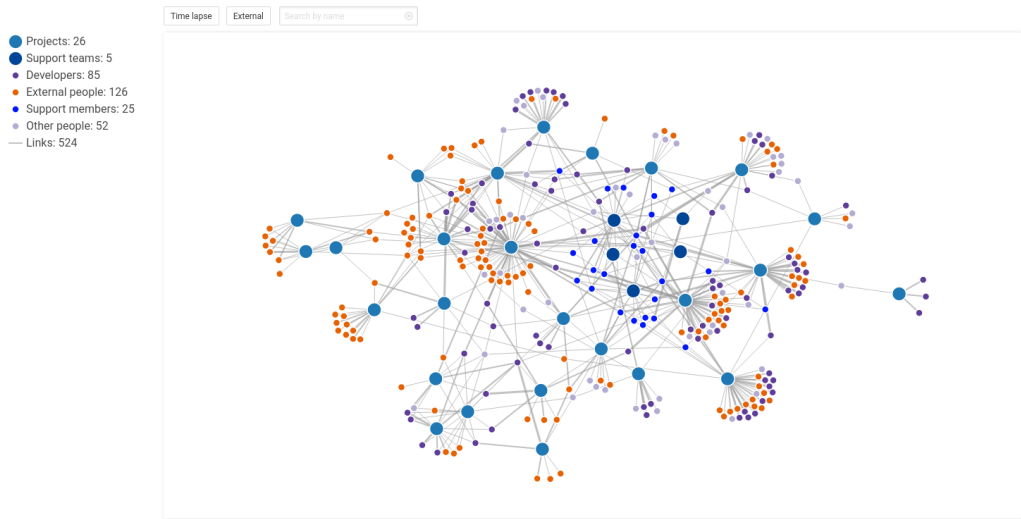


Figure 5.18: Visualization of a collaboration graph of a software development organization.

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

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

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

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

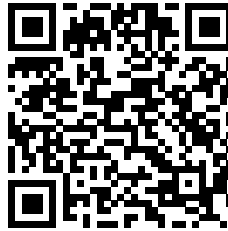


Figure 5.19: QR code of video demonstration of the collaboration graph.

5.6.2 Process flow

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

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

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

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

Data

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

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

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

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

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

Rendering

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

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

$$\begin{aligned}
 S_1 &= \{\text{Open, To Do, New, Requested, Design, Accepted}\} \\
 S_2 &= \{\text{In Progress, Approved, Reviewed, In Review, Ready}\} \\
 S_3 &= \{\text{Reopened}\} \\
 S_4 &= \{\text{Resolved, Committed, Done, Removed, Completed}\} \\
 S_5 &= \{\text{Closed, Validated}\} \\
 S &= S_1 \cup S_2 \cup S_3 \cup S_4 \cup S_5
 \end{aligned} \tag{5.2}$$

$$\begin{aligned}
 R &= \{\epsilon, \text{Fixed, Won't Fix, Duplicate, Incomplete, Cannot Reproduce, Not Fixable,} \\
 &\quad \text{Building, Manual Testing, Automated Testing, In Review, Processed} \\
 &\quad \text{Known Issue, Redundant, Works as designed, Invalid}\}
 \end{aligned} \tag{5.3}$$

$$\begin{aligned}
 \forall s_1 \in S_1: T(s_1) &= \text{Open} \\
 \forall s_2 \in S_2: T(s_2) &= \text{In Progress} \\
 \forall s_3 \in S_3: T(s_3) &= \text{Reopened} \\
 \forall s_4 \in S_4: T(s_4) &= \text{Resolved} \\
 \forall s_5 \in S_5: T(s_5) &= \text{Closed}
 \end{aligned} \tag{5.4}$$

$$\forall (s, r) \in V: s \in S_1 \cup S_2 \cup S_3 \Rightarrow r = \epsilon \tag{5.5}$$

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

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

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

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

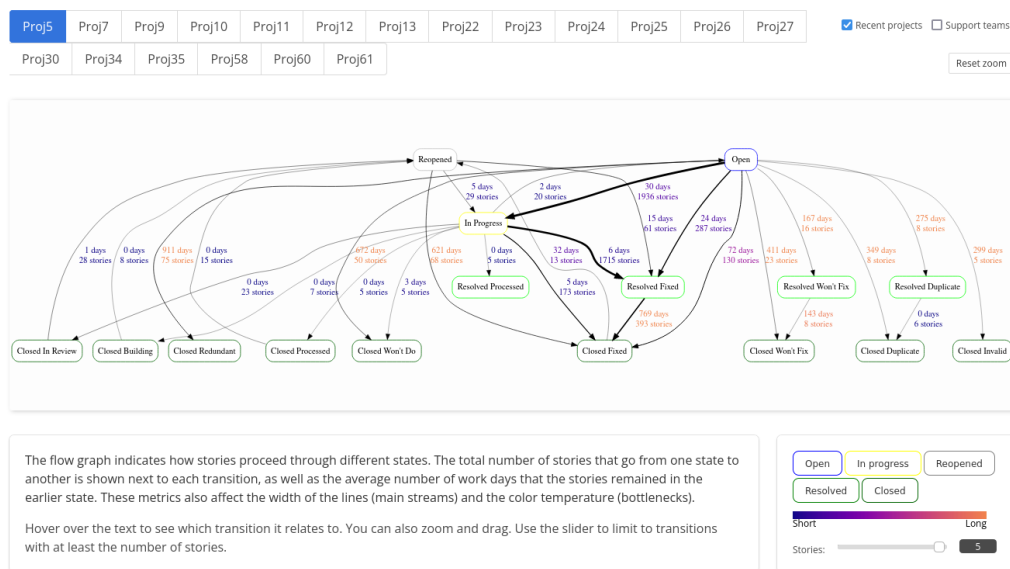


Figure 5.20: The flow graph of a project including legends and other controls in the process flow visualization.

Interaction

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

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

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



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

5.6.3 Heat map

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

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

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

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

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

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

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

Data

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

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

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

Rendering

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

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

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

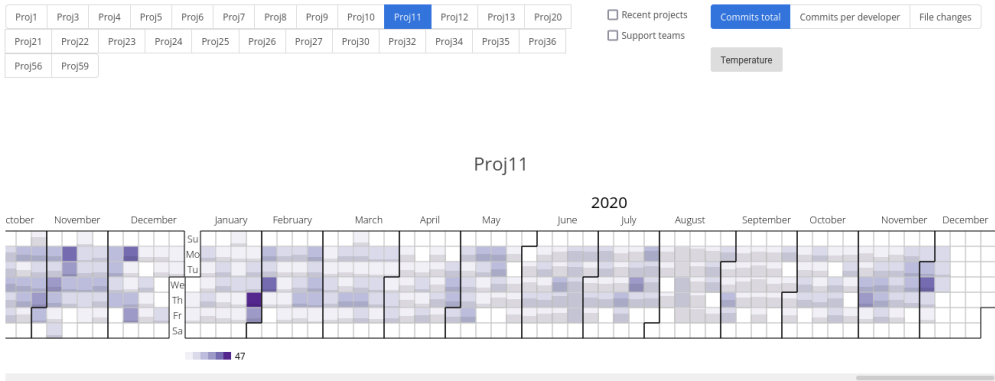


Figure 5.22: The heat map of a project, showing the most recent commit volumes including temperature bars.

Interaction

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

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

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

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

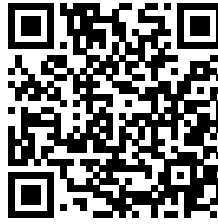


Figure 5.23: QR code of video demonstration of selecting projects and scales in the heat map.

5.6.4 Platform status

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

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

Data

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

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

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

Rendering

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

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

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

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

Interaction

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

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

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

BigBoat status

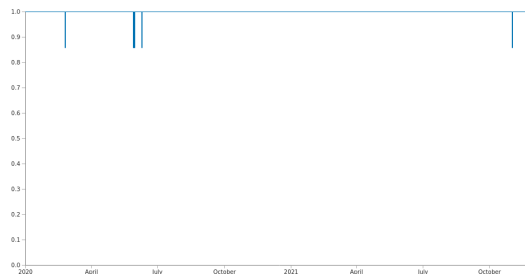
Proj5	Proj6	Proj7	Proj9	Proj10	Proj11	Proj12	Proj13	Proj20	Proj22	Proj23	Proj24	Proj25	Proj26	Proj27	Proj29
Proj30	Proj32	Proj34	Proj35	Proj36	Proj61										

Proj5

Full Month Week

Last checked: 2021-11-29 18:29:26

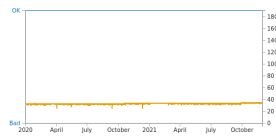
Average Reliability



Reliability per component

Available IP Addresses

An indicator of whether the dashboard can assign enough IP addresses for all the instances



BigBoat Agent

Whether the dashboard can reach the backend agent, which handles all actions regarding containers and storage buckets



Data Storage

Space that is in use by the storage buckets



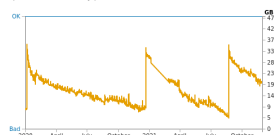
Docker Graph

Space that is in use by (intermediate) Docker images



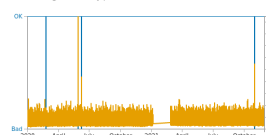
Memory

Space that is in use by processes in the containers



System Load

Load average caused by processes that want to use CPU time in the containers



System Uptime

Time since the dashboard was last restarted

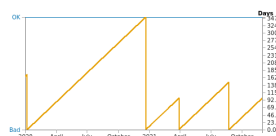


Figure 5.24: Platform status for a project.

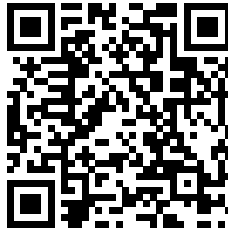


Figure 5.25: QR code of demonstration video of the platform status charts.

5.7 Novel backlog visualizations

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

5.7.1 Product backlog burndown chart

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

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

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

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

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

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

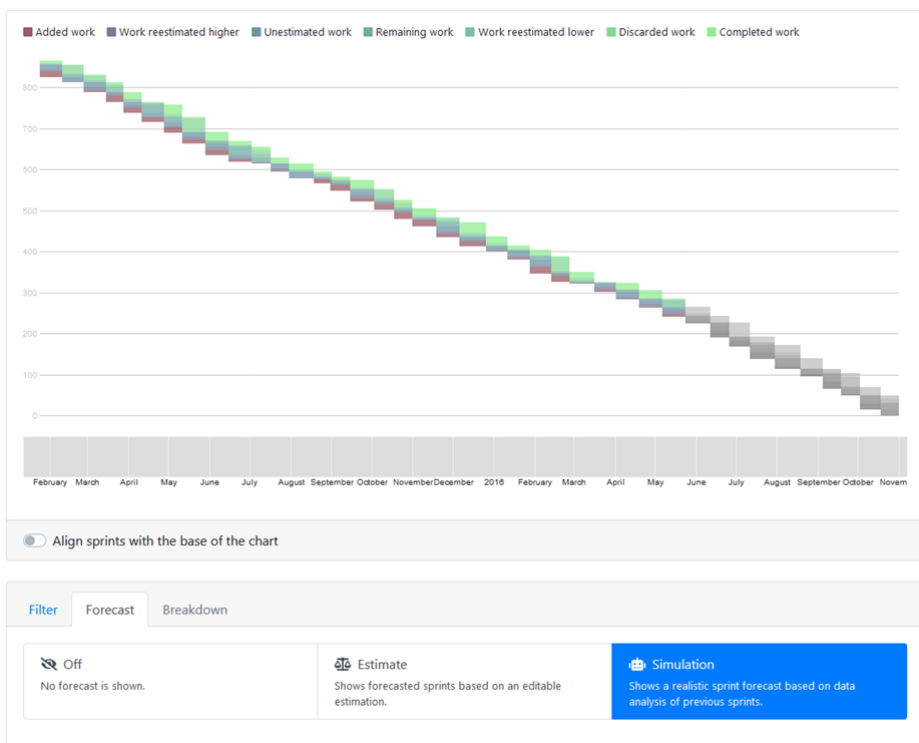


Figure 5.26: Product backlog burndown chart showing a Monte Carlo simulation of future sprints.

5.7.2 Product backlog progression chart

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

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

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



Figure 5.27: Example of the product backlog progression chart.

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

5.7.3 Product backlog relationship chart

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

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

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

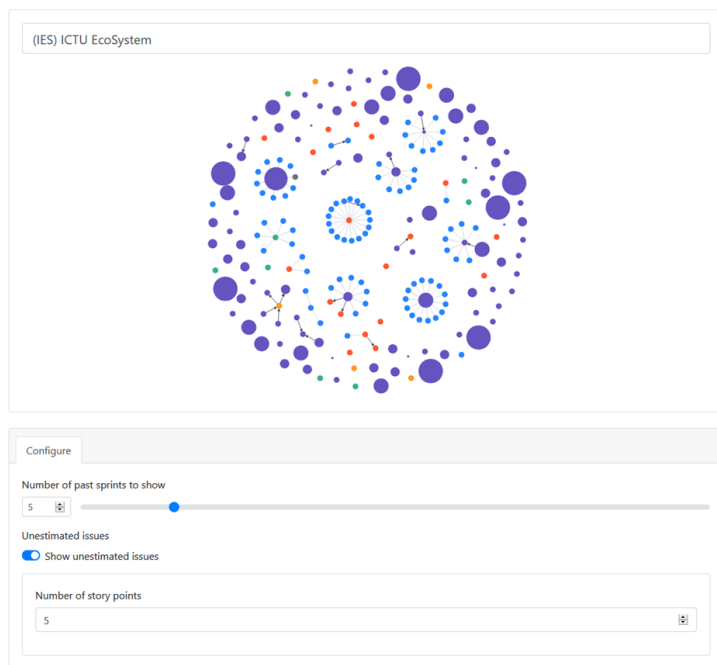


Figure 5.28: Example of the product backlog relationship chart.

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

5.8 Evaluation

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

5.8.1 Assessment

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

- **Similarity:** Functions that a visualization provides to enable access to certain controls or subselections of data, should behave the same way as the function that does this in another visualization on the dashboard. Examples of these functions are: selecting a certain project to display, comparing it to other projects, zooming in on temporal or coordinate data, displaying additional data via tooltips or exporting the data.
- **Traceability:** It should be possible to find out how a certain data point was determined, by providing information on how it was calculated or by adding a link to the source of the data, sometimes referred to as data provenance.
- **Tests:** Various portions of the visualization should be able to be found and interacted with using an integrated test, where a browser is controlled via an automated system which attempts to visit a web page on an isolated HTTPS website. It is then instructed to click specific buttons or other controls. The test then validates if certain HTML elements or other content is found on the page. Being able to emulate user input is also an indicator of perceptive difficulty, as visualizations that require fewer steps to work with are also easier to test this way.

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

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

VISUALIZATION	SELECTION	COMPARE	ZOOM	TOOLTIPS	EXPORT
Sprint report	✓	✓	✓	✓	✓
Prediction results	✓	✗	✗	✓	✓
Timeline	✗	✓	✓	✓	✗
Leaderboard	✓	✓	✗	✗	✗
Collaboration graph	✗	≈	✓	≈	✗
Process flow	✓	≈	≈	✓	✗
Heat map	✓	✗	✓	✓	✗
Platform status	✓	✗	✓	✓	✗
Burndown chart	✓	✗	✓	✗	✗
Progression chart	✓	✗	✗	✓	✗
Relationship chart	✓	✗	✗	✓	✗

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

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

An important design principle that we started out with is to primarily display data relevant to the patterns that we wish to present. Many of our visualizations are built around this principle.

Still, some users want to find more details, either temporally or within another coordinate space of the visual form. To do so, a number of visualizations offer a control which zooms the view, leading to a different selection being shown. Some visualizations use buttons or scroll wheels to zoom in and out. Another method is to select or adjust a rectangular area to zoom into using a “brush” or within the area itself, with another gesture to zoom out. The collaboration graph only zooms in when a search takes place. Each control is different, but is often indicated using a button, a mouse cursor or a search field. This addresses the need for this functionality.

Another method of making both information and controls more insightful and clearly labeled, while avoiding clutter on the page, is to place these details or guides in tooltips. Many visualizations use this, albeit in slightly different formats and situations. Most visualizations label the controls with longer descriptions. Some have larger tooltips that indicate more features for a project or sprint. However, in the collaboration graph and process flow, only small tooltips are used for some details, which is less usable and accessible for many users.

Only the sprint report and the prediction results have controls that export that data shown in the visualization in different formats. The integrated dashboard optionally shows a download link for each visualization. An exported version is helpful for offline inspection, reporting and standalone presentation. Specific export controls are buttons which open the selected format in another browser tab or provides a direct download, depending on browser settings.

Most controls and rendering approaches will feel familiar to users, such as buttons, checkboxes, sliders, graphs and charts. We consider these effects of similarity when it comes to adoption in Section 5.8.2. The other objective criteria are laid out in Table 5.3 and we discuss further how some visualizations fulfill these or not.

VISUALIZATION	TRACEABILITY	TESTS	MOBILE	ACCESSIBILITY	COLOR
Sprint report	✓	✓	✓	✓	✓
Prediction results	✓	✓	✓	✓	✓
Timeline	≈	✓	✗	✓	✓
Leaderboard	✓	✓	✓	✓	≈
Collaboration graph	✗	✓	✓	✓	✓
Process flow	✗	✓	≈	✓	✓
Heat map	≈	✓	✓	✓	✓
Platform status	≈	✓	✗	✓	≈
Burndown chart	≈	✗	✓	✗	≈
Progression chart	≈	✗	✓	✗	✓
Relationship chart	✓	✗	✓	✗	✗

Table 5.3: Comparison of assessment criteria regarding the visualizations (✓ is good, ≈ almost, ✗ bad/missing).

With regard to traceability, we have shown that some of the visualizations provide details on how features were collected and extracted, as well as providing links to where they come from when the visualization is deployed within the organization where access to the sources is possible. Often, there is only a link for specific entities, such as a sprint, a code repository or a Docker dashboard. Only the collaboration graph and process flow do not have links to their sources, making it harder to trace back their metrics.

All visualizations have tests that ensure that controls and expected rendering function properly. The only exceptions are the Jira plugins, which do not have automated test cases to perform these actions mechanically. The large majority of actions and gestures have been tested in this way and shown to be reproducible in this integrated setting.

When it comes to different screens and input devices, most visualizations work well in both small, mobile formats as well as in a larger lay-out. Additionally, we ensure that gestures that are important for the functioning of the visualization remain possible with touch or remote, although a small number of touch methods are not available when they conflict with normal scrolling behavior. For example, the leaderboard, where the cards properly fit a device size, cannot make use of dragging to create new normalization factors. The collaboration graph also provides slightly less interaction. The main issue for the nonconforming visualizations—the timeline, process flow and platform status—is the size of the charts or graphs that are displayed, which become too small when scaled down.

As part of our integration tests, we also use an automated accessibility testing engine [XXXII] which checks for common practices regarding web page structure and is able to find the majority of issues that could lead to failure compared to the relevant accessibility guidelines [XXXIII, XXXIV]. Any failures are aggregated and reported based on the tests. Meanwhile, browser extensions track them in development environments.

To check for potential problems for color blindness, the accessibility tests perform contrast ratio measurements between foreground and background elements, but this is not a sufficient test. We find that most visualizations use proper color schemes and/or glyphs with textual labels to differentiate colored data. The ranks used in the leaderboard use green, yellow and red, which is technically not enough to differentiate the different categories, although the rank numbers provide an indication. The platform status indicates collection dates with green and red. The Jira plugins use some colors that do not go well with another in the same rendering, despite an attempt to improve upon the original color scheme.

Overall, the visualizations provide an integrated, familiar, accessible and clear experience, putting the focus on the information being displayed. By sharing interface controls and applying them in similar ways, users are able to learn how to use a new visualization more quickly and to customize it according to their needs. Through tests, we are able to ensure that the visualizations remain available according to these criteria. This also clarifies goals of further development of a data hub for developers. Improvements to the functionality easily fits along with the existing situation.

5.8.2 Adoption

During the Grip on Software research project, we performed a limited amount of evaluation of usability and adoption among representative users. For three early-developed visualizations, namely the timeline, collaboration graph and heat map, we conducted a survey including a System Usability Scale (SUS) questionnaire. This provided some insight in the potential of these visualizations and steered the focus afterward.

The questionnaire showed that the respondents found the visualizations to be easy to learn and easy to use. One point of contention was that it many users would not be frequently accessing any of the three visualizations during their usual work routine. A likely factor that the questionnaire accentuated was that, at the time, the three visualizations did not make clear what certain elements were meant to do [118].

Based on more meetings, discussions and presentations, we focused on integration of new visualizations, predictive patterns and situation reports. The resulting visualizations are an iterative effort, where prototypes were displayed to stakeholders for further feedback and desires. This included exploring which kinds of attributes were more relevant to them. We often enrich the result rather than leave out the data that was found less interesting, so that it is still accessible through a tooltip, for example. In order to ensure that features exhibited data that users felt familiar with, we collaboratively looked for reasons where data did not adhere to expectations and adjusted filters and other conditions. This also improved the attributes for the prediction and estimation algorithms, as the input data more accurately follows the events that they described.

Specific care was taken for the information visualizations with regards to usability. Users tested whether they could acquire the knowledge provided in them, including users with color blindness. The novel structuring of data allows some users to extract knowledge that they could not easily find before. Providing existing data in fresh formats through intuitive interactions helps users find their way more quickly [119]. In particular, the sprint report and prediction results were used by various Product Owners, software delivery managers and quality managers for multiple teams. The collaboration graph's overview and timelapse mode was seen as helpful at an organizational level, as well as a nice way to showcase the organization's own history.

The predictive power of the sprint report's future sprints as well as the backlog burndown chart has helped team members in leading roles to find out when a project would finish if no additional stories are added to the backlog, or when additional features need to be suggested by the client in order to maintain enough volume. Based on exploratory interviews with users that have various roles in the organization, an inventory of missing insights was formulated. These ideas further influenced the design of the novel backlog visualizations.

5.8.3 Conclusion

In this thesis, we study patterns and analyses regarding the aspects and iterative results from SCRUM software development processes. This chapter focuses on delivering the research output to the relevant stakeholders, including development team members, Scrum Masters and Product Owners. We aim to do so in an intuitive way that integrates with the existing development ecosystem. We create a dashboard of multiple information visualizations to make our results accessible to the aforementioned stakeholders, highlight different types of data and ease the discovery of useful patterns. We provide sprint reports, prediction results, timelines, leaderboard ranking, network graphs, flow graphs, heat map calendars and platform status monitoring. Finally, we design various backlog visualizations that integrate with Jira, an often-used issue tracker.

These visualizations are designed in a way that they meet certain requirements when it comes to applicability within the software development framework. The goal is to improve the process, by supporting the stakeholders in finding out what went well and what can be improved. The information is provided with clear labels that describe what it means. Focus is placed on the important factors, but it is still possible to inspect details and trace sources through zooming, clicking links or expanding elements [109].

The view and interaction of each visualization is based on considerations for colors, icons, glyphs, gestures, interactions and controls. The rendering of the visualization is based on a visual form, such as a graph or a collection with relationships between certain fields. The visual form is defined by the type of data that we select and filter from the data pool, which includes the Grip on Software database, results from prediction algorithms and external data sets.

The information visualizations produced by the Grip on Software project are available as code repositories and a live website is publicly accessible for research purposes^{††}. The visualizations on this dashboard use anonymized data; teams, projects and people have no identifying information. Moreover, there are no links to the systems where the data is collected from. Versions of the dashboard with descriptive identifiers and source links are available to the organizations that were involved in the research, namely ICTU and Wigo4it. The Jira plugins with novel backlog charts are also deployed at ICTU. We consider the application of these visualizations in two organizations to be part of our case studies into their specific workflows, while the produced results and evaluations make them relevant for wider application [120].

We introduced automated metrics in our assessments through unit tests and accessibility tests, in addition to other objective metrics and user feedback [121]. Together, they are a part of our iterative design and construction process for the visualization phase of the GROS pipeline.

Selecting a proper visualization to support the user in making an informed decision within a process is sometimes difficult, especially when there are many potentially relevant metrics and features to provide to them [122]. For our visualizations, we have taken some design principles in mind, which include familiarity, relatability, simplicity and intuitiveness. These principles mostly revolve around keeping the default view for the user focused and uncluttered, while providing details when a clearly labeled control is interacted with. Each visualization has different formats that were chosen to fit with the type of data.

The application of information visualization within Agile software development methods is an area of research that has not been extensively explored. Most earlier work focuses on visualizing specific aspects or abstract flows. The integrated dashboard provides new opportunities for stakeholders to obtain the knowledge to improve their processes.

In conclusion, we demonstrate that it is worthwhile to provide results from predictive models regarding SCRUM software development in the form of information visualization to those involved in the process. Further research could show how the adoption of the visualizations affects long-term decision making, for example regarding backlog planning. Finally, a full usability specification, study and survey—with an updated SUS questionnaire involving more of our visualizations—helps position the use of visual interaction systems within Agile software development frameworks even better.

^{††}<https://gros.liacs.nl/visualization/?lang=en>