Universiteit Leiden
The Netherlands

Opleiding Informatica

Visalization of Patterns in

Scrum Software Development

Laurens Groeneveld

Supervisors:
Prof. Dr. Ir. F.J. Verbeek & L.S. Helwerda MSc

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

July 27, 2017

Abstract

ICTU is a Dutch governmental agency that creates software products for various branches of the government. To manage their development process they use the Scrum methodoloy. Over the years they have gathered a lot of data related to the software development process. To make this data more accessible and to provide insights for different stakeholders three visualizations were created. First, a heatmap which shows a comparison between external data such as temperature, and internal data such as the number of commits per day. Secondly, a network graph which shows the relations between people and the projects they work on, making it visible who works on multiple projects. Lastly, an existing timeline visualization was enhanced with a burndown chart to better show velocity. We find that these visualizations can help with identifying patterns in large datasets, but the data and measure needs to be chosen carefully. Usability is very important to help the user understand the purpose of the visualization and find value in it.

Contents

Chapter 1

Introduction

Scrum is a methodology frequently used by software development teams to manage their development process. The Dutch governmental agency ICTU creates software products for various branches of the government and uses Scrum. Over the years they have gathered a lot of data related to the software development process. A collaboration between ICTU and Leiden University, Grip On Software (GROS), researches how this data can be used to improve the development process. The researches closely collaborate with different stakeholders, such as team members and management, to provide useful insights into data from current and previous sprints. This should provide ICTU with a better understanding of their processes and ultimately lead to software products that more accurately mirror the requirements. In this thesis we will explore new visualizations that can be used to provide insights into this data for the different stakeholders, including researchers. They can be used to discover patterns and anomalies in the data, giving an overview of certain areas and leading to possible new research directions.

1.1 Research Question

What are "good" visualizations for trends in software development using Scrum?

To what extent a visualization is "good" can be determined by measuring the usability of the visualization. This will be more detailed in Chapter 2.5.

1.2 Subquestions

To answer the research question, three visualizations will be created, each with their own goal. To provide guidance a research question is formulated for each of them:

1. *How can external data, such as temperature, be correlated to project activity?*

   The dataset used for this thesis consists of a large amount of gathered data, however this is all internal data. It's possible that external data provides insights and can be correlated to this internal data.

2. *How can relations between members of different teams be visualized?*

   There are a lot of different teams and projects at ICTU, however many people work at multiple projects. It can be interesting to research how this can be displayed.

3. *How can velocity in a burn down chart be better displayed in a timeline graph?*

   An existing visualisation used at ICTU is a timeline with sprint data. It only displays events, where velocity can be another interesting metric to be displayed.

1.3 Thesis Overview

This thesis is structured as follows. In Chapter 2 the background of this thesis is discussed: more about ICTU, a short explanation of Scrum, and an overview of the GROS project and the different tools that ICTU uses. In Chapter 3 the available data and methods to create the visualizations are discussed. Chapter 4 evaluates the experiments, with a part for each of the three visualizations. Chapter 5 concludes and briefly discusses future work. The main conclusions are that correlating external data to internal data can be useful, but not in the form of the created heatmap visualization. A network graph can show relations between team members and projects and show which people work the most on different project. Different roles can be shown in such a graph. Adding a burn down chart to the timeline visualization provides more insight into the velocity of a team during a sprint.

This "bachelor thesis" was written as part of research done at LIACS, and was supervised by Fons Verbeek and Leon Helwerda, at ICTU Frank Niessink was involved in the supervision.

Chapter 2

Background

2.1 ICTU

The research described in this thesis was done at ICTU. ICTU is an IT organization which is part of the Dutch government. They work solely for the government and as stated on their website [1]. ICTU is "an independent consultant and executor within the government". The Ministry of the Interior drives the basic outline of ICTU's management. ICTU works on IT projects for all government layers, including municipalities, provinces and ministries and other government instutions like the National Police.

ICTU is organized into three branches. *Directie Advies* is responsible for acquiring and management of new clients, communication (both internal and external) and architecture, which converts complex issues into workable architecture solutions. *Directie Projecten* is the executive branch of ICTU, consisting of a project management department and *Software Realisatie* (ISR), the actual department responsible for creating software products. This is the part of the company where this thesis was completed. The third branch of ICTU is *Directie Bedrijfsvoering*, responsible for supporting the ICTU organization internally.

Software products that are developed by ISR are mostly internal applications for the clients. Examples of projects include a registration for people who live for a short time in the Netherlands, tackling address and ID fraud and a registration for non-academic teachers. More information about the specific projects ICTU works on can be found on their website [2].

2.2 Scrum

The research of this thesis deals with visualizing data from the software development process at ICTU. All new projects at ICTU are developed with an *Agile* methodology, specifically *Scrum*. The term 'Agile' was coined in 2001 by the authors of the "Agile Manifesto" [3]. It described what they had come to value with their work in software development:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

An important part of working software is to enable developed software to be deployed at any time (usually after each sprint). Another important aspect is collaboration with the customer throughout the development process and allowing for changing requirements during the development.

Framework

Scrum is a framework that describes an agile software development process [4]. Scrum divides the process into a finite amount of *sprints*. A sprint is usually two to four weeks to which a small chunk of the *product backlog* is assigned. The product backlog is a prioritized wishlist of features, created by the *Product Owner* - usually the client. These features are usually formatted as *user stories*. A user story is a natural language description of the feature, in the form of ”As a ’role’ I want to ... because ...”, where the role can be for example ”end user” or ”system admin”.

For each sprint the product owner decides what will be implemented from the sprint backlog in a *refinement* session. During this refinement they can also decide that certain user stories need to be made more clear and cannot be implemented in the current form. The relative complexity of each story is determined with the ”planning poker” method. The team allocates each story a certain number of story points. These points have an ascending scale, starting at 0.5 or 1 and doubling for each step, so the sequence has the form of 1, 2, 4, 8 etc. A larger number of points means the feature will take more time to implement. Based on the number of points the team finished in previous sprints they select a number of stories for each sprint to finish.

At the onset of the sprint they make a commitment to implement all selected features, though this goal can also not be met. In that case all available features in the end product should be working, otherwise one or moreselectedfeatures will get dropped from the sprint.This happens because inScrumthe duration of a sprint is not adjusted to cater for the actual amount of time needed but rather the scope is adjusted to cater for the time available.

Each day a software development team meets in a *Daily Scrum* (or ”standup”) where they discuss the progress made.

Note that there is no formal ”management” role in the Scrum process. For each sprint a ”Scrum Master” is assigned to the team who helps to keep the team focused on its goal, but does not decide what the team will create or how they will implement the features from the backlog.

At the end of a sprint, the developed software should be working and shippable. This is part of the ”working software” principle of Agile and enables continuous deployment. As indicated earlier, features that are not

working should be removed from the product increment. Any work that is not finished in a sprint gets put back on the product backlog. For the next sprint, the team again decides which user stories to work on.

The team meets at the end of a sprint in a *retrospective* session. There the team discusses how they experienced the sprint and if the rules that were put in place before need any change.

Note that through the Product Owner the client collaborates throughout the process with the team, but the team decides which features to implement in each sprint. Usually features that the client prioritizes are worked on first, but the software developers have the freedom to choose otherwise.

A project can end for different reasons: Most or all of the product backlog is implemented, a specific deadline which was decided beforehand is reached, or the client decides to discontinue the development process (because the budget is depleted or the product is cancelled).

At ICTU the Scrum framework is implemented like described above. Most projects have sprints of two weeks although in practice sprints of three weeks occur as well.

2.3 GROS

Grip On Software (GROS) is a collaboration between ICTU and LIACS, Leiden University. It aims to improve the quality of the software developed at ICTU by analysis of measured data. Data from both the software development process and the end product are taken into account. GROS focuses on three areas: Historic lessons from software engineering, prediction of success of software projects and measuring the effectiveness of changes.

GROS has two distinct goals: On one hand, it will determine objective criteria and norms which determine the quality of the software development and end product. On the other hand, these norms can then be used as a augmentation for the quality system and tests that are already in place at ICTU.

The research within GROS is targeted at sprint level. There is a limitation in the number of projects to ensure reliable or representative results, but there are enough sprints whose properties differ throughout the development of the product. For a start, the project will focus on analyzing gathered data, but in the future the results could be use to predict success of sprints.

It is believed that combining visualization with data mining algorithms provides for more effective exploration of the data, while preserving user control [5]. This thesis primarily looks into providing visualizations. The overall GROS project is (more) concerned with mining of data insights, which can then be made accessible in a visualization.

It should be noted that not every project at ICTU currently takes part in GROS. Each project team, and in some cases the client, needs to approve before their data is shared with researchers within GROS. We discuss our use of sensitive data in Chapter 3.

2.4 Tools

ICTU uses a diverse toolset to help in the development and quality assurance of their products. The data used within GROS uses a large number of these tools to gather the desired information. For more information about which data exactly is gathered and how it is used, refer to Chapter 3.

JIRA

JIRA Software is used for project tracking, user stories and issues. It provides features needed to plan sprints and projects and keep track of problems (issues) in the product. JIRA is used extensively at ICTU and is one of the primary data sources for GROS. With JIRA, teams keep track of their tasks and have an overview of the remaining work. It provides an overview dashboard where teams can view at a glance the status of the project. JIRA is also used to keep track of product releases. While working with sprints, it provides an spare burndown chart, showing the completion rate of user stories during a sprint. Issues in JIRA have a range of properties, such as a priority, the developer assigned to fixing the issue, any comments made on the issue and file attachments.

GitLab

Most projects at ICTU use Git as their source control system. Each project has their own GitLab instance where the different Git repositories for that project are hosted. It provides an interface to browse the repository and to download (”clone”) a local copy. GitLab allows for ’Merge Requests’, meaning that new code is not immediately merged with the main branch but often first reviewed by a team member. Other developers can also comment on specific lines of the code.

Quality Dashboard

In order to continuously measure several quality measures during a sprint ICTU has developed its own quality system with a dedicated Quality Dashboard [6]. This dasboard displays several metrics and whether or not they conform to the (configurable) norm (Figure 2.1). Sources include JIRA, source control (with metrics such as the total number of lines) and SonarQube, a system to measure code quality [7]. The intended use is to look each day during the Daily Scrum to the dashboard to see how the project is going. Red metrics mean they require immediate action and need to be looked at. Yellow metrics are almost conforming to the norm, and green metrics are meeting the norm. A team can accept certain degrations as technical debt, in which case they are not reported as needing immediate action anymore, but need to be looked at at a later date.

Periodically, a new report is generated with the latest data from all configured sources. Every report is saved so the history of the metrics for a project is available. This data is then used for data analysis and the visualizations.

Figure 2.1: An example of metrics on the quality dashboard.

2.5 Usability

As part of this thesis multiple visualizations were created (see Chapter 4). To see if they have value for the targeted stakeholders, the usability of the visualizations can be measured.

Heer et al. find that not all users of a visualization have the same skillset or the same interest [8]. To provide the maximum amount of value for the user, a few things are needed: automatic selection of visualization type, which is of course always done as part of the created visualizations, useful defaults so the user can use the visualization right away, and contextual information such as legends, labels and pop-ups, which inform the user what is going on. These are especially visible in the timeline visualization (Chapter 4.3), which has a clear legend and provides a useful default view.

Colors

An important part of visualizations is the color choice. Color makes it possible to distinguish different items. While it is easy to use established colors such as green and red to indicate ”positive” and ”negative”, these colors are not easy to distinguish for color blind people. Therefore, color blindness should be taken into account when designing the color scheme of an application. For visualizations which use one dataset one color or even a single hue can be used, with different brightness. Even while the color may not look the same to people with color blindness, they can still spot different brightness. For visualizations that do not use a scale but have several distinct values, more colors need to be used. Bang Wong created a color scheme which he found distinguishable for people with different forms of color blindness [9]. This color scheme is used in the timeline visualization.

Measurement

To measure usability several actions can be performed. For example, it can be measured how long it takes a user to do a certain action. To measure how a user thinks about the usability of a system, a System Usabilty

Scale (SUS) questionnaire can be used. The SUS measures the opinion of from a user regarding a device, in this case a visualization [10]. It uses a Likert scale, meaning that users indicate for then questions how much they agree or disagree with them, in a range from 1, meaning 'Strongly Disagree', to 5, meaning 'Strongly Agree'. For each of the visualizations a SUS questionnaire was conducted, with the results listed in Appendix A. The most interesting results will be discussed in the chapters for the experiments.

Chapter 3

Method

3.1 Data

To create the visualizations data from different sources within GROS is used. Some of these tools were already outlined in Chapter 2.4. The data includes information from both previous and current sprints from the different projects that partake in GROS. At the moment of this writing data from 13 projects is available, with 501 sprints in total. Primarily, there are these sources:

- **JIRA**. JIRA is used for user story and issue tracking during each sprint. Information gathered from JIRA includes sprint information such as start/end date, velocity and planned stories and the progress of those stories. Using this, over or under committing can be measured. Furthermore, history changelogs of all stories and issues are available.

- **Quality Reports**. The quality reports measure the progress of projects based on data provided by JIRA but also other systems. It then determines which metrics meet the norm and which ones need attention. Available information includes project information such as the name of the project and sources of the metrics, and data of all metrics such as the actual measured value, the category, the moment at which a metric measured, and the norms per metric and when they were changed.

- **Version Control**. Most projects use the Git version control system. Some projects use Subversion instead of Git, both are available in the dataset in the same format. For Git projects, either Gitlab or Team Foundation Server (TFS) is used to share files and serves as a host for the different version control repositories of each project. All commit data from repositories is available, including the commit message and date, the number of insertions and deletions and the changed filenames.

- **Topdesk**. Topdesk is a self-service reservation system where employees can book rooms to hold meetings and report any problems. The reservations can help in providing an overview of when teams hold meetings and how well they keep track of this.

Furthermore, project documentation is available for some projects but is not used currently in the GROS research project. The same applies to timesheets of Timetell, the administrative tool used to collect work times.

Limitations

While the current dataset has data from a lot of different sources, there are several limitations. First, not all data can be currently gathered by an automated system - prime examples are project documentation, because teams use many different systems to manage, and data from Topdesk which needs to be exported manually. This hampers efforts to provide continuous insights into how projects or ICTU as a whole is doing, as recent data can be unavailable.

Another problem with the data is that not every team member follows the (Scrum) process to the letter, in contrast to what is indicated in Chapter 2. Sometimes the administration in JIRA can also deviate from the process. For example, some user stories change their state from open to done, without the normal 'in progress' step. This can interfere with the ability to do proper data analysis or visalization as there can be noise, which need to be dealt with.

Lastly, some data is simply not available yet in the dataset, but could provide scientific benefits. An example is how the support teams of ISR work and whether support requests from teams get handled quickly. There are also values gathered from the quality system, and those could be expanded with getting the default values in the dataset.

Sensitive Data

Some data is sensitive and needs to be adapted, such as personal data. In the end, there should no data remain that makes a project or person recognizable. Especially for team members this is an extra challenge as they often need to be identified across different data sources, but still be anonymous in the sense that it is not possible to trace it back to the actual person. For example, for data analysis or visualization purposes a person from version control needs to be mapped to the same person in the JIRA issue tracker.

3.2 Pipeline

To create visualizations the data first needs to be gathered to a common format for easy processing. Most of the earlier mentioned sources provide an API which allows for automatic gathering of both the current and previous data. All sources are gathered and processed using different Python scripts, and then imported in the database with a Java program (Figure 3.1).

The database is a MonetDB instance. MonetDB is a column-store database management system (DBMS) targeted at applications that need to analyze large amounts of data [11]. It uses several techniques to make working with large datasets more efficient [12]. The use of column based storage instead of more traditional row based storage makes working with all columns from a table faster. Compression and different optimizations are used to reduce memory usage. MonetDB supports SQL and provides programming interfaces for various programming languages.

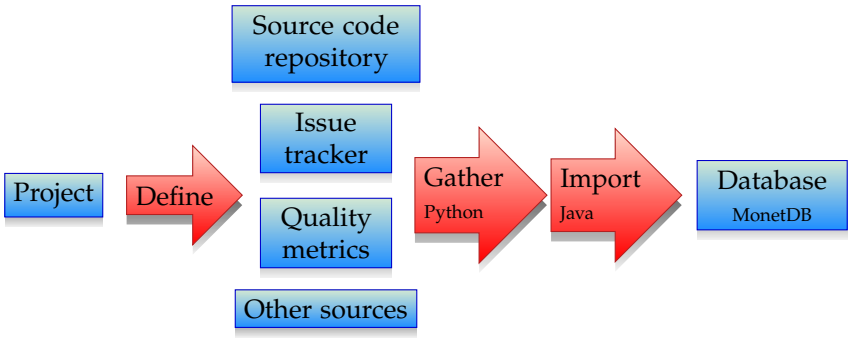


Figure 3.1: Pipeline for gathering data.

Queries

When the data is imported in the database, it is available for extraction to use them for data analysis or visualizations. This can be done by querying the database using SQL, and then feeding that data to an algorithm or visualization.

The database is divided into a large amount of different tables. Not all tables are relevant to the visualizations of this thesis and some are really straightforward, like `project` which simply holds the project names and their internal identifier (`id`), which can then be used to couple mentioned projects in other tables with a project name.

On the basis of a sample query used for the Heatmap visualization (see Chapter 4.1) we will explore some of the tables and possiblities of the dataset. This query retrieves the amount of commits for each day that has commits.

Listing 3.1: Query to retrieve amount of commits per day for each project

```
1 SELECT project_id, CAST(commit_date AS DATE) AS commit_day, COUNT(*) AS value
2 FROM gros.commits
3 JOIN gros.vcs_developer ON commits.developer_id = vcs_developer.alias_id
4 WHERE vcs_developer.jira_dev_id <> -1
5 GROUP BY project_id, commit_day
```

The whole query is as shown in Listing 3.1. To explain how the data is retrieved, we will break the query down into several parts and annotate each line. Note that all tables are prefixed `gros`, this is simply the database

identifier.

- Line 1

```
SELECT project_id, CAST(commit_date AS DATE) AS commit_day, COUNT(*) AS value
```

  The end result is a list of items with a `project_id`, the date (`commit_date` is cast as a date instance, `commit_day`) and the number of commits (`value`) for that day.

- Line 2

```
FROM gros.commits
```

  The three fields from line 1 are retrieved from the `commits` table. This table holds all commits for each project, with all associated metadata. This includes the commit message and date, which sprint it belongs to and the number of changes. In this case, we are only interested in the total number of commits for each day, but more interesting information is available (and used for other visualizations).

- Line 3-4

```
JOIN gros.vcs_developer ON commits.developer_id = vcs_developer.alias_id
WHERE vcs_developer.jira_dev_id <> -1
```

  The commits table is joined with the `vcs_developer` table. This table is an intermediary table used to identify developers between tables. It holds their name and email (both encrypted) and two identifiers, the `alias_id` and the `jira_dev_id`. The alias is an id used to identify developers in version control systems, so this is matched with the `developer_id` on the commits table.

  The `jira_dev_id` is to match people with JIRA and is separate from the developer id, because not all people do commits in version control (for example, the Product Owner).

  The join in this case is to filter out people with a `jira_dev_id` of -1. There are three cases for this id. Team members which can be matched to other information are assigned a unique number greater than zero. An id of -1 is assigned to automated systems which can do commits in a repository. These are filtered out as they influence the result for the visualization, which is not wanted as this particular visualization wants to display when developers work (see Chapter 4.1). A third case is an id of 0, which is assigned when it is unkown who this person is. We want those in the resulting set, because they are real persons who have worked on the project.

- Line 5

```
GROUP BY project_id, commit_day
```

Finally, the results are grouped by first the project, and then the date, so that we can obtain the number of commits per day for each project.

Other tables

The sample query only uses the `commits` and `vcs_developer` tables. At the time of this writing more than 30 tables are available in the dataset, but these are often clusters of related tables. Below is a list of tables used for the visualizations, grouped by type:

- **Issue** - Tables relating to issues from JIRA, which include user stories and bugfixes. The main table is `issue`, which has an entry for each version of an issue. Fields that are available on this table include the dates the issue was created and updated, who the issue reported and who resolved it, which project and sprint this issue belongs to and the full description. When the issue is a story, the story points are stored.

  Other tables associated with issues that were used for the visualizations are:

  - `comment`, comments that were added to a JIRA issue.
  - `issuetype`, the exact type of an issue, such as Bug, Task or Story.
  - `resolution`, why an issue was closed or resolved.
  - `developer`, names of JIRA users.
  - `project_developer`, per developer an entry for each project they worked on.

- **Sprint** - One table, `sprint`, with basic information from JIRA about each sprint, such as the project it belongs to, the name of the sprint, and when it started or ended.

- **Version Control** - Tables associated with the source control systems. Besides the earlier mentioned `commits` and `vcs_developer` tables, these include:

  - `repo`, a table with the names of all repositories (”sub-projects”), similar to the project table mentioned earlier.
  - `change_path`, this table holds all filenames that are edited in a commit, with the corresponding commit hash and repository of that commit. This can then be matched to the specific commit or project.
  - `tag`, release tags specified in a repository.
  - Other tables specific to some version systems like GitLab, including information about merge requests and events such as pushes of commits. These are not used by the visualizations.

- **Metrics** - These tables include data from the quality dashboard project definition and stored metrics data.

- `metric`, the metric types.
- `metric_value`, data from the quality report, including the exact value of the metric, which rating (such as 'good' or 'perfect') the quality dashboard assigned to that metric and when this item was measured.
- `metric_version`, all versions of a project definition in which changes to target norms of a project were made.
- `metric_target`, manual changes to the metric targets of a project.

Processing

The results of the query from Listing 3.1 are then processed by an R script. R is a programming language used for statistics and data analysis. By matching the `project_id` with the project name from the `project` table, and formatting the date a JSON file is created.

Listing 3.2: JSON file output from R script

```
{
    "ProjectName1": [
        { "day": "2017-01-01", "value": 1 },
        { "day": "2012-01-03", "value": 2 },
        ...
    ],
    "ProjectName2": [
        ...
    ],
    ...
```

The structure of this file is displayed in Listing 3.2. The file is structured as an associative array with each project name as the key. Each project then has an array consisting of day/value pairs. When a project has no commits, the array is empty. Like mentioned earlier, days without commits are omitted because the visualization only highlights those with values and leaves the other days empty.

The visualization then uses this JSON file to display the data it wants, but does not have the knowledge which data sources or database tables were involved in obtaining the data. This provides some flexibility as the visualization only needs to know where the file is located and what its format is, but it is not directly connected to the database.

Build pipeline

The visualizations use Jenkins to periodically reload with the latest data. Jenkins is a tool that can be used for continuous integration (CI) and task automation. On a configurable interval (for example, once per hour) Jenkins builds one or more files of the latest data using the aforementioned queries, and then rebuilds the visualization. This also ensures that when a new version of the visualization code is pushed to the corresponding Git repository, this new version is available for inspection by anyone at ICTU within an hour.

3.3 D3

All created visualizations are web-based. This ensures that they are easily accessible without the need for installing special software, which can aid in the eventual adoption of the visualizations.

The visualizations are written with JavaScript, a programming language that is included with every browser. It is standardized in the ECMAScript specification. For the visualizations the most recent version, ES2015 (formerly known as ES6), is used, but compatibility layers provided by certain libraries (Webpack and Babel) add support for currently incompatible browsers (such as Internet Explorer).

To create the visualizations a JavaScript library called D3.js or simply D3 was used. This widely used library provides similar functions for manipulating the *Document Object Model* (DOM) as for example jQuery, but adds more functions related to data manipulation and the creation of interactive graphs [13]. D3 offers options to use both the HTML canvas element and SVG images. The latter was chosen as SVG is also easily manipulated via the DOM and because it is widely adopted by browsers [14].

With D3 animation and interactivity can be added to a visualization. Using built-in methods it is easy to allow the user to interact with the visualization and change certain aspects of it. In the visualizations created as part of this thesis this is especially visibile in the Network (Chapter 4.2).

Building a Visualization

Each visualization consists of a repository with a static HTML page and Node.js-based JavaScript files. Node.js allows to use JavaScript outside a browser context [15]. Its built-in package manager *npm* [16] makes it easy to install packages like D3.

Building a visualization involves three steps:

1. Building the dependencies using npm, which retrieves all development and runtime dependencies.
2. Bundling the JavaScript files with the data using Webpack. Webpack is a JavaScript tool to package all modules the application needs [17]. Starting with the index.js file it looks up all dependencies and changes modern ES2015 to the older ES5 format were needed. This ensures older browsers which do not

support new features can execute the JavaScript file while enabling the developer to use newer language features. Finally, Webpack bundles some of the JSON files with the compiled JavaScript. Some files are left out because they are loaded on demand when using the visualization.

3. The scripts are published to one `bundle.js` file, which is loaded by the HTML page.

Structure

The structure of the visualizations is largely the same. Listing 3.3 displays a simplified folder structure.

Listing 3.3: Visualization folder structure

```
lib/
    index.js
node_modules/
public/
    data/
    index.html
    bundle.js
package.json
webpack.config.js
Makefile
...
```

The `lib` folder holds all source files. The `index.js` file is the start point of the visualization. The code in this file is loaded first when opening the web page. The required files are fetched and the visualization itself is built. Each visualization has its own components which also live in this folder and which are loaded by the index file.

The `node_modules` folder is a folder generated by the Node.js package system. These dependencies are then included when needed.

The `public` folder is where the actual application lives. This folder needs to be published to make the visualization accessible. It provides a static `index.html` file which loads the needed JavaScript. `bundle.js` is generated by Webpack as described above and is included in the HTML file. The `data` folder is where all JSON files are put by the automatic Jenkins job (or manually when testing locally). In Figure 3.2 an outline of the data flow is shown.

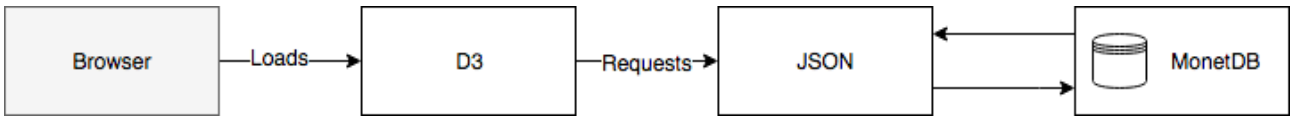


Figure 3.2: Visualization flow chart.

In the root folder there are some generic files like a readme, a `package.json` which Node.js needs and a `webpack.config.js` file which provides some configuration and setup for the Webpack builder. Lastly, to build the project either a `Makefile` is present which can be used, or for some visualizations a custom JavaScript file using the Laravel Mix library [18]. Both work roughly the same and provide scripts to run a local development server and build the project. All resources needed to build a visualization are shown in Figure 3.3.

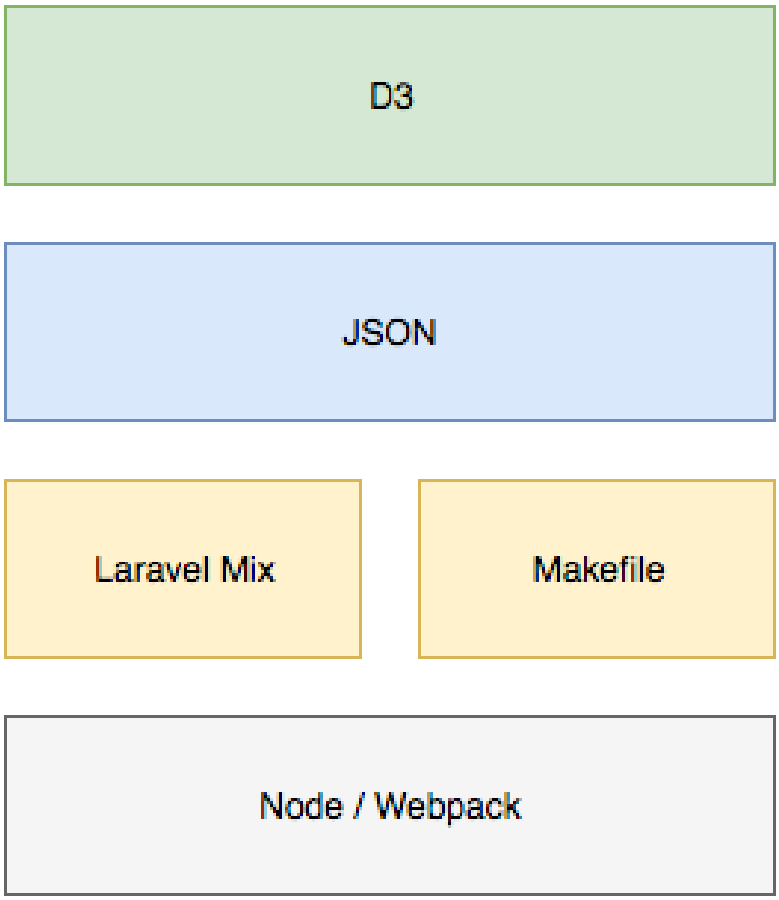


Figure 3.3: Resources needed for a visualization.

Chapter 4

Experiments

In line with the subquestions outlined in Chapter 1 three visualizations were created. The goal for each was to provide valuable insight for people at ICTU. Different stakeholders were targeted and involved during the process:

- Software development team members who work on projects at ICTU. With clear visualizations they can be helped to understand the development process and spot issues with it.
- Management, meaning both the ISR core team and project leaders. With a visualization they can easily see how different projects are doing.
- Researchers such as other people in the GROS project, who can spot anomalies and look for insights.

4.1 Heatmap

Associated research question: *How can external data, such as temperature, be correlated to project activity?*

Background

The dataset as outlined in Chapter 3 consists of a large amount of gathered data, however this is all internal ICTU data. By correlating it to external data (i.e. data not gathered by ICTU) we can see if there is a relationship between the two. An external resource that is easily available is temperature. The Dutch weather forecasting service KNMI provides a dataset [19] which can be used to this end. The idea behind using temperature is that certain temperatures might correlate with the amount of work done. For example, a high temperature can lead to more people working from home, or a low temperature for more traffic problems.

The next step is to choose which data will be used to correlate this temperature with. There are a large number of metrics available, as seen in the previous chapter. A large amount of the data consists of information about commits. A larger commits volume could indicate more work. This is probably an oversimplified way of measuring work amount, but it's one that is easily available from the data. In version control systems like Git commits can be merged ("squashed") together [20], so that only one or a few commits remain for a larger amount of work. It would be expected that it influences the end result. It should be possible to swap this dataset with another in the resulting visualization if it indeed is not reliable.

Design

To find a correlation between the data the primary dimension of the visualization is time. By viewing activity over time, it should be possible to find trends or correlations if there are any. Users could distinguish certain events even when they are not directly shown in the visualization, for example when some team members were not available during a time period.

There are different stakeholders who could use this visualization. First, the project manager and ISR management could use the correlation between days, activity and external data to predict how productivity will be for a certain day, given each of those factors. This depends of course on whether or not a certain correlation is found. Furthermore, they have an overview of which days of the week see the most activity.

Secondly, researchers could use this visualization to more easily find out which datasets correlate with eachother.

The colors are adapted from the Color Brewer library [21]. By using a single hue in the main calendar view and having the brightness indicate how much activity there is on a day, it is also usable for people with color blindness.

Implementation

The visualization created is a heatmap which displays the amount of commits, similar to how websites like GitHub display user contributions [22], with a calendar for each month. While the days are not labeled, sunday is always on top, so it is possible to distinguish weekends and workdays. This is also visible as there are almost no commits during weekends, which is expected. Instead of users the contributions for a project are shown. With a button the view can be switched to another project. The calendar automatically adjusts to only show the years in which this project was active. The initial view is visible in Figure 4.1.

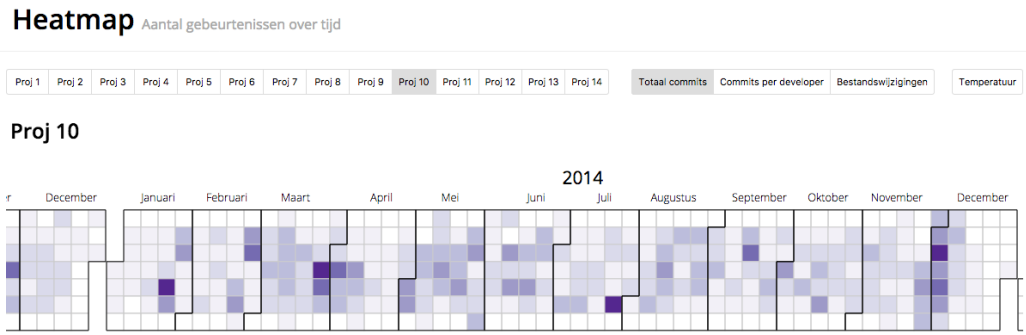


Figure 4.1: Heatmap overview: project buttons in the top left, data switch in the top right.

It's possible to switch the view to commits per developer rather than total commits, see Figure 4.2. The color scale adjusts accordingly.

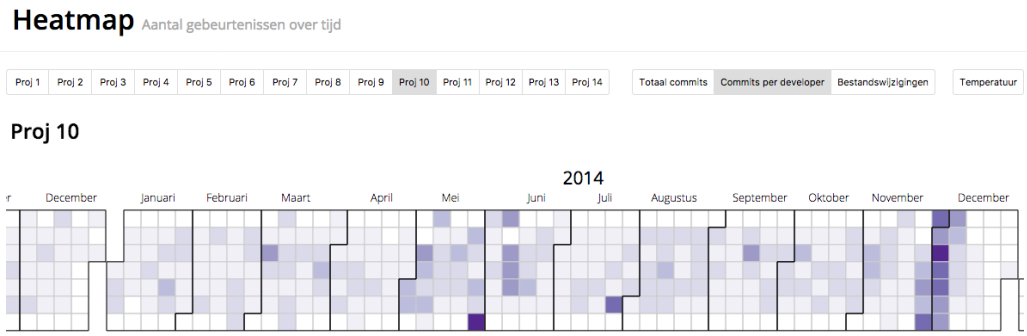


Figure 4.2: Commits per developer instead of total commits.

External data can be toggled with a separate button. Currently this displays the weather data from KNMI like described above. When enabled, the temperature is shown as a small bar inside each day cell, the height depending on the actual temperature (Figure 4.3). It's worth noting that this temperature is from the weather measuring station in The Hague, where ICTU is located.

Hovering the mouse pointer over a day cell opens a tooltip with more information for that date, depending on the view. The number of commits and developers will be displayed, alongside the exact temperature (Figure 4.4).

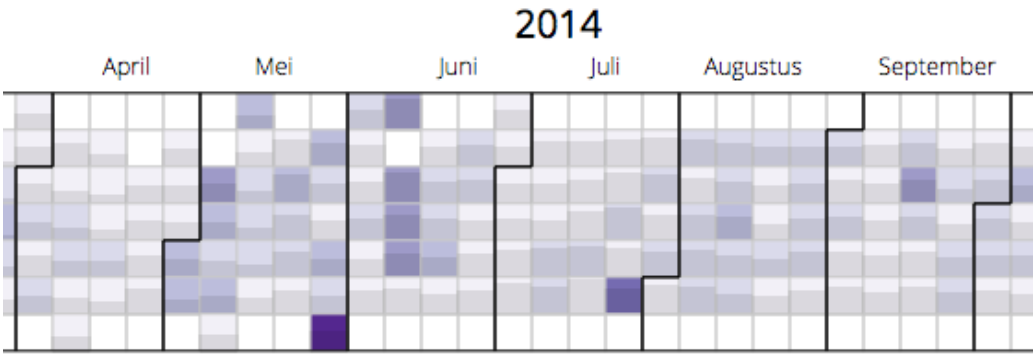


Figure 4.3: Temperature bars on the heatmap.

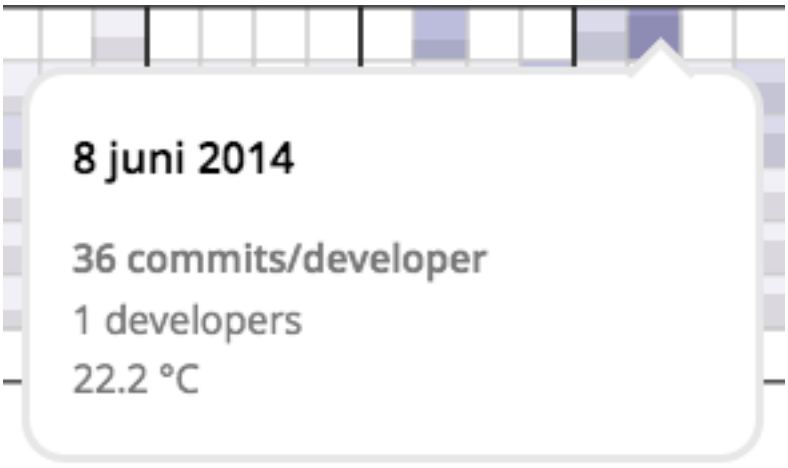


Figure 4.4: A tooltip can be displayed.

Correlation insights

A correlation between the amount of commits and temperature is not really visible. In the summer there are generally fewer commits, but that can be attributed to fewer people working on a project due to vacation. Though the commits per developer should somewhat mitigate that effect, it still does not seem to show a correlation. As mentioned before, swapping the commit data for a different dataset that more accurately measure the productivity may be worthwhile. It might also be possible to extract the amount of team members on vacation from the Timetell tool, but as mentioned in Chapter 3 these are currently not available in the GROS database.

File changes

A third view which is a bit separate from the other data in this chapter is that of ”changed files after a long time”. This came up during a meeting at ICTU where some people mentioned that this could be interesting to visualize. Essentially this shows an overview of files that were left untouched for more than a month, and then edited again. This is part of the same heatmap as the commits and temperature, though external data is

currently not available in this view. It would depend on which external data is used, but it does not seem to make sense to try to correlate temperature with file changes.

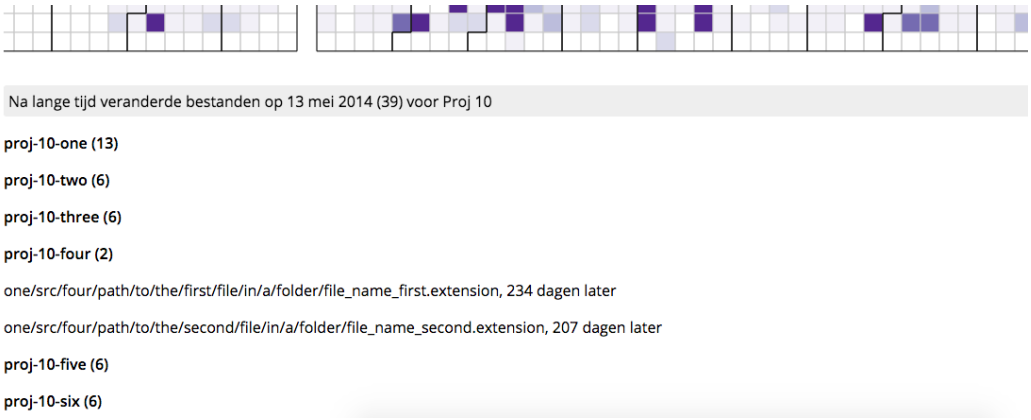


Figure 4.5: File changes overview.

The date indicated on the heatmap is when a file was edited again, with a darker color when there are more files edited on that date. Clicking on a day will open a list with all version control repositories within the current project, with for each repository a list of files. The complete path of the file and after how long it was edited are shown. An interesting thing that becomes visible with this visualization is that some projects have repositories with identical content (the same files are changed at the same time in different repositories). This could be due to one repository being used as a submodule of another repository within the same project.

Feedback

During the development process of the heatmap feedback was gathered from the different stakeholders, and afterwards with the System Usability Scale questionnaire (Appendix A). There were some suggestions for improvement and notions where the visualization does not really work.

What was brought up frequently was the suggestion to use other external data than temperature. As we have seen earlier there does not seem to be a correlation between commits and temperature. The idea that there might be a correlation between weather and activity was not ruled out, but some suggestions were made to use other data. For example, in the Netherlands the weather service gives warnings for bad or extreme weather. Those could be used instead of temperature, as temperature in itself does not really tell if the weather was bad or good. Another option in this regard is a sort of ”weather grade”, similar to what ski slopes use: the weather is graded with a number, which could then be compared to see if the weather was good or better than another day.

Another suggestion for a whole different set of data was to use tickets created or calls to the servicedesk. When people experience problems and these are not solved swiftly, it will probably influence their workrate. Showing this could provide insight in when or whether this happens and which projects are influenced.

The dimension of the heatmap is a calendar view where months and days of the week are visible (sundays on top). It was suggested by a user to use a different dimension. An example of this would be to group it by sprint, as most if not all work happens in sprints. It could be interesting to look into this and to see if there are other correlations to make.

During a meeting with software developers they noted that currently no adjustment is made for vacations. When fewer people work on a project during vacation, the number of commits naturally goes down. We see this during the summer where there are notably fewer commits. The commits per developer could be used but this does not seem to always mitigate the effect, so we could look into other options.

Another point is that the file changes view is quite barebones: only the number of changes is shown in the heatmap, and for each change we show only the full path and after how long the file was edited. This could be expanded to include more related data, such as who edited it this time and who previously edited this file. People on a team can then use this to more easily find out more about the history of a file. This can be especially helpful for teams that still use Subversion.

All respondents of the SUS questionnaire say they find the visualization easy to use, but they also think there is too much inconsistency in the visualization. This could be due to the different modes, commits and file changes. It currently is not clearly explained that the external data (temperature) is not available in the file changes view and why that is. Better explaining the differences between the modes might help users in understanding the visualization, so that it is more valuable to them.

4.2 Network

Associated research question: *How can relations between members of different teams be visualized?*

Background

There are a lot of different teams and projects at ICTU, and many people work at multiple projects. We can see this in the data by coupling users between systems. This needs to be done carefully, as users can use different email addresses between projects or even within the same project, due to different version control system configuration on different computers. By looking at the relationships between people and projects, we can create a visualization where this becomes visible.

For the purpose of this visualization, working on a project is defined as having committed code to one of the project's repositories or being mentioned in the changelog of a JIRA issue. This way we can include all people that are relevant to a software project. This also includes support teams and the Product Owner.

Design

A goal of this visualization is to provide insight into how people at ICTU collaborate on projects. This can be useful for different stakeholders. Team members can easily look up who worked in the past at a certain project and what other projects they work on. ISR management gets an overview of the current active projects, and who works on what project. It also shows relations between projects. A further insight that can be useful is who made the most changes certain projects or on the most projects.

Implementation

To represent the collaboration network an undirected graph was created (Figure 4.6). In this graph, the larger blue nodes represent projects. Each smaller node is a person, with different color codes: Orange colored nodes represent people who are marked as "external", meaning they do not have an email address ending in `@ictu.nl`. These are often people enlisted by the client who are assigned to a project, such as the product owner. Green nodes are developers, given that status based on whether they made commits in a repository for a certain project. When somebody has at least 30 commits over all projects they are involved in, the node is marked green. Lastly, the light blue nodes are people who are neither external nor developer. These are for example people from support teams.

Hovering over a node displays either the project name or that persons name (unless the data is encrypted, it then displays an anonymous hash), and highlights all connected nodes with a red line (Figure 4.7). The visualization is entirely interactive and allows for dragging the different nodes around. This allows to change

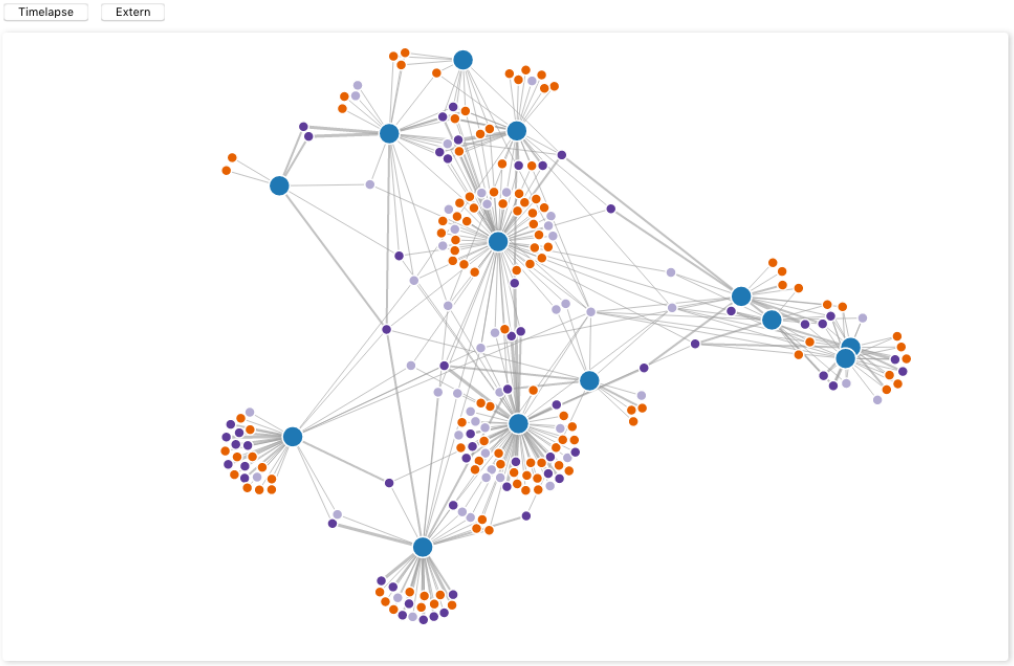


Figure 4.6: Default network overview.

the order in which the nodes are displayed to get a better view of a particular node. When the user stops dragging a node, it gets pulled to the center of the visualization so that it stays compact.

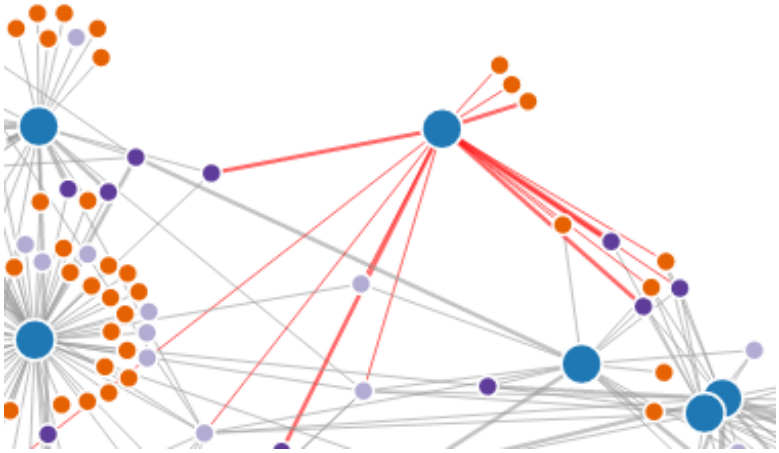


Figure 4.7: A node is dragged around with all connected lines highlighted.

With a button it is possible to filter out all external people, resulting in a graph with only internal ICTU employees remaining (Figure 4.8). It would be easy to also implement such a button for other roles like developer.

Nodes are placed in the visualization using the force layout algorithm of D3, first described by Tim Dwyer [23]. It combines a physical simulation with a constraint-satisfaction method. Currently the weights are the same for all project nodes a person node is connected to, so the nodes are placed somewhere in the middle between

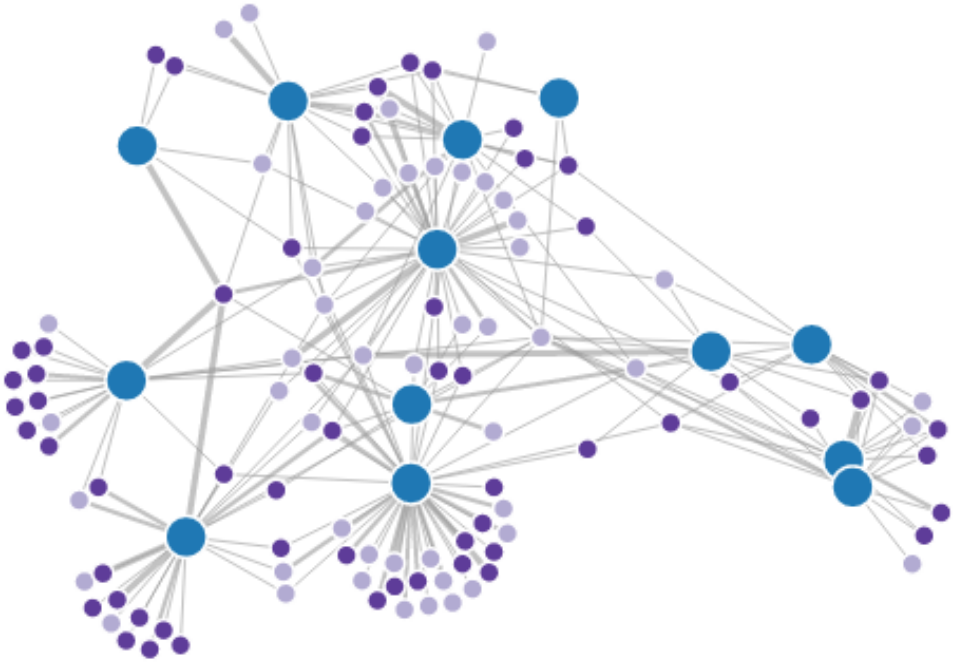


Figure 4.8: Network without people marked "external".

nodes. This could be changed so that nodes are placed closer to certain other nodes depending on whether a person was more involved with that project, using the aforementioned algorithm.

Timelapse

A second button leads to a timelapse mode. This mode makes the growth of ICTU visible. For each month that is available in the data the graph is shown (Figure 4.9). People who have activity in that month (either in JIRA or by committing code) are immediately added to the graph. When a person does not have any activity for three months, the node is removed. This ensures the changes are not too frequent and chaotic when people do not work at a project for a short time.

Currently the timelapse displays the activity per month. Another option would be to have a different time frame such as sprints, quarters or years. The way the visualization is set up makes this possible, with maybe a slight change needed to the display of the date. Simply changing the data to use a larger timeframe would already work.

In the timelapse mode currently only external people are marked (again with an orange color), but in the future it could also mark other nodes similar to the default graph. The timelapse also has some simple controls to pause it or decrease or increase the speed.

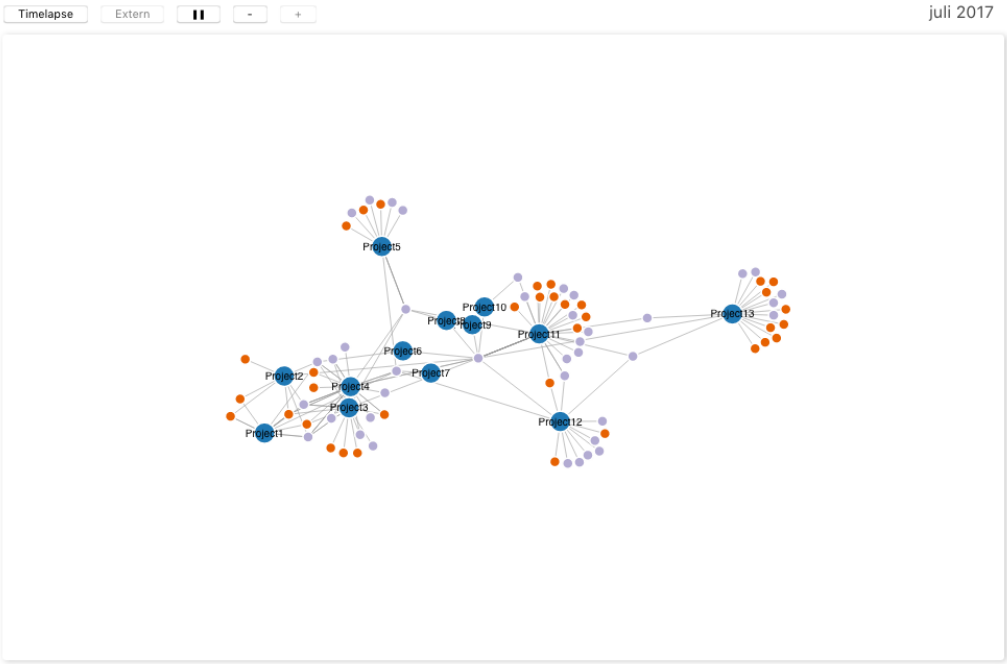


Figure 4.9: Timelapse view of June 2017.

Feedback

During the development process of the network graph feedback was gathered from different stakeholders, and afterwards with the System Usability Scale questionnaire (Appendix A).

One thing that most people noted was that the timelapse view did not remove inactive nodes (people that left a project and did not work on it anymore). It was mentioned that they expected that this would happen. This feature was subsequently implemented, as the default network shows the graph for all data, and the timelapse now only shows the state for a certain month, so the two views serve different purposes.

The timelapse has only controls to start or stop time, and can only restart to the beginning. It would be useful to have more control over the time. For example a slider could be used which allows the user to skip to a certain point in time to show that particular month, or even skip to the end to see the current state of the network.

To make it easier to find people and to filter the data, it was suggested to add a search field. This field could then search for specific people's name or filter on role, similar to what the "External" button does. Another thing to note here is that currently are only three "roles": external, developer and 'other'. This could be expanded with more roles, such as 'support team'. This is currently not implemented because these roles can not be extracted from the GROS database. It could be possible to get this from another source, for example an HR system. Otherwise manually creating a list with roles and people belonging to them would be a possiblity. This would also prevent people from being mislabeled, like someone who has a certain number of commits but is not actually a developer, or external people using an internal email address or vice versa.

It is also not immediately obvious to most users what the two buttons do. The 'Extern' button might need a

better label, such as a descriptive 'Filter all external people out', though this would still not explain why a user is marked external. This would need to be explained regardless, though it could of course be added somewhere else. Furthermore, a better explanation of the timelapse mode could be added, such as the timeframe used.

To see how much work a person did on a project, the total commits count is used as a weight to make the line connecting the person and project node thicker. A different datasource could be used, an example that was mentioned during a meeting was story points. The weight of a story might tell more about the actual workrate (in time) than the number of commits. Another possible measure here is the size of commits (insertions/deletions), though this could be skewed by large refactors. These metrics only apply to developers though, so it may be worthwhile to research if there are other applicable metrics. For example, the number of JIRA issues a user participated in is available.

A user noted that there is currently no legend for the meaning of each color in the visualization. As explained earlier this is an important part to help the user in understanding the visualization and finding value in it [8]. This also follows from the SUS questionnaire, where many users answer that they think the visualization is easy to use, but most of them also think they would need assistance in using it and thought there is too much inconsistancy. For future work adding a legend would be one of the most important additions to make to the graph.

It was also suggested to normalize the model as each team and project works differently. Different ways of measuring work could be used, but they should be presented in a similar manner to provide easy comparison between people and projects.

Users disagree on whether they want to use the current visualization frequently. Some users said they do not see a direct use for it as it only shows the current or historical size of teams. Others said they might use it given some modifications at outlined before were made. The SUS questionnaire shows this divide too - it is possible that users answered the question based on their expectations of improvement. A suggestion to improve this is to take the network in a whole new direction, by using the code of a project as data instead persons. The network could make clear which components depend on each other, and when new components are added to a project. Furthermore, it could show which parts of a system are stable and which parts are edited frequently, with for example the timelapse. This would provide value to both project managers and software developers, to gain insight into the organization of a software product and how it is developed.

4.3 Timeline

Associated research question: *How can velocity in a burn down chart be better displayed in a timeline graph?*

Background

The timeline is an existing visualization created as part of the GROS project. It displays different types of events from Scrum sprints over time. It also makes the durations of sprints visible. This way patterns between sprints or even projects could become visible.

Current events that can be displayed are:

- **Sprint start & end**. Simply shows when particular sprints start or end. These also define the ”sprint blocks” that are displayed on the timeline.
- **Red metrics**. These are red metrics from the ICTU quality system. Each time the quality dashboard is built an history file is generated which holds amongs others the number of red metrics. Only metrics that are 'below the norm' (red) for a longer period of time are shown.
- **Rank change**. When the rank of a user story changes. 'Rank' is the priority in the product backlog that is assigned in JIRA by the Product Owner to a user story.
- **Story point change**. If the amount of points assigned to a user story changes during a sprint, an exceptional circumstance. This can also happen when a user story is 'partially done'.
- **User story**. These are other changes than rank changes, and refer to the end date of a user story. An event is created when the user story is done. Also available in the data is the date when the story was started (in progress), but this is not visible in the timeline.
- **Impediment**. An impediment is a problem the team encounters that influences their ability to finish the sprint or a user story, that cannot be solved by the team itself. For example a user story that is not clear enough and which the client needs to clarify. Currently impediments are loaded from JIRA. A slight problem with this is that sometimes other problems are marked as an impediment in JIRA while the team could solve them themselves.

The timeline displays project on the y-axis and time on the x-axis (Figure 4.10). Each sprint is a block with a colored background and clear boundaries. Different events can be toggled on or off, by default red metrics, changes in user stories and impediments are shown.

By clicking on a sprint a details view opens with more information about that particular sprint (Figure 4.11). It shows all events for the sprint, with in a second bar all commits. This makes it possible to easily see if there is a correlation between the commit and other events.

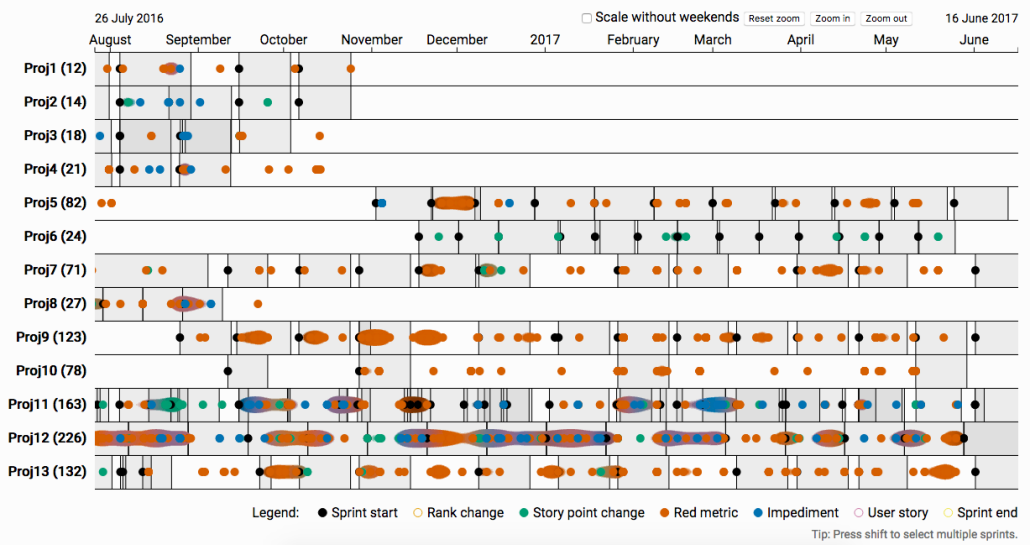


Figure 4.10: Default timeline view. Projects on y-axis, time on x-axis.

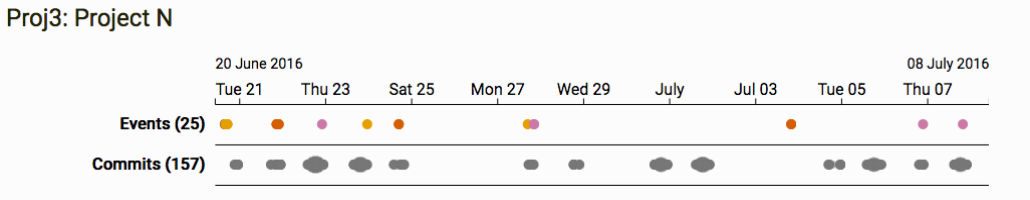


Figure 4.11: Default timeline view. Projects on y-axis, time on x-axis.

When hovering the mouse over a certain sprint in the project overview, a popup opens with more information about that particular sprint (Figure 4.12). Much of the available data as outlined in Chapter 3 are available here. Examples include the number of JIRA developers in a sprint, the number of VCS developers (people that made commits), total amount of events such as impediments and the velocity. Velocity is a measure of the productivity during a sprint, measured by the total number of finished story points per time unit (usually work days). The current format of only displaying the name of the metric and the corresponding value is not very intuitive and makes comparing difficult.

Design

The primary goal for the timeline is make it easy to see how each sprint or project is going, and to make comparisons between sprints and projects easy. To answer the research question, we take a further look at velocity and add a burn down chart to the timeline. A burn down chart is often used by Scrum teams and displays the progress of a team during a sprint. The x-axis displays time, ranging from the start to the end of the sprint. The y-axis displays story points and ranges from zero to the total amount of story points in a sprint. For each time a story is finished, a data point is added with for the new total number of story points. These points are then connected by a line which creates a descending trend. There is a second linear line which shows the 'ideal' or 'perfect' slope. For the 'ideal' line it is assumed that each day of a sprint the same amount of work is done.

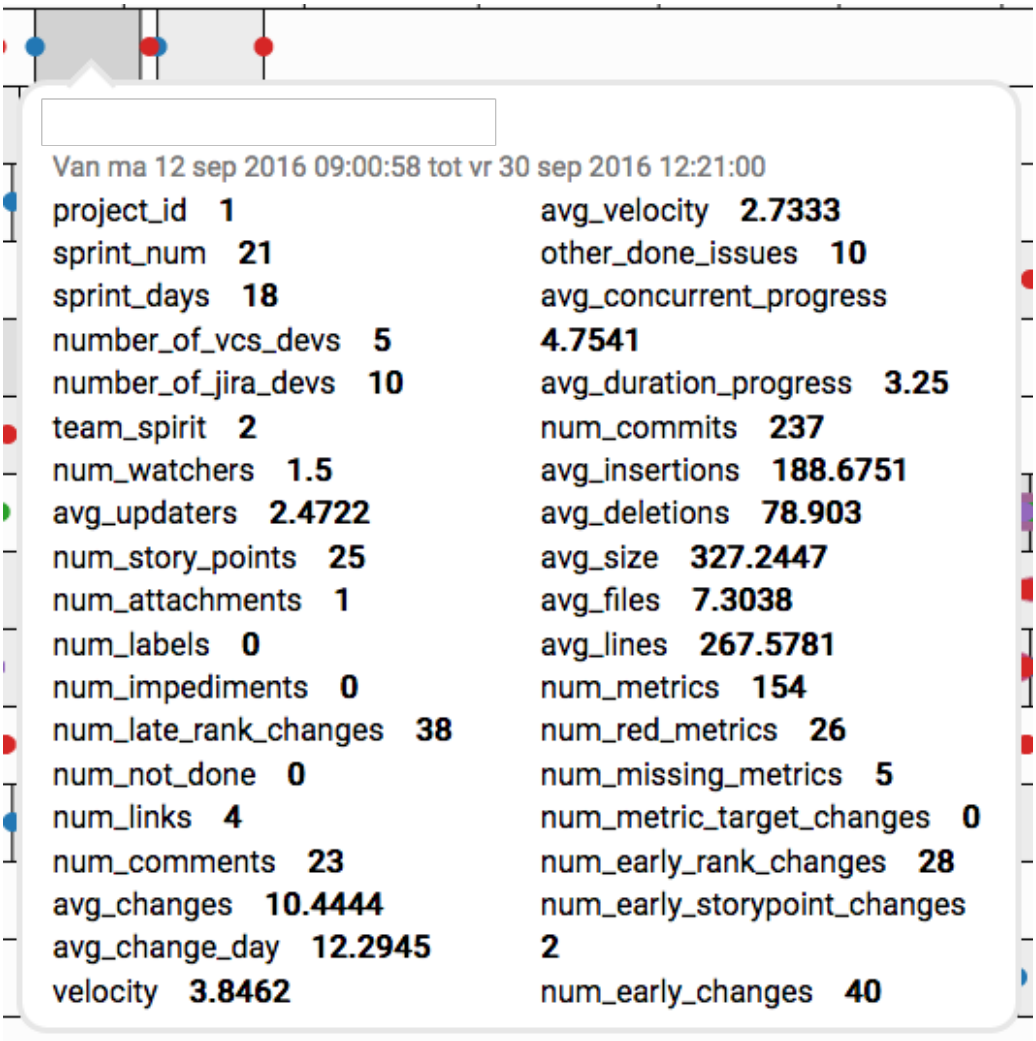


Figure 4.12: Details of a sprint.

While the actual line is never the same as this perfect line, it shows if the team is under- or overperforming. When the data points are constantly above the line, the team can expect to not finish al story points before the end of the sprint.

The timeline can be interesting for researchers and management to find anomalies in the sprint data. These could then be further analyzed.

Implementation

A burn down chart was added to the sprint information (when clicking on a sprint). It scales to the same timeframe as the sprint. By having both the other events and the burn down in the same view, a user can compare the two. The user can hover the mouse over a data point in the timeline to view the exact amount of story points left at that moment and the time at which the change occurred.

Furthermore, the timeline was improved by several other additions. A feature was added where by clicking on a sprint an overlay opens with the information about that sprint. Previously, it would be added beneath the

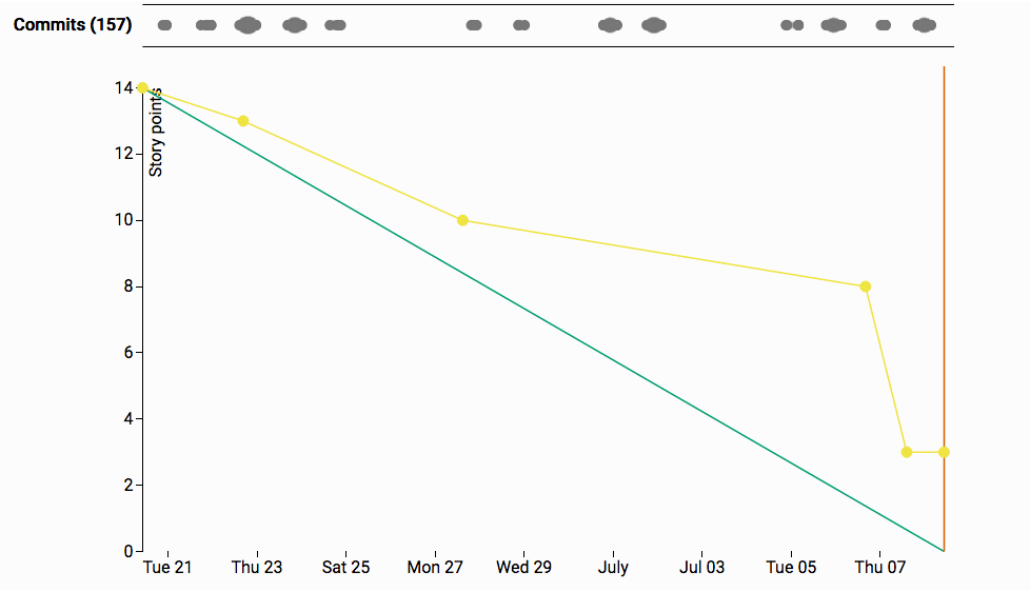


Figure 4.13: A burn down chart for a particular sprint.

whole chart which required scrolling to the bottom, and then to the top again when one was to pick another sprint. It is possible to use this older method, but it was enhanced so the view automatically scrolls to the bottom of the chart to show the details of the sprints. A benefit of this is that the user does not have to scroll down themselves. The main drawback is that this cannot be used when selecting multiple sprints (see below), because it's not possible to know when the user is done selecting more sprints. The same problem exists for the overlay, so a button was added where the user can manually open the overlay. With the overlay the user does not have to scroll at all, as it can just be closed by clicking.

Also, a feature was added for selecting multiple sprints by holding shift (Figure 4.14). It is important to provide tools for the selection of a desired subset, as Keim et al. note [24]. The multiple select feature allows for easy comparison between sprints of the same project, which then all show in the same view. The burn down chart displays a red line for the end of each sprint to make the sprint boundaries clearer. The view is reset when clicking on a single sprint without holding shift, or by selecting sprints of a different project.

Insights

By looking at the burn down charts it becomes clear that for many sprints not all story points are finished at the end of the sprint. Normally the team makes a commitment to finish all story points, so it's interesting that this happens so often. To find out what is really happening further analysis is needed. A possibility would be that the changes to story points are not correctly registered in JIRA by the team (meaning the user stories are finished but not registered as such, until after the sprint ends).

Another thing to note is that there are a lot of red metrics visibile in the timeline. These do not necessarily indicate that a project is headed in the wrong direction, but can also indicate that some metrics have set thresholds that do not align with the quality standards that the team uses in practice. A metric can also be red

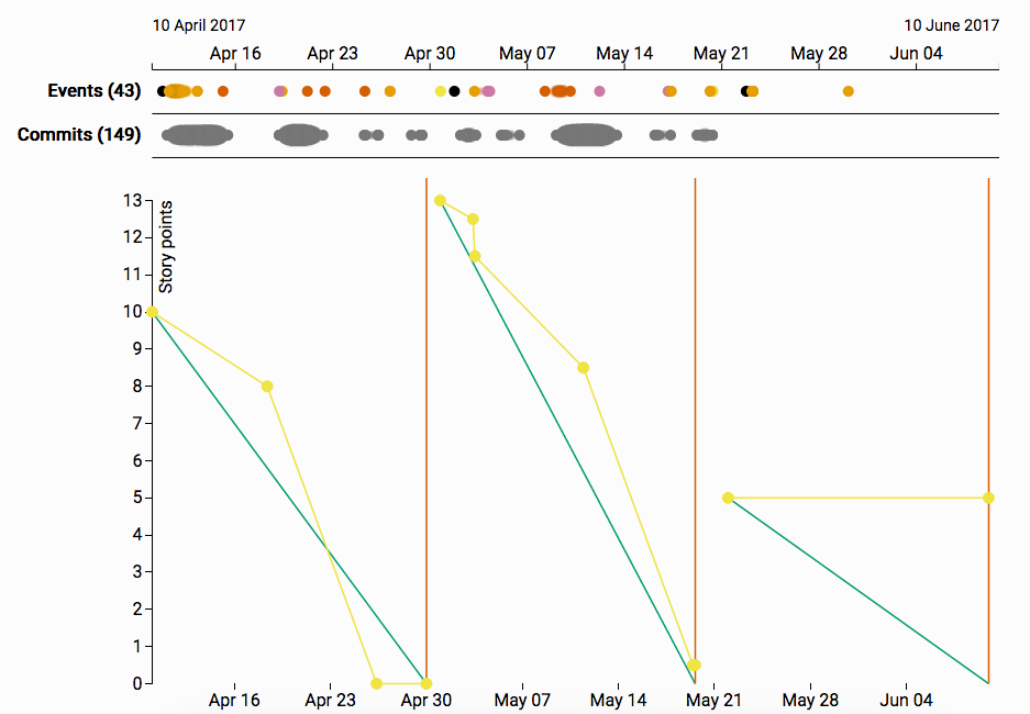


Figure 4.14: Multiple sprints selected.

for other reasons such as incorrect administration in JIRA.

Feedback

As the timeline visualization already existed, some feedback was already gathered and implemented before the research of this thesis. More feedback, including the System Usability Scale questionnaire (Appendix A), was gathered to provide insight into the different additions and changes to the timeline.

From the results of the SUS questionnaire it follows that users are divided on most issues. Some would use the visualization frequently while others strongly disagree. This might be due to the users not understanding the goal of the visualization and value for them. Some users said they did not understand at first that the legend was clickable and that they had control over which events were shown.

Another common problem users encounter with the timeline is the scroll behavior. When clicking on a certain sprint, the user needs to scroll down past the other sprints. But when the user use the scrollwheel or, when on a laptop with a touchpad, a scroll gesture inside the main area, that are will zoom in or out. This can be very annoying as the after clicking a sprint, the mouse needs to move to the side of the visualization to be able to scroll to the bottom. This was one of the reasons for the implementation of the overlay, which does not require scrolling. It does however obstruct the view of the other sprints. It might be worthwhile to look into another solution where for example the application makes an educated guess whether the user wants to zoom or to scroll. Zooming can still happen with the buttons at the top, but to get at the bottom scrolling is the most straightforward way.

Earlier versions of the timeline used a default colorscheme of D3. Because it used amongst others a combination of green and red, these colors were not easy distinguishable for the color blind. Therefore, a new color scheme of Bang Wong was implemented [9]. These colors are distinguishable even for people with color blindness.

Lastly, many users do not understand what the red metrics mean. While most them are used to the quality reports, it was not clear which metrics are shown, and why they are red. Not every metric is as important to every team. Providing more insight into this could result in a better of the red metrics in general.

Chapter 5

Conclusion and Discussion

In this chapter, we briefly discuss the results of the experiments and then answer the subquestions, and finally answer the research question. Lastly, some ideas for future research are mentioned.

5.1 Conclusions

How can external data, such as temperature, be correlated to project activity?

Using a heatmap with a calendar view external data can be compared to project activity, measured by commits. Though no correlation was found between those metrics, changing either or both of them to another metric might yield different results. Using a different dimension than a calendar such as the sprint can be more useful to identify patterns or anomalies.

The file changes view provides an insight into which files are changed after a long time. While it currently only shows the file name, more information could be added that would make it more useful, such as the name of the person that last changed the file. By doing this the visualization could help in understanding the history of certain files.

Users find the visualization easy to use but also inconsistent. Better explaining the different modes can help them in better understanding the purpose of the visualization.

Concluding, this visualization can help in identifying patterns or correlation, but is not useful for that in the current form with the current data. Changes to those two parts can help in making this visualization more useful.

How can relations between members of different teams be visualized?

A network graph which contains nodes for projects and people can show the relations between members of different teams. With a timelapse mode the state of projects at ICTU for each month displayed. Removing inactive nodes was important to make this more useful. The timelapse can be used to show the growth of ICTU as a whole, and is already used for that reason in presentations for clients. As a sidenote, only projects that partake in the GROS project are added, so this should be kept in mind.

Currently three 'roles' are distinguished: "external", "developers" and "other". More roles could be added, such as support team or branch management, but this needs to be done manually as these cannot be derived from the data. Adding more roles could provide more insight into which different roles and people work together on projects.

The visualization shows a thicker line when a person has more commits for a certain project. To make it more clear which people work more on every project other metrics can be used. Commits only apply to developers, so a metric that also works for other roles could provide more insight. Another option is to use a whole other dataset such as the project code. Some users would find this more insightful than the current form.

The usability needs to be improved by adding better explanations for the user. Part of this would be a clear legend and information about the different buttons.

Concluding, the network graph provides an overview of projects and team members at ICTU and can help in finding which people work more on certain projects. The timelapse view shows the growth of the organization and displays for each month the state of the network. Improvements can be made to the usability, to make it easier to find certain people and to make projects contributions more clear.

How can velocity in a burn down chart be better displayed in a timeline graph?

The timeline is a visualization that shows Scrum sprints over time. For each sprint different events like user stories, red metrics and impediments are shown. It can be used to indentify problems in the development process for certain teams, and to compare sprints of different teams. In a tooltip more information about a sprint is available, but the contents are not formatted in an intuitive way. An important metric is velocity, essentially the amount of work a team finished during a sprint.

By adding a burn down chart to the detailed sprint information the velocity during a sprint can be easily compared to other events, though the story point changes often do not correlate with other events. The burn down shows that teams often do not finish all user stories in a sprint. To find the reason for this further research would be needed.

Other improvements to the timeline include the ability to select multiple sprints which allows for easy comparison, and to display the sprint information in an overlay instead of at the bottom of the graph. The color was adapted to be color blindness friendly.

The usability of the timeline can be improved more. Scrolling is sometimes problematic, and users want more explanation about the red metrics. The popup with sprint information is not very structured and has too much information.

Concluding, the burn down chart provides more information into how teams work on user stories. By adding it to the existing detailed sprint information it can be compared to other events. Lastly, other improvements make working with the timeline easier, such as an overlay with sprint details and the ability to select multiple sprints, but more improvements to usability need to be made.

Conclusion

Recall that the research question for this thesis is *"What are good visualizations for trends in software development using Scrum?"*.

In general, visualizations can help in identifying patterns in a dataset, but as we have seen, the specific data and measure need to be chosen carefully. Otherwise, the user might not find any value in the visualization, or a potential correlation is missed. Changing the existing visualizations with other data can create much more value for the stakeholders.

The current datasets of the heatmap did not show a correlation but using different data might change that. Comparing external data with internal data thus could be an interesting way to broaden the scope of research into identifying patterns in Scrum software development.

Usability is one of the most important aspects of a good visualization. Clear explanations to the user are needed so they understand a visualization quickly and can determine whether it has value for them. Simple additions such as a legend could make parts of the visualization much clearer. Attention should be given to the color choice to account for color blindness.

Researchers can use these visualizations to identify anomalies in the dataset. Management can use visualizations such as the heatmap to find patterns in the data. Software developers can use them to find where changes in the process are needed. Overall, these visualizations can help the different stakeholders in supporting their work, given that they have the right data, are clearly explained and have a good usability.

5.2 Future Work

The visualizations in this thesis primarily provide overviews of different parts of the dataset. For ICTU these visualizations help in understanding how the development process went in the past. A direction for future research could be to implement a sort of anomaly detection. A visualization would mark interesting or "deviating" data to help identifying problems.

Another interesting thing to look at would be to find out when there are faults in the developed software, and to correlate that to other items in the dataset. Similar to that, a detection whether faults have been prevented, for example in a new release. Software errors are very expensive for companies and handling them better by providing visual insight, and possibly even preventing them would be a great help.

Bibliography

[1] ICTU.nl. https://www.ictu.nl/about-us. Accessed: 2017-05-21.

[2] Projecten - ICTU. https://www.ictu.nl/projecten. Accessed: 2017-06-08.

[3] Agile Manifesto. http://agilemanifesto.org/. Accessed: 2017-06-07.

[4] What is Scrum? An Agile Framework for Completing Complex Projects - Scrum Alliance. https://www.scrumalliance.org/why-scrum. Accessed: 2017-06-07.

[5] Ben Shneiderman. Inventing discovery tools: Combining information visualization with data mining. *Information visualization*, 1(1):5–12, 2002.

[6] GitHub - ICTU/quality-report. https://github.com/ICTU/quality-report. Accessed: 2017-06-08.

[7] SonarQube. https://www.sonarqube.org/. Accessed: 2017-06-17.

[8] Jeffrey Heer, Frank Van Ham, Sheelagh Carpendale, Chris Weaver, and Petra Isenberg. Creation and collaboration: Engaging new audiences for information visualization. In *Information Visualization*, Lecture Notes in Computer Science, pages 92–133. Springer, 2008.

[9] Bang Wong. Points of view: Color blindness. *nature methods*, 8(6):441–441, 2011.

[10] John Brooke et al. SUS-A quick and dirty usability scale. *Usability evaluation in industry*, 189(194):4–7, 1996.

[11] Stratos Idreos, Fabian Groffen, Niels Nes, Stefan Manegold, Sjoerd Mullender, Martin Kersten, et al. MonetDB: Two decades of research in column-oriented database architectures. *A Quarterly Bulletin of the IEEE Computer Society Technical Committee on Database Engineering*, 35(1):40–45, 2012.

[12] Peter A Boncz, Martin L Kersten, and Stefan Manegold. Breaking the memory wall in MonetDB. *Communications of the ACM*, 51(12):77–85, 2008.

[13] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. D3: Data-driven documents. *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)*, 2011.

[14] Can I Use - support tables for HTML5, CSS3 etc. http://caniuse.com/#search=svg. Accessed: 2017-06-01.

[15] Node.js. https://nodejs.org/en/. Accessed: 2017-07-12.

[16] NPM. https://www.npmjs.com/. Accessed: 2017-07-12.

[17] Webpack. https://webpack.js.org/concepts/. Accessed: 2017-07-12.

[18] GitHub - Laravel Mix. https://github.com/JeffreyWay/laravel-mix. Accessed: 2017-07-12.

[19] Daggegevens van het weer in Nederland. http://projects.knmi.nl/klimatologie/daggegevens/selectie.cgi. Accessed: 2017-06-07.

[20] Git - rewriting history. https://git-scm.com/book/en/v2/Git-Tools-Rewriting-History. Accessed: 2017-06-07.

[21] Mark Harrower and Cynthia A Brewer. Colorbrewer. org: an online tool for selecting colour schemes for maps. *The Cartographic Journal*, 40(1):27–37, 2003.

[22] Github blog: 'Introducing contributions'. https://github.com/blog/1360-introducing-contributions. Accessed: 2017-06-07.

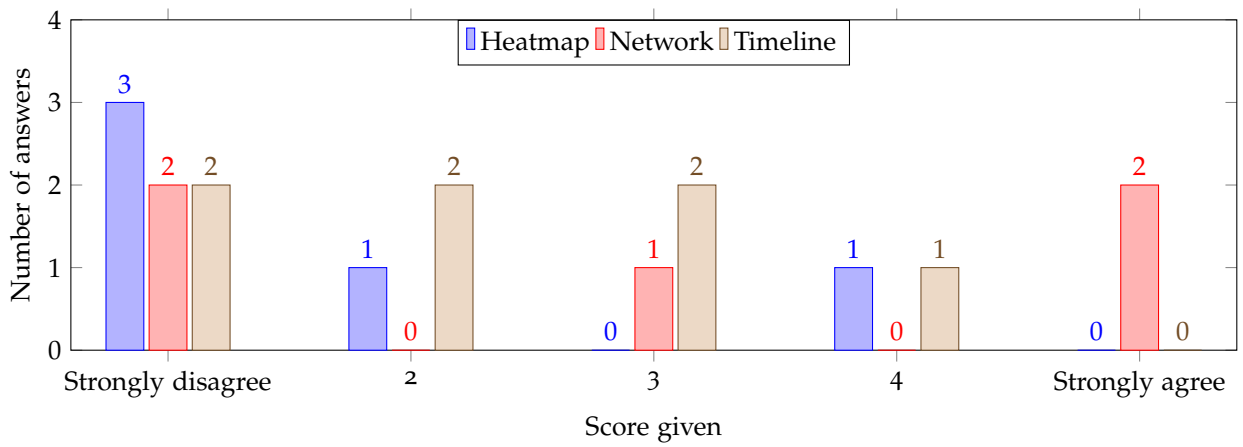
[23] Tim Dwyer. Scalable, versatile and simple constrained graph layout. In *Computer Graphics Forum*, volume 28, pages 991–998. Wiley Online Library, 2009.

[24] Daniel A Keim. Information visualization and visual data mining. *IEEE transactions on Visualization and Computer Graphics*, 8(1):1–8, 2002.

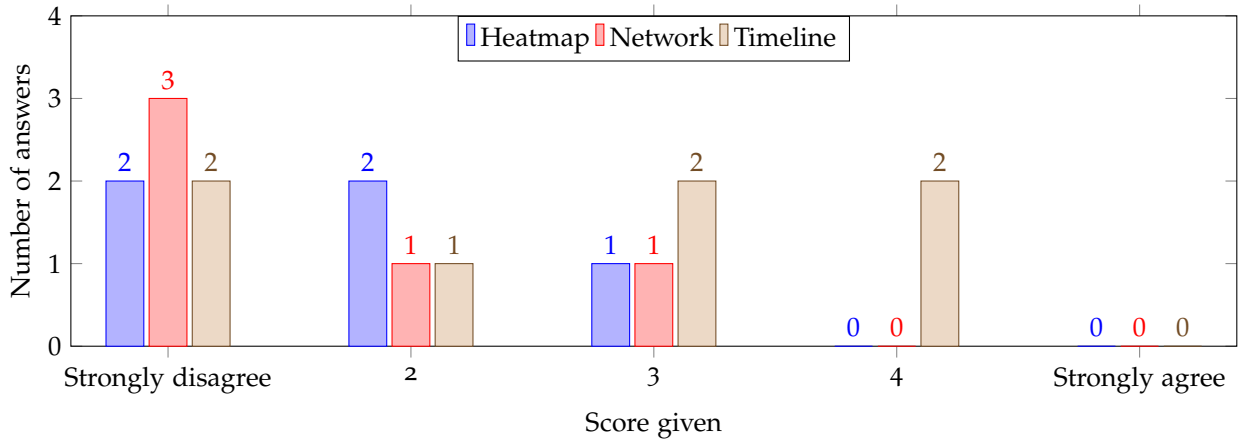
Appendix A

System Usability Scale questionnaire

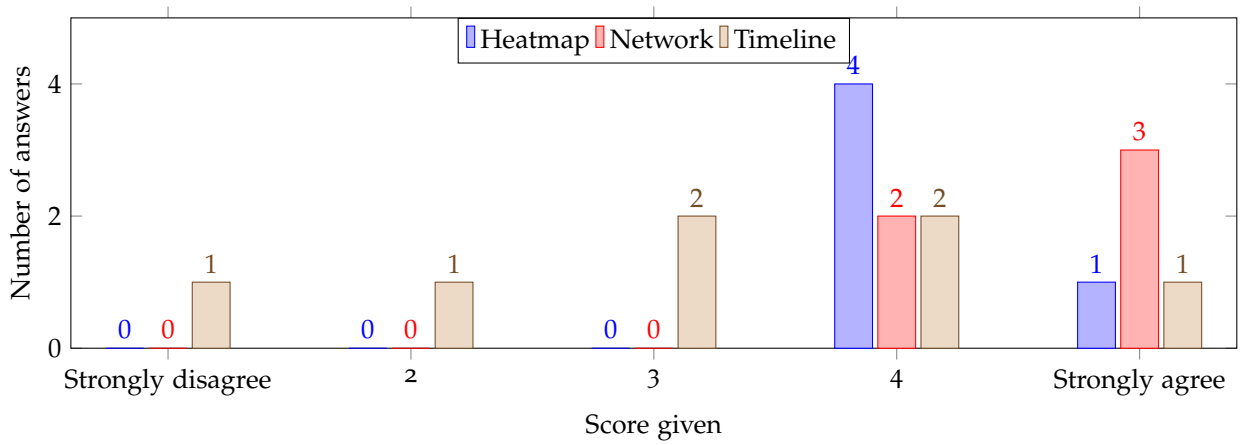
I think that I would like to use this visualization frequently.



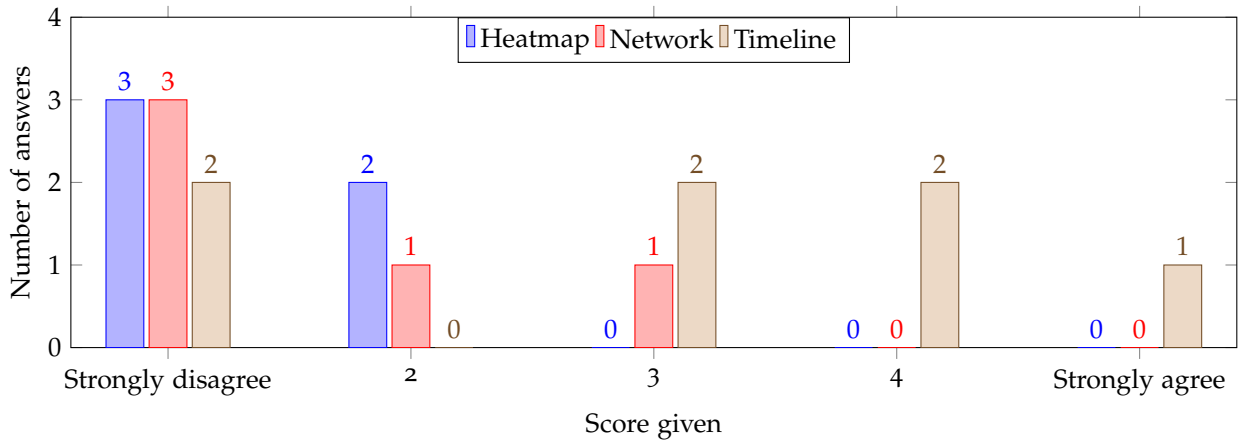
I found this visualization unnecessarily complex.



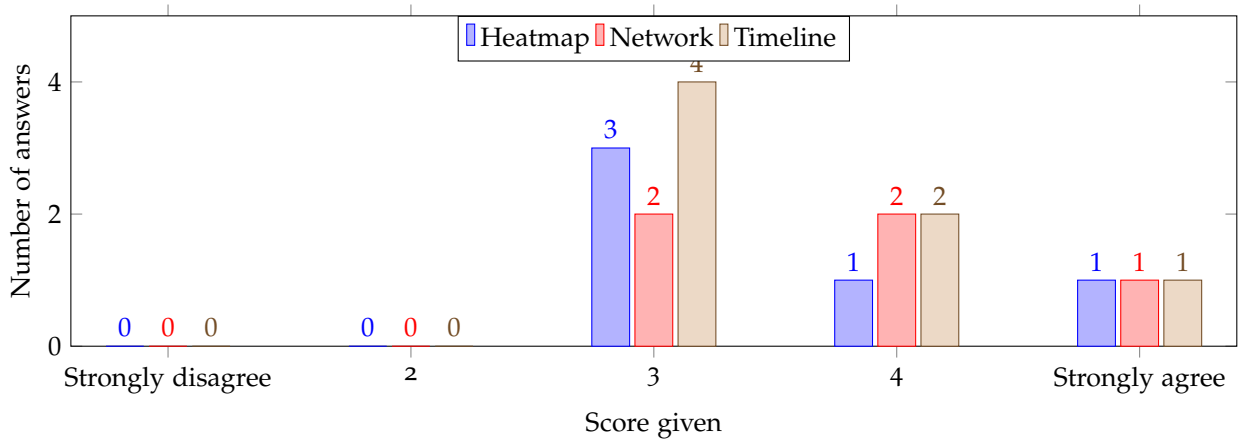
I thought this visualization was easy to use.



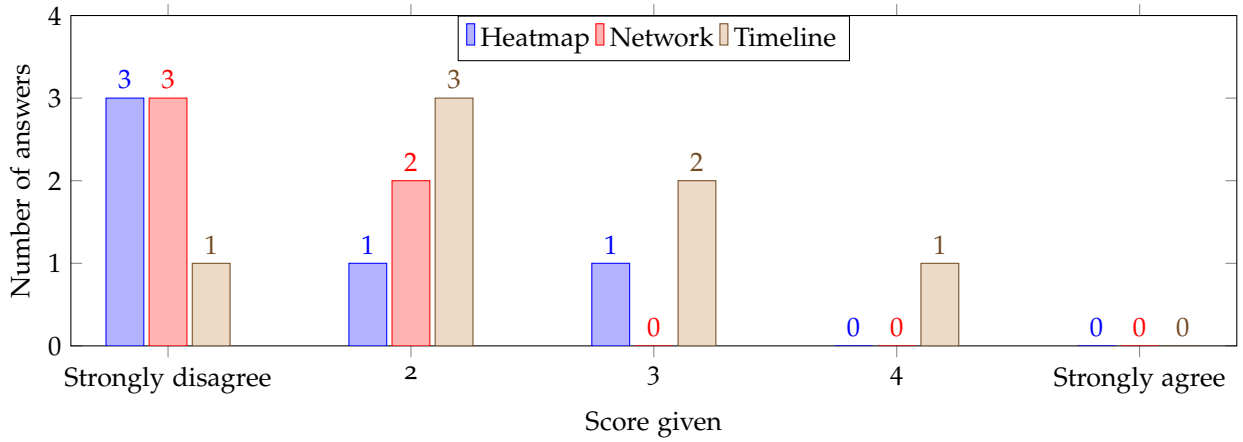
I think that I would need assistance to be able to use this visualization.



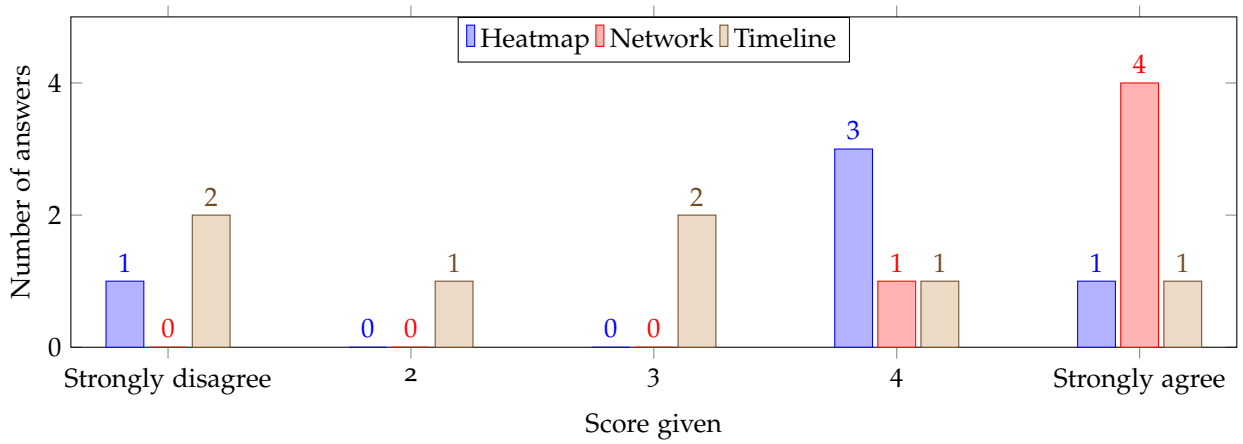
I found the various functions in this visualization were well integrated.



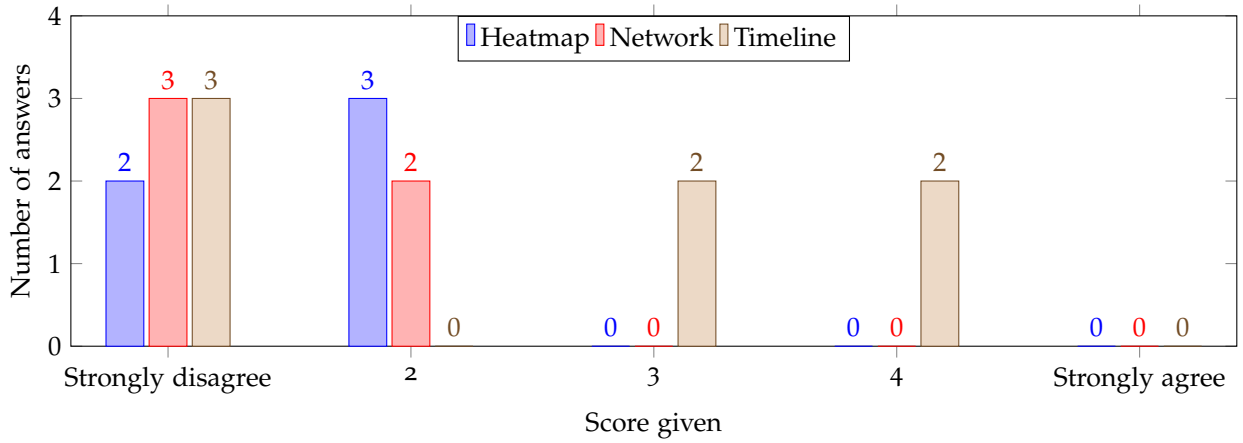
I thought there was too much inconsistency in this visualization.



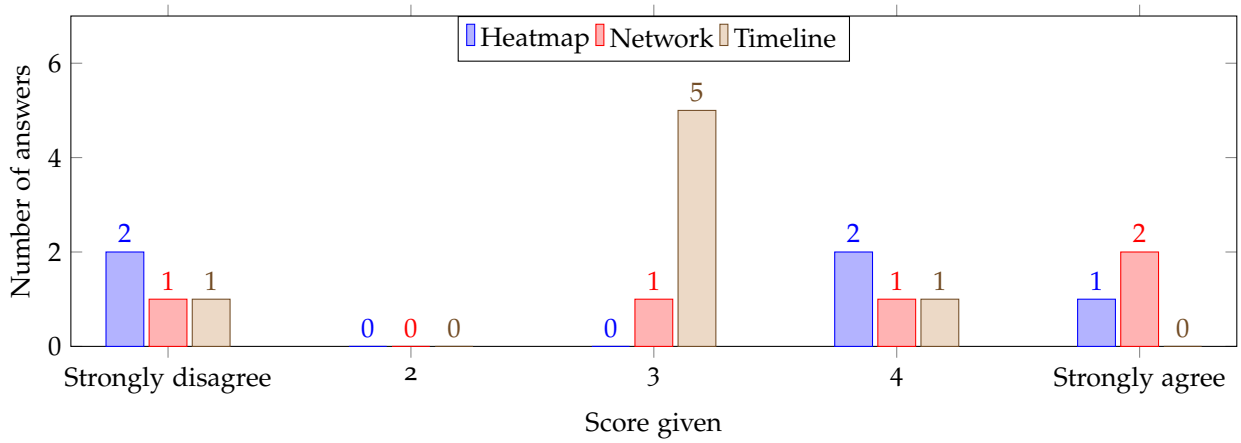
I would imagine that most people would learn to use this visualization very quickly.



I found this visualization very cumbersome/awkward to use.



I felt very confident using this visualization.



I needed to learn a lot of things before I could get going with this visualization.