Master Thesis

April 28, 2014

SQA-Timeline

A Timeline-Based Visualisation Approach for Software Evolution Data

Jens Birchler

of Zurich, Switzerland (08-707-929)

supervised by

Prof. Dr. Harald C. Gall

Martin Brandtner

University of Zurich^UZH

s.e.a.l. software evolution & architecture lab

Master Thesis

SQA-Timeline

A Timeline-Based Visualisation Approach for Software Evolution Data

Jens Birchler

University of Zurich UZH

s.e.a.l. software evolution & architecture lab

Master Thesis
Author: Jens Birchler, jens.birchler@gmail.com
Project period: 29.10.2013 - 29.04.2014

Software Evolution & Architecture Lab
Department of Informatics, University of Zurich

Acknowledgements

I would like to thank Prof. Dr. Harald Gall for the opportunity to write my thesis at the Software Evolution & Architecture Lab at the University of Zurich. Furthermore, I would like to thank Martin Brandtner for his assistance and valuable feedback. Last but not least I am grateful for the excellent collaboration with Stefan Hiltebrand who was responsible for the back end development.

Abstract

Stakeholders of modern software engineering environments rely on a variety of tools which support the development process. Examples are source code repositories and issue trackers as well as tools to support practices such as Continuous Integration (CI) and Software Quality Assurance (SQA). Considering the available data is scattered across multiple tools featuring different user interfaces, it can be difficult for a stakeholder to maintain awareness of the projects event history. To assess the cause of certain events such as build failures or metric changes, stakeholders require insights into preceding events in context of time. An approach featuring a chronological visualisation of software evolution data called SQA-Timeline was implemented in this thesis. The tool visualises various event types and metrics on a timeline. With its drill-down functionality it offers different information granularities from an overview down to a source code view. A user study was conducted to find out how the approach can foster the understanding of software evolution. The results showed that the participants using SQA-Timeline solved certain tasks with higher correctness and efficiency than users which had a set of commonly used tools at their disposal.

Zusammenfassung

Stakeholder von modernen Softwareentwicklungsprozessen verlassen sich auf eine Vielzahl von Werkzeugen, welche den Entwicklungsprozess unterstützen. Beispiele dafür sind Source Code Repositories und Issue Tracker, sowie Werkzeuge, welche Methoden wie Continuous Integration (CI) und Software Quality Assurance (SQA) unterstützen. Da die verfügbaren Daten über mehrere Werkzeuge mit verschiedenen Benutzeroberflächen verteilt sind, kann es für einen Stakeholder schwierig sein, die Projekt-Awareness bezüglich der vorgefallenen Ereignisse aufrecht zu erhalten. Um die Ursache für bestimmte Ereignisse, wie Build-Fehler oder Metrik-Veränderungen abzuschätzen, benötigen Stakeholder Einblicke in vorangehende Ereignisse im zeitlichen Zusammenhang. In dieser Arbeit wird der Ansatz einer chronologischen Darstellung von Software-Evolutions-Daten umgesetzt. Das Tool namens SQA-Timeline visualisiert verschiedene Ereignistypen und Metriken auf einer Timeline. Mit seinem Drill-Down-Ansatz bietet das Werkzeug unterschiedliche Informationsgranularitäten von einer groben Übersicht bis zu einer Quellcodeansicht. Um herauszufinden wie der Ansatz das Verständnis von Softwareevolution fördern kann, wurde eine Benutzerstudie durchgeführt. Die Ergebnisse zeigen, dass die Teilnehmer mit SQA-Timeline bestimmte Aufgaben mit höherer Korrektheit und Effizienz lösen können als Benutzer, welche eine Reihe von häufig verwendeten Tools zur Verfügung hatten.

Contents

List of Figures

List of Tables

List of Listings

Chapter 1

Introduction

Modern software engineering environments rely on various software tools which support the development process. Projects are usually stored in a *Source Code Repository*. This can be a publicly available hosted service such as GitHub[1] or an internally set up solution. Source code repositories allow the developers to work on the same software project simultaneously. Contributions to the source code are usually reviewed and then added to the code base. Any contribution is tracked and can be reversed by the included versioning system. Features such as forking allow a code base to be developed individually at any time and even merged back into one piece at a later point.

Issue Trackers are used to keep track of bugs and upcoming tasks of the developed product. However, they are often much more than that and have a significant social aspect [BVGW10]. They serve as a platform for communication and coordination among developers and are usually a vital part of the development process. Popular issue trackers such as Jira[2] sometimes include project management features such as a sprint backlog for SCRUM [HBP09]. Open source communities often use issue trackers as a primary tool to discuss new features and improvements.

Code Quality Platforms such as SonarQube[3] analyse the source code and emit various metrics. These metrics can be used to evaluate code quality and other factors. Modern software engineering processes use metrics as part of quality control also known as Software Quality Assurance (SQA). A properly implemented SQA lowers the cost and risks of software development [SHK98].

A *Continuous Integration Platform* is responsible to deploy and test the source code committed to the repository. The build process is automated and ensures all required tests are passed before deployment. Modern agile software development processes require builds to be scheduled multiple times a day to make sure the current state of the code runs through the build process without errors or test failures [FF06]. Frequent builds allow the developers to use and test a current version and it is potentially easier to identify responsible commits after a build failure.

1.1 Motivation

The previously mentioned software tools are frequently used in development environments. Each tool provides its own user interface which can make it difficult to connect the data and draw

[1]http://github.com
[2]http://www.atlassian.com/software/jira
[3]http://www.sonarqube.org

conclusions. The integration of data across multiple platforms provides valuable insights into the developed project. Our previous project called SQA-Mashup successfully showed that using an integrated dashboard increases project awareness and reduces the time to solve specific tasks [BGG14]. Although SQA-Mashup platform features a timeline view, the tools focus was never on chronological data visualisation and is lacking many features in that respect.

Developers and project managers have certain information needs regarding their project. A chronological visualisation could prove to be useful to comprehend and draw conclusions out of a series of events. To find a commit responsible for a build failure, the developer needs to focus on commits since the last build success. A chronological visualisation will help the developer to identify the commits in between builds and could speed up the error identification process. Scenarios like the mentioned example are a part of the daily work of different stakeholders in software engineering. Information needs of developers are compiled into questions by different sources like Ko et al. [KDV07] or Fritz and Murphy [FM10].

1.2 Goal

The goal of this thesis is the implementation and evaluation of SQA-Timeline, a timeline based visualisation tool for chronological display of software evolution data. The visualisation needs to be capable of handling large scale software projects and the large amount of events they are associated with. A multi level navigation concept will help the user to navigate through different zoom steps and focus quickly on the desired time frame. To enable extended drill-down navigation through the large amounts of available data, an overview visualisation will support the user to assess some summarizing data before continuing to the timeline view. The additional source code view will enable drill-down to the source code for detail analysis.

Multiple features will support the navigation to provide the means to quickly display the desired data. Search and filter functionalities are one aspect of this. Other features will be navigating between linked events such as the date an issue was opened, updated or closed. The user should have different means to navigate the timeline and be able to select a way based on the goal he wishes to achieve.

The timeline will include the possibility to visualise certain patterns in the project history. Potentially error prone events or otherwise interesting time frames will be detected and displayed for the user upon request. The pattern purpose and definition will be defined and agreed upon with the developer of the back end solution called SQA-Pattern. The back end will handle the data mining and pre-processing including the pattern recognition and is not part of this thesis. The SQA-Timeline is responsible for the visualisation and will communicate with the back end using an API.

The evaluation of SQA-Timeline will try to answer the following research question: *How can a timeline-based visualisation approach foster the understanding of software evolution in large-scale software systems?* In other words it will be evaluated if SQA-Timeline provides an additional value towards satisfying information needs of developers. A user study will be conducted with two user groups. One group with SQA-Timeline support, the other one using the standard UIs of available tools. Both groups will solve a series of tasks in order to provide insights about effectiveness and possible improvements of SQA-Timeline.

Chapter 2

Related Work

2.1 Software Quality Assurance

To be able to develop and release state of the art software systems a development process needs to involve some form of Software Quality Assurance (SQA). With increasing size and complexity of software systems, having reliable SQA tools and processes in place becomes even more fundamental. Back in 1984, Buckley and Poston [BP84] described SQA as relatively new and autonomous field. However, they did correctly predict that the importance of SQA will increase in the future. They identified three main reasons why software developers should be concerned with SQA:

- **Legal Liability** is always important if some form of contract is closed. Sold software needs to fulfil certain requirements. These can be strictly functional but sometimes assurances in code and development quality need to be made as well. If a released software does not work as intended due to quality issues, a customer can take action against the developer the same way as it is with any physical product.
- **Cost Effectiveness** is an economical incentive to implement SQA. Even though effective SQA is cost intensive, research and practice has shown that SQA lowers the overall cost and many risks affiliated with software development. This is valid even more so for nowadays large scale, highly complex software systems [SHK98].
- **Customer Requirements** are closely linked to legal liabilities. Some customers demand implementation of certain SQA standards. Software is used in many business critical environments like banks which need reliable software to do business properly. Some software, for example in aircraft, even has lives depending on it and has very strict quality requirements.

Nowadays, many standards of SQA exist and it is up to the developer how it is implemented. There are different definitions of the scope of SQA, but mostly it is of a very general nature. It includes development and management processes as well as technical definitions like testing and code analysis [OK09]. Agarwal et al. [ANM$^+$07] provide a definition of the term:

> "SQA is the planned and systematic approach to the evaluation of the quality of and adherence to software product standards, processes and procedures. It includes the process of assuring that standards and procedures are followed throughout the software lifecycle."

2.2 Source Code Analysis

To be able to conduct effective SQA, information about the written code is required. Source Code Analysis is the field of automatically analysing the source code to get information about it [Bin07]. It is an important part of the field of Software Quality Assurance an this is why it is mentioned separately at this point. Software suites and tools like SonarQube[1] are available for developers, testers and managers to provide automatically extracted data about the source code. This data can direct the right people to the source of potential problems and weaknesses in the code. The resulting values of a source code analysis can have different forms. Calculated numeric values, called software metrics, are commonly used to get information about the code. The following list shows a few examples of software metrics used in SonarQube:

- **Lines of Code**
  A simple metric counting the lines of code. It shows if the code basis is growing or shrinking. Both effects can be desired depending on the situation.
- **Average Class Size**
  The average class size is another size metric that shows the average lines of code of a class. In object oriented programming a small class size is usually desired and indicates well structured code.
- **Test Success**
  This metric shows the ratio of test success for a selected piece of code. Test failure indicates programming or integration errors and usually needs to be addressed immediately.
- **Rules Compliance**
  The rules compliance metric is often referred to as code quality metric. It sets different code issues in relation to the lines of code. SonarQube is able to detect various code issues like bad coding style or potential error sources. The more of these issues that are found in relation to the code size, the potentially worse the quality of the code.

Hundreds of these software metrics are available and some can indicate quality of the code [FP98]. Software metrics are only indicators and it can be dangerous to rely too much on these calculated numbers [KMB04]. For example we can see how much of the code is commented, but it is not visible what the quality level of these comments actually is. However, software metrics can be a powerful monitoring tool to ensure code quality. Statistical Process Control (SPC) using software metrics can be applied to a software engineering process to ensure process and product quality [FCJV13]. An implemented SPC using software metrics is proposed in Capability Maturity Model Integration (CMMI)[2] on maturity level 4. Models like the CMMI are aimed at process improvement such as software engineering processes.

2.3 Continuous Integration

In early days, software engineering projects would be integrated when certain parts were fully developed. This integration process can be very time and resource consuming and even be the reason for a project failure. Continuous Integration (CI) is the practice of continuously integrating a software project, which means merging development copies with a shared main line [FF06].

[1]http://www.sonarqube.org
[2]http://cmmiinstitute.com

Usually these merges will happen at least once a day. Nowadays, CI platforms have many features such as testing, code inspection, database integration, compilation, and deployment. Many different software solutions, open source or proprietary, are available today. The key to effective CI is to integrate and build newly committed code as soon as possible. A build failure which can for example be caused by failed tests will immediately draw attention to the recently committed code. The more often builds are executed, the potentially easier it will get to identify the faulty commit. A properly implemented CI helps to reduce risks and increase software quality [DMG07]. In order to keep track of builds and events which happened in between them, a proper monitoring tool is needed. The SQA-Mashup tool described in Section 3.1 provides monitoring functionality for different data sources. It has been shown that the tool provides more team awareness and better project visibility compared to stand-alone CI-tools [BGG14].

2.4 Developer Information Needs

In software engineering projects, developers have certain information needs. Haenni, Lungu and Schwarz [HLSN13] categorized information needs of developers who work on projects which are part of a larger ecosystem. They analysed upstream and downstream projects and identified different needs. Both developer categories need information about the other and their ecosystem to be able to develop efficiently. Ko, DeLine and Venolia [KDV07] managed to identify 21 types of information needs occurring in collocated software teams. They analysed search times and outcome for each need and their role in decision making. Some needs are easily satisfied and some remained unsatisfied. According to them, innovations in tools and processes would be able to increase the satisfaction of information needs. Another list of questions developers ask is provided by Fritz and Murphy [FM10]. They introduced an information fragment model to help answer these questions.

One general type of information need is focussed on metrics and code analysis. The results of source code analysis can be used to satisfy information needs of developers. In order to do so, tools need to be easy to use, fast and produce concise output [BZ12]. Other approaches focus on bugs and how they can be predicted. Bug prediction uses statistical means and machine learning to find pieces of code with high probability for bugs. Software metrics usually play a central role in the analysis process which makes bug prediction a sub field of Source Code Analysis. However, not all forms rely purely on code. Various approaches are currently being researched [DLR10]. Most of them use information from versioning systems to calculate probability of bugs. This approach is used very often, because history of code changes and repository metadata can proof to be very valuable. The knowledge which piece of code has changed enables significant performance improvements in otherwise very costly analysis processes [RPH$^+$11].

Many information needs in collaborative software engineering are user-focussed [BZ12]. The second most sought-after information identified by Ko et al. [KDV07] is 'What have my coworkers been doing?'. Project management and social components are an important factor in software development. Information from source code repositories and continuous integration platforms can be used to satisfy some of these information needs. Treude and Storey [TS10] conducted a study how developers maintain awareness of various aspects in software engineering using dashboards and event feeds. They show that team awareness is very important and that the right tools help the developers to satisfy their information needs.

2.5 Time Based Data Visualisation

The visualisation of time based data is a very broad field. Time based data has been visualised since ancient times in many different ways. These visualisations range from a timeline in a history book to a financial time series chart. Miksch and Schumann [MS11] released a very thorough collection describing many different ways of visualising time-oriented data. They collected visualisation practices and applications from many different fields which facilitate a solid foundation of how time based data should be perceived and visualised. The data which is visualised in this thesis has different properties. Metrics for example consist of quantitative numeric values and commits or issues form qualitative data. In order to successfully visualise data following criteria have to be met [SM99]:

- **Expressiveness** requires the visualisation of the exact information contained in the data.
- **Effectiveness** addresses the ability for a human to intuitively recognise and interpret the visualisation.
- **Appropriateness** sets cost factors like required computing power and size into relation to its value.

Visualisation of time based data in software engineering has been done before. Many applications like SonarQube allow the user to see data on a timeline. Other approaches try to support developers by showing concerns (aspects) of a developed software on a timeline to create a heatmap of how many times a particular concern is brought up [TS09]. The unique feature of the visualisation in this thesis is the combination of different data sources into one timeline to provide a more complete picture of the software engineering process.

Chapter 3

Background

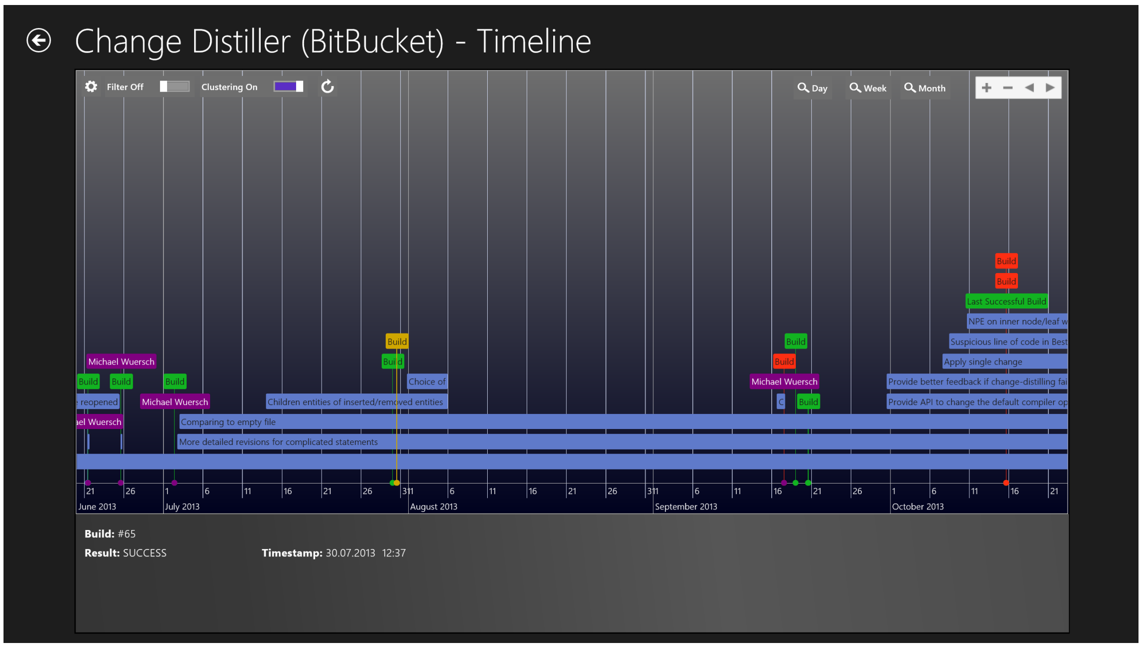
3.1 SQA-Mashup

As described in Chapter 2.2, software metrics are only indicators and do not allow absolute conclusions about a piece of source code. There are other data sources available which combined with metrics can help to conduct effective SQA. This includes source code repositories, issue tracking systems, and continuous integration platforms. In our last project called SQA-Mashup[1] we tried to combine these different data sources to provide a more effective SQA experience [BGG14]. The SQA-Mashup back end features a piping system which enables flexible configuration and mashup of the desired data. In the front end, various widgets are available to display the data resulting from the pipes. Different widgets can pull data from various data sources like build information, issues, commits, and software metrics. Widgets can be dynamically added, moved or removed from the dashboard. The main dashboard also features different views which can contain a different set of widgets. This feature can be interesting if the software is used by different stakeholders such as developers, testers, or software architects.

Besides the main dashboard, the application features a timeline view, which displays commits, issues, and build information in a simple timeline. The timeline view features basic zoom and panning functions. The different objects are all displayed in the same space and are simply stacked on top of each other. Single time events such as commits and builds are displayed very straight forward at the point in time they happened. However, issues feature multiple time events for example when they are opened, closed, or updated. As long as an issue is open it will show up as a time interval bar in the timeline which ends when it is closed. If it is reopened, the bar will show up again. If an issue is not closed up to today, the issue interval will be shown until the today marker. To avoid heavy overload of the timeline, a clustering system can be activated which combines element groups to a cluster element. Clicking on a visible item displays further information in a reserved space below the timeline. To navigate in the timeline, the user could use the pan and zoom buttons on the top left or use touch or mouse with mouse wheel to interact. Three additional buttons on the top right side allowed the user to quickly change the displayed time interval between day, week, and month.

The timeline allows filtering of displayed items. Issues, builds, and commits can be filtered by various event specific properties such as build result or issue type. Apart from these basic filters, a smart filter system is available which only displays potential error sources and therefore interesting events on the timeline. The smart filter system is a first attempt to automatically analyse the mashed up data and attempts to direct the user to potential error sources. The timeline

[1]http://www.ifi.uzh.ch/seal/research/tools/sqa-mashup.html


Change Distiller (BitBucket) - Timeline
Filter Off
Clustering On
Day
Week
Month
Build
Last Successful Build
NPE on inner node/leaf w
Suspicious line of code in Best
Apply single change
Provide better feedback if change-distilling fa
Provide API to change the default compiler op
Michael Wuersch
reopened
Choice of
Children entities of inserted/removed entities
Comparing to empty file
More detailed revisions for complicated statements
June 2013
July 2013
August 2013
September 2013
October 2013

Build: #65
Result: SUCCESS
Timestamp: 30.07.2013 12:37

Figure 3.1: The SQA-Mashup timeline.

featured in the SQA-Mashup project is by no means a perfect solution and was more thought of as a nice feature to have next to the main dashboard. The evaluation and collected feedback about the SQA-Mashup project have shown that the timeline approach has a lot of potential and could be useful to keep track of software engineering projects and to identify potential problems. This is why the SQA-Timeline project is now being implemented in this thesis. Usage and evaluation have shown drawbacks in the SQA-Mashup timeline which will be addressed in the new implementation in form of the standalone SQA-Timeline.

3.2 Source Code Repositories

Source Code Repositories are a vital component of any continuously integrated code base [FF06]. They include a Version Control System (VCS) which keeps track of developer contributions and changes on the code base. Among many popular solutions are the server based Apache Subversion[2] (SVN) or the distributed Git[3]. The repository can either be hosted privately or by a provider such as GitHub[4] or BitBucket[5]. These cloud based services are a popular choice for open source projects and often provide additional functionalities such as issue trackers and other ways for the developers to interact with each other and discuss the code.

GitHub offers a dashboard including a few graphs which show the recent repository history. The charts show the amount of commits, additions and deletions per week, the activity of individual contributors, and the distribution of the commits over each week day. These charts allow a quick

[2]http://subversion.apache.org
[3]http://git-scm.com
[4]https://github.com
[5]http://bitbucket.org

overview over project activity but only offer limited in-depth information. Some charts, such as the contributions over time chart, as shown in Figure 3.2 are interactive and allow the selection of a time frame to show more detailed information. When a time frame is selected, a graph of contributions of each developer appears and with a click the involved commits are displayed in a list. This navigation concept of selecting a time frame to filter the displayed content is similar to the concept which will be used in SQA-Timeline. However, the GitHub visualisation requires a click on the individual developer graph and will navigate on a new page containing the list of commits which does not support a fluid selection and zoom on an individual commit.

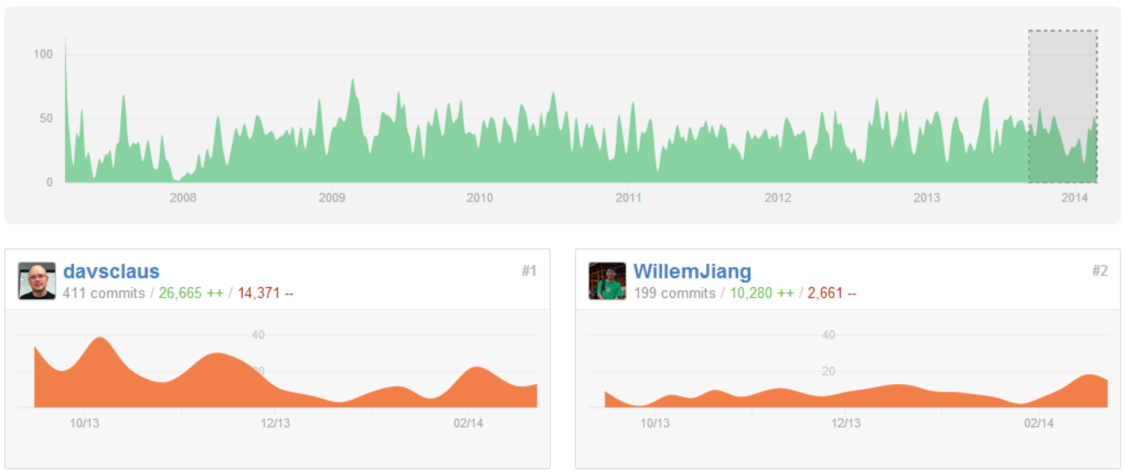


Figure 3.2: Interactive commit visualisation on GitHub (Apache Camel Project).

3.3 Continuous Integration Platforms

Continuous integration platforms build and deploy the source code in an automated fashion. Popular open source solutions are Jenkins[6] or Apache Continuum[7]. Since tests and builds are run automatically it is important the user can keep track of the build history. Jenkins relies mostly on list representation of projects and build histories supported by a few line charts which show the test success rate and build durations. In most cases, a user will only be interested in the current state respectively the last build result. List representations are valuable for this purpose. The build history can also be displayed on a timeline which displays the duration of the build. However, the lack of a zoom feature and the fact that only builds are displayed, minimise this visualisations usefulness. Apart from the built in dashboard, there are numerous plugins available for Jenkins offering additional visualisations.

[6]http://jenkins-ci.org
[7]http://continuum.apache.org

3.4 Software Quality Platforms

Source Code Analysis produces a lot of data which needs to be visualised properly. SonarQube[8] is an open source code quality platform which includes a dashboard. The main screen of the dashboard offers an overview over the tracked project. Various charts and widgets indicate the current values of several code metrics. However, SonarQube also offers an analysis of metrics history visualised in an interactive line chart called Time Machine. Knowing the development of certain code metrics can be a first step to identify possible error sources. For example a significant increase in test failures could indicate that faulty code was committed before that point in time. Because only metrics data is shown, commits and other events like builds need to be looked up in an external source. This increases the difficulty of the problem solving process.

The navigation in SonarQube UI is implemented using a drill down concept. The user starts with a complete overview and has the possibility to drill down in the project tree until he has reached a single source code file. Intermediate steps are for example code packages or classes which can help indicate which sections of the code are responsible for a particular metric value. The next sub level of the selected entity is displayed using a treemap, which enables visualisation of two parameters with block size and colour [JS91]. This gives the user an easily readable chart of two code metrics, which are by default 'Lines of Code' for size and 'Test Coverage' for the color of the treemap. To be able to quickly perceive important facts about the whole project and then being able to go into details using different granularities is a concept which is also incorporated in SQA-Timeline.

3.5 Technology Decisions

The choice of what technologies should be used to implement a software can radically affect the result. The SQA-Timeline is a front end application including graphical visualisations and standard user interface components. Modern web technologies offer many features and have huge community developing libraries and content to build upon. An important characteristic of web technologies is the platform independence which can barely be matched by any alternative. Any platform running a modern browser will be able to display the content without installation or setup. Web applications can also be adopted for mobile devices and compiled into a mobile application by using a technology such as Apache Cordova[9].

Alternatives to a web application would have been platforms such as Microsoft .NET[10] or Adobe Flash[11]. However, these platforms are struggling to keep up with the increasing popularity and possibilities of web technologies. They are also proprietary and lack the open specification of web standards.

3.5.1 HTML5

HTML5 is the buzzword when it comes to new web technologies. HTML stands for Hyper Text Markup Language and is the standard used to create web pages. It was designed for static

[8]http://www.sonarqube.org
[9]http://cordova.apache.org
[10]http://www.microsoft.com/net
[11]http://www.adobe.com/devnet/flash.html

web pages and with increasing demand in more complex applications, a new version 5 is currently a candidate recommendation[12] of the World Wide Web Consortium (W3C). This means the standard is not fully finalized yet, however many features are already implemented in modern browsers. HTML5 offers various new options for the development of web applications including video, audio, local storage and graphics features like SVG or Canvas. Cascading Style Sheets (CSS) is used to apply a display style to HTML and the newest version 3 is technically a separate developed standard, but its used by HTML5 and often not separately mentioned.

HTML5 offers a very flexible and powerful foundation to implement a more complex visualisation project like SQA-Timeline. Support for SVG and other display features will be heavily relied upon. This means only the newest browsers implementing all necessary standards will work to full extent. Google Chrome and Mozilla Firefox are considered the target environments. Other browsers may work, but will not be extensively tested during the development process.

3.5.2 SVG

SVG stands for Scalable Vector Graphics and is a standard for vector-based graphics by the W3C. SVG offers much more features to draw shapes and graphics than HTML and has better performance if many objects have to be rendered on screen. It is an XML based format which can be directly embedded in HTML5. This means SVG elements will be added to the Document Object Model (DOM) when rendered and do get some of the properties of normal HTML nodes. CSS styles can be applied to SVG nodes as well as click handlers. Each displayed element is a unique entity as in the DOM which makes the development of interactive and dynamic visualisations much more convenient than using for example HTML5 Canvas. On HTML5 Canvas each pixel is drawn separately and the structure of the drawn content is not represented in the DOM. This makes it very cumbersome to use and more appropriate for renderings with less interactivity.

3.5.3 JavaScript

JavaScript is the primary client side language supported by all modern web browsers. It is a scripting language and does not offer the structuring and object oriented design principles like fully fledged programming languages such as Java or C++. It was not designed for large projects and it is therefore imperative to use design patterns for structuring and encapsulation. The SQA-Timeline is a fairly code heavy web project which includes a lot of data processing. Navigating and filtering the timeline will require many DOM updates and traversal of large data sets to display the desired content. The JavaScript community offers many libraries like jQuery or D3 which makes JavaScript a very flexible language for any kind of task. Modern browsers like Chrome and Firefox consistently improve the performance of their JavaScript engine and offer features like garbage collection.

3.6 Data Visualisation with D3

Instead of using a pre built timeline library as in SQA-Mashup, the visualisations for SQA-Timeline are built with D3[13]. The low level graphics library offers the flexibility needed to shape the visualisation to its requirements without the restrictions a higher level library would come with. D3 stands for Data-Driven Documents and offers the means to display data in any shape or

[12] http://www.w3.org/TR/html5
[13] http://d3js.org

form in a web browser.

A web browser displays content using a Document Object Model (DOM) which is a representation of HTML or XML content. The DOM is a tree structure and is used by the browser to render and display content. By manipulating the DOM with JavaScript, the displayed content can be altered dynamically. Most graphics libraries introduce a layer of abstraction and encapsulate the DOM with a higher level API to make the display of content more convenient. D3 however has a different approach and provides a set of tools to transform the DOM directly [BOH11]. The approach is very similar to other document transformation libraries such as jQuery[14], which make the manipulation of the DOM more convenient than using the DOM API provided by the browser.

D3 offers full access to native representation which makes complete use of the power of web standards. The encapsulation other low level libraries like Raphaël[15] provide, can be more convenient, but they ultimately impose a simplification of the full functionality the web standards provide. When using D3, the user has the choice to use the full expressiveness of web standards like HTML, SVG and CSS [BOH11]. It is also no problem to use other document transformation libraries like jQuery alongside D3, since they can offer more convenience in certain areas.

The core functionality of D3 is the visualisation of data. Instead of having a representation of the displayed data somewhere in a JavaScript model, it can be bound directly to DOM nodes. To accomplish this, D3 offers a data join function to bind data to existing nodes or insert new nodes in the process. This functionality can be used for dynamic representation of data in the DOM. The following example of the update, remove and insert pattern is how the data binding mechanism works:

```
//select all existing circle elements
var circles = svg.selectAll("circle");

//join the 'data' array with existing circles (update)
circles.data(data);

//remove unused DOM nodes
circles.exit().remove();

//insert new DOM nodes
circles.enter().append("circle")
   .attr('r', function(d){ //set 'r' (radius) attribute dynamically
      return d.radius;
   });
```

Listing 3.1: The data binding pattern in D3

D3 is heavily based on DOM selections. This concept is also used by other document transformation libraries like jQuery. In order to transform DOM nodes, they need to be selected first. Selections are done using the CSS selector syntax which is a web standard. The data binding pattern shown in Listing 3.1 is also built around selections. The data is bound to a selection of nodes. In the example, all circle elements contained in the SVG node are selected. This selection can also be empty, the data is still bound to it using the `.data()` command. During the data

[14]http://jquery.com
[15]http://raphaeljs.com

binding process, D3 creates two additional selections. The DOM nodes which do not match the data can be selected using the `.exit()` function and are removed. Data values which have no matching DOM node yet will be selected using the `.enter()` function which returns a set of yet empty DOM nodes. This selection can now be used to insert the DOM content. Once the data is bound to nodes, attributes and other properties can be set and updated dynamically according to bound data. This feature makes D3 very flexible because the bound data can directly influence the appearance of the node in any desired form [Mur13].

Apart from low level DOM manipulation, D3 includes a large set of helper functions to support the user in data manipulation and graphics display. The SQA-Timeline makes heavy use of scale functions which transform data values to pixel coordinates to be able to display them on an axis. On top of these scale functions, axes can be generated and other features like zoom can be added to provide the user with basic tools to create an interactive graph or timeline. D3 supports HTML as well as Scalable Vector Graphics (SVG), which is an XML based vector graphics format supported by most modern browsers. Like HTML, SVG is DOM based and any displayed element is a DOM node. To display graphical objects it offers more features and better performance than HTML. Therefore most higher level features of D3 are only available when using SVG.

3.7 Requirements Derived from SQA-Mashup

The following section features a list of requirements defined for the development of SQA-Timeline. The timeline included in the previously implemented SQA-Mashup tool gave valuable insights in a timeline based approach. The tool has been evaluated in a user study and feedback was received. The identified drawbacks are addressed in this requirements list to improve the user experience and the value of the newly developed SQA-Timeline. Other influences are possible use cases and the satisfaction of information needs of developers.

Displayed Data
Data will be gathered from source code repositories, issue trackers, software quality platforms, and continuous integration platforms. The data is mined and processed by the back end project which is not part of this thesis. Key data items which need to be displayed on the timeline are:

- Commits (Repository)
- Issues including duration and events (Issue Tracker)
- Builds (Build Platform)
- Software metrics (Quality Platform)

The user will be able to get detailed data regarding each item, which should prevent him from having to look up the item at its original location. The data will be visualized on three separate screens: overview, timeline and source code view. Each screen will offer different detail granularity of visualised content. Using the available screens, the user will be able to navigate through content from a general summary down to an individual source code change.

Scalability
The SQA-Timeline needs to be able to display data from small to large projects. Large projects usually have more events which need to be handled by the interface. This is one of the main concerns of the SQA-Mashup timeline implementation which stacks all displayed elements on

top of each other. A clustering function helps to reduce large stacks, but it is inconvenient if the desired data is hidden under a cluster. To be able to see details but still keep track over a larger time frame, a multi layer view will be implemented for SQA-Timeline. The separate layers will feature different detail granularities and interaction possibilities.

Organised Display
The user should be able to comprehend the displayed data quickly. Different elements like commits or issues need to be displayed in an orderly fashion. The SQA-Mashup timeline displays all items in the same display space. There is no specific order to it, which can lead to confusion. To prevent this, colour coding and distinct sub spaces such as lanes will help the user to read the visualisation even if many items are displayed.

Navigation
In order to quickly display the desired data, multiple navigation options need to be present. The SQA-Mashup timeline only allows panning and zooming, which is not always the best way to get to the desired data. The SQA-Timeline will tightly interact with the adjacent control view which offers different features like searching and list representations of linked items. The user will be able to focus the timeline on different events as well as display more detailed information back on the control view.

Search
A search functionality will allow a user to quickly find a desired event. The search will support searching by keyword or date. It should be easy to use and does not require any additional configuration. The user will have the possibility to search for date in different formats or keywords and even combinations of the two. The input is done in one search field and will be interpreted accordingly. Search result will be categorised by event type and sorted by search term matching grade.

Filters and Highlighting
Developer focussed filtering will be included in SQA-Timeline. The user will be able to highlight and filter commits or issues associated with a developer. This can be a valuable feature if only the activity of one or multiple developers needs to be analysed. The highlighting feature will also help distinguish the activity of a developer and compare it to other events.

Patterns
The user will be able to display various patterns occurring in development history. Patterns are configurable and can be used to detect policy or convention breaches, possible error sources or otherwise interesting time frames. Pattern matches and a user set severity will also be calculated for every developer, so policy breaches and other possible events can be related to a certain developer.

Configuration Options
The SQA-Timeline will be designed to be used with projects of different sizes and properties. Various options will be available to configure the visualisation to individual preference and requirements. The user should have the choice what data he wants to see and how it is visualised. If for example only issue data is required and nothing else, he will be able to turn off the display of other events.

Additional Views
To justify SQA-Timeline as a stand-alone application, two additional views will be added. The overview screen will provide a quick overview of the project and the development activity. On the other side, a source code view will enable the developer to see source code changes of individual commits without using an external tool to do so.

Chapter 4

SQA-Timeline

SQA-Timeline is the name for the application which has been implemented during this thesis. The application offers one user interface for data which is usually scattered across different tools. The currently integrated tools are source code repository, issue tracker, CI-platform, and SQA-platform. SQA-Timeline enables stakeholders to interlink the information contained in these tools and offer the means to analyse and interact with the visualised data. Depending on stakeholder and project setting, the application may serve a different purpose. The main use cases in mind when developing SQA-Timeline are summarised in the following list:

- **Increase Project Awareness**
  To know what is going on in a project is important for developers as well as managers. Even more so if the stakeholder is involved in multiple projects. With SQA-Timeline it is possible to get a quick overview over events and development of metrics without having to gather the data in separate tools. It provides a single point of access for software evolution data. Increased project awareness helps stakeholders to notice irregularities which they may have overlooked if the information was located in separate tools [TS10].

- **Localise Error Sources**
  In larger projects it can be crucial to know who changed which piece of code in a certain time frame. SQA-Timeline is designed to support stakeholders in localising and identifying error sources in certain development scenarios. If an error occurs at some point, it is often the first step to find out who is responsible. Having a chronological visualisation of events can support this process. Other potential error sources may be identified by using patterns. An example use case is the identification of potential causes of a build failure. Using SQA-Timeline the user can identify commits and issue changes which occurred prior the build failure. Using this information he is able to narrow down possible causes.

- **Impact Analysis of Code Changes**
  Code changes influence various software metrics which are monitored during software quality assurance processes. The SQA-Timeline displays software metrics in context of source code contributions. This enables the stakeholder to analyse the impact of code changes on the software metrics and to counteract a negative development. For example, if the code quality metric decreases significantly after a commit, the developer may take the necessary steps to maintain the quality level.

- **Identifying Policy Violations**
  Software projects usually implement certain rules and policies. These rules can be there to ensure an orderly software engineering process as well as code quality. Some of these rules may be implemented as a pattern in SQA-Timeline in order to monitor compliance for the project. Violations can be quantified quickly and necessary steps may be undertaken. A possible use case example is the monitoring of a commit message policy which demands an issue-id for each commit. A pattern in SQA-Timeline can help a manager to monitor if this policy is followed.

The core part of the SQA-Timeline is the chronological visualisation of events in form of a timeline. However, the application features three screens, which follow an information drill-down concept. Each screen offers a different level of information granularity. The user is able to freely navigate between screens in order to get to the information he desires. A detailed description of the different screens and their functionalities is given in this chapter. The following list summarises the three screens and their purpose:

- **Overview**
  The overview screen offers a quick glance at some important facts about the project. It features a summary of different events concerning the project as well as developer activity and pattern matches.

- **Timeline**
  The timeline screen is the core view of the application. It features an interactive timeline which displays events in chronological fashion. It has many features to navigate and display desired events as well as patterns.

- **Code**
  The code screen allows the user to analyse the contents of a commit. Code changes in a file are visualised by a diff view.

4.1 Layout

The layout of each individual screen follows a unified concept as displayed in Figure 4.1. The three screens can be navigated by using the navigation bar at the top. Each screen features one or multiple visualisations which are displayed in a main content area. To support the main visualisation, context sensitive controls are displayed on the right hand side. These controls are categorised into tabs and offer features for the main visualisation which include searching, options and displaying of more detailed content of featured data entities. Each screen has its own set of control tabs and has a unified design and functionality in order to improve usability of the application. Each application screen and control tab maintains its state on navigation. This helps the user to navigate back and forth in the application without having to worry about finding the displayed content again.

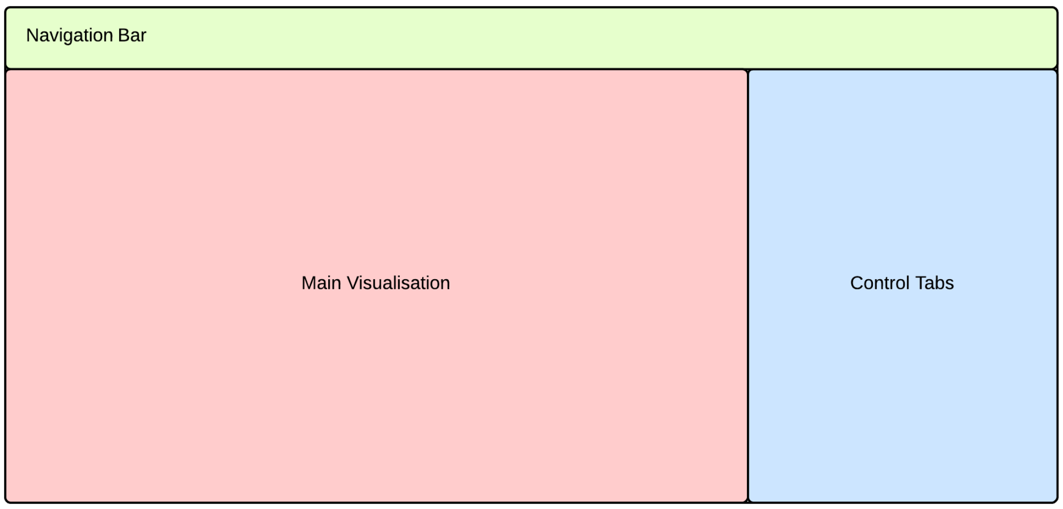


Figure 4.1: The SQA-Timeline layout.

4.2 Overview Screen

The overview screen offers a summary of the project. It is designed to act as a dashboard which quickly visualises several aspects of the project without having to look at multiple sources. The overview is interesting for managers and lead developers to keep track of the project state and recent events. Furthermore, it could for example prove to be helpful for developers involved in multiple projects or after absences. They would be able to quickly asses what has happened in a project. The overview screen offers an entry point for the timeline view. The stakeholder may decide to proceed to the timeline view in order to get more detailed information. However, in certain cases it could also be the case that a stakeholder is satisfied with the information available on the overview screen. For example if a manager of multiple projects sees that no build failures occurred and the commit numbers are reasonable, he may have no reason to get more detailed information about the project.

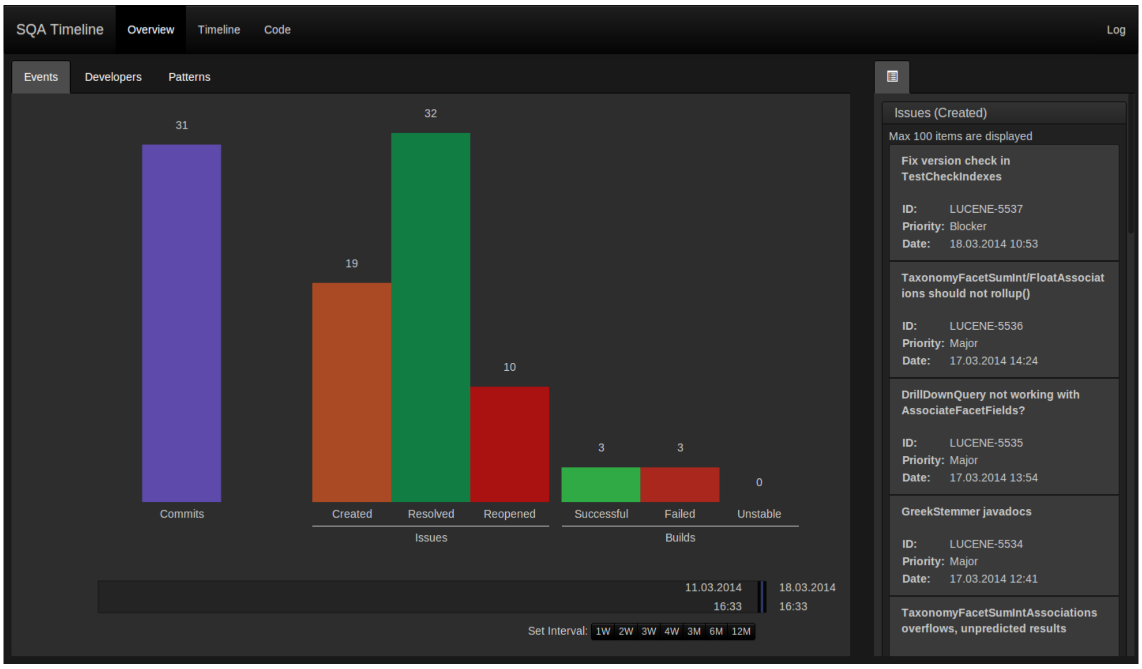
The overview screen features three tabs which offer different information. An *event tab* summarises certain events which occurred during development. The *developers tab* offers insights into developer activity. The *pattern tab* points out occurrences of defined patterns for this project. Each tab as well as the concept of the pattern severity rating will be introduced in the following sections.

4.2.1 Events Tab

The events tab is displayed by default when navigating to the overview screen. It features an interactive bar chart representing the number of occurrences of the following events:

- Commits
- Issues (created, resolved, reopened)
- Builds (successful, failed, unstable)

Per default the bar chart shows all events over the full time range available in the project data. However, using the selector bar below the chart, a user can select an arbitrary time range. The two end handles at both ends can be dragged to fit the desired time window. The interval itself can be dragged as well and will be kept constant when doing so. The bar chart instantly updates the displayed values on changing the selected time frame. Instead of choosing the interval by dragging the end handles to the right position, several presets are available represented by buttons below the selector bar. These presets range from one week to a full year and make it easier to quickly set a time interval.



Issues (Created)

Max 100 items are displayed

Fix version check in TestCheckIndexes

ID: LUCENE-5537
Priority: Blocker
Date: 18.03.2014 10:53

TaxonomyFacetSumInt/FloatAssociations should not rollup()

ID: LUCENE-5536
Priority: Major
Date: 17.03.2014 14:24

DrillDownQuery not working with AssociateFacetFields?

ID: LUCENE-5535
Priority: Major
Date: 17.03.2014 13:54

GreekStemmer javadocs

ID: LUCENE-5534
Priority: Major
Date: 17.03.2014 12:41

TaxonomyFacetSumIntAssociations overflows, unpredicted results

Figure 4.2: The Overview Screen Events Tab (Apache Lucene Project).

The event which has the most occurrences in the selected time frame will feature the tallest bar and set the maximum of the chart. All bars follow the same scale which makes the relation of values between events quickly perceivable. A click on any of the bars will display the contained events as a list in the control tab area on the right side. The list features a few details about the events and shows the latest occurrence first. Like the bar chart, the list is automatically updated when a new time frame is selected. For performance reasons the list is limited to the last 100 events of the particular selection. If the information about a displayed event in the list is not sufficient, the item can be clicked to select it on the timeline screen. The list view and the possibility to investigate an item even further on the timeline allows the user to transition through different information granularities.

Figure 4.2 shows the event screen representing the Apache Lucene[1] project. The time frame is set to display events of the latest week. This could for example be an interesting view for a project manager doing his weekly project review. The manager is able to see a summary of what has happened during the last week of development. After a click on the created issue bar, he is able to look through the created issues last week. He notices an issue with the highest priority rating and chooses to further investigate. A click on the issue will display the event on the timeline screen. More details as well as the history and current state of the issue are now available.

The described use case focussed on a manager or lead developer being the user. However the visualisation is interesting to any stakeholder who wants to quickly select and assess the events over a certain time period. This could potentially be a developer who was on vacation for two weeks or a tester wanting to take a look at the build failure to success ratio.

4.2.2 Developers Tab

The developers tab offers insights into developer activities. Two visualisations are available to the user, a bubble chart or a tree map. Both visualisations contain the following information for each developer having at least one commit in the selected time frame:

- Number of commits
- Weighted severity rating

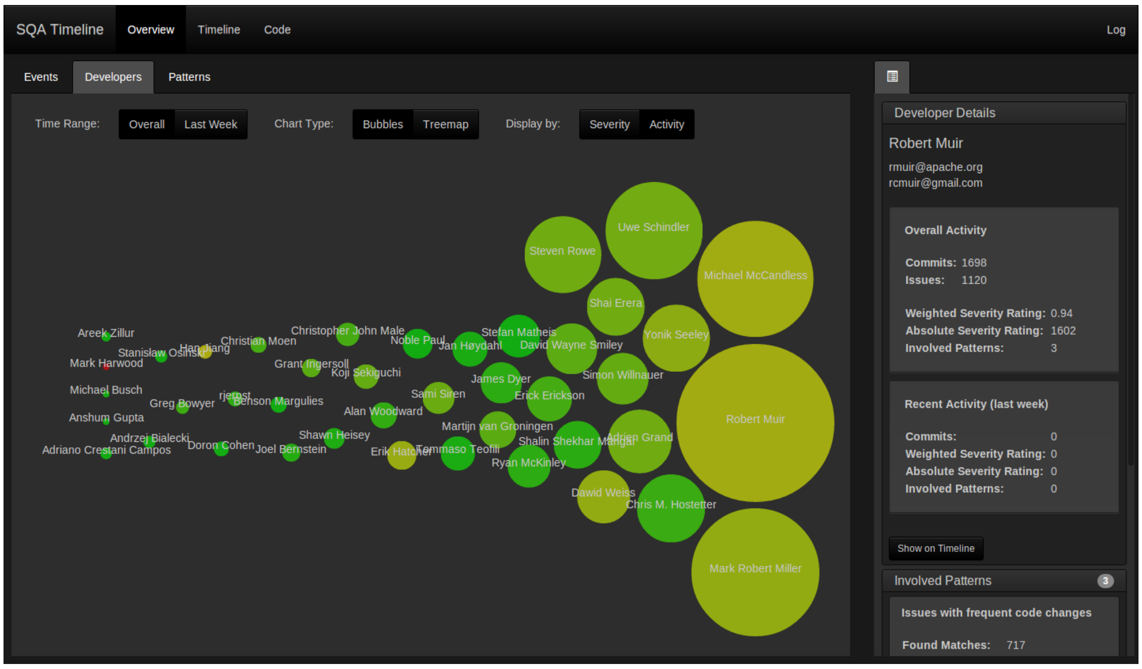
The visualisations offer two time frames for the displayed data: 'Overall' and 'Last Week'. The option causes the visualisation to be rendered with either the full data set or a subset only containing commits of last week. The bubble chart features a draggable bubble for each developer whose area grows in size according to the amount of commits of a developer. The colour of the circle is determined by the weighted severity rating. The severity rating is a concept described in depth in Section 4.2.4. It offers a way to detect the grade of involvement of the developer in patterns which have an assigned severity rating. The displayed colour ranges from green to red, which indicates lowest to highest weighted severity. The bubbles are sorted on the horizontal axis according to either the weighted severity rating (color) or the amount of commits (area) as it is shown in Figure 4.3. The positioning is not absolute and due to the movable nature of the bubbles, the sorting is only a tool to organise the graph to a certain amount.

If a developer is selected in the visualisation, a summary of his overall and recent activity is shown in the control tab on the right. Additionally displayed are the patterns this particular developer is involved in. Clicking the 'Show on Timeline' button or a pattern in the list causes a navigation to the selected entity on the timeline page.

Figure 4.3 shows the developer bubble visualisation for the Apache Lucene project. The bubble size reflects the amount of commits the developer has made over the projects duration. The colour of each individual bubble is defined by the amount of commits which do not have an issue assigned. It is deemed good practice for Apache[2] projects and recommended for projects using Jira in general to do so. The colour changes towards red the lower the ratio of fulfilment of this convention is.

[1]http://lucene.apache.org/core
[2]http://www.apache.org/dev/committers.html#applying-patches

The colours of the bubbles are to be understood as grade of involvement in certain patterns. The bubble colour is fully dependent how the severities are configured for the specific project. A red colour does not necessarily mean a developer did something wrong. The system could be configured to point to potential error sources like multiple commits on the same issue.



Time Range: Overall | Last Week
Chart Type: Bubbles | Treemap
Display by: Severity | Activity

Developer Details

Robert Muir
rmuir@apache.org
rcmuir@gmail.com

Overall Activity

Commits:	1698
Issues:	1120
Weighted Severity Rating:	0.94
Absolute Severity Rating:	1602
Involved Patterns:	3

Recent Activity (last week)

Commits:	0
Weighted Severity Rating:	0
Absolute Severity Rating:	0
Involved Patterns:	0

Involved Patterns 3

Issues with frequent code changes

Found Matches: 717

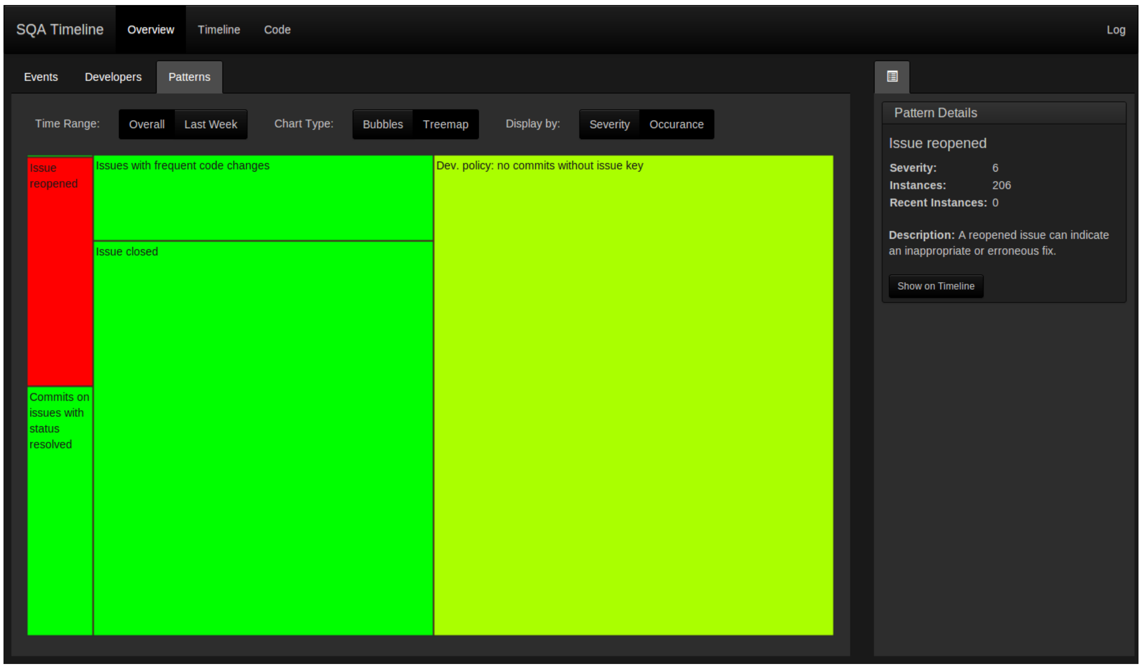
Figure 4.3: The Overview Screen Developers Tab (Apache Lucene Project).

4.2.3 Patterns Tab

The patterns tab offers a bubble and a tree map visualisation with the same features as the developers tab. However, the displayed data is the following:

- Number of pattern instances
- Severity of pattern

The purpose of the visualisation is to summarise all defined patterns for the selected project. A closer look at the available pattern types and possible use cases are provided in Chapter 4.3.6. The pattern tab shows the number of found matches which are called pattern instances in the application. The instances can be displayed over the full project duration or only last week like in the developers tab. The colour indicates the severity which has been defined for the particular pattern. Patterns are defined for different purposes and indicate, for example, possible error sources or policy breaches. Using the pattern tab visualisation, it is possible to quickly determine the amount of found instances. Figure 4.4 represents the patterns visualisation in its tree map form. A click on an individual pattern will show some details and allows the user to navigate to the timeline to further investigate the pattern. The colour indicates the amount of impact on the severity rating of a developer.



Issue reopened

Severity: 6
Instances: 206
Recent Instances: 0

Description: A reopened issue can indicate an inappropriate or erroneous fix.

Figure 4.4: The Overview Screen Patterns Tab (Apache Lucene Project).

4.2.4 Pattern Severity

As mentioned in the previous sections, the severity rating is calculated for each developer. It is a measurement for the grade of involvement in patterns which have a positive severity assigned. Each pattern has a severity value between 0 and 10. The following scale is used in the editor to define the severity of a pattern:

- 0: Neutral
- 2: Minor
- 6: Major
- 10: Critical

This mapping exists to make it easier for the user to pick a value. The default value is 0, which means the pattern has no impact on the severity rating of a developer. If a user chooses to set a severity on a pattern, each found match will increase the absolute severity rating of a developer. To calculate the weighted severity it is divided by the amount of commits this developer has made.

If the pattern detecting commit messages has a severity of 2 and there are 5 matches found for a developer, his absolute rating will be 10. If he made 100 commits in total this means his weighted severity rating is 0.1. Therefore, the weighted severity is a number which can be compared to other developers to indicate the ratio between non-issue commits and the ones that follow the guidelines.

The pattern severity system is an attempt to make pattern matches visible quickly on the overview screen. It indicates the involvement of developers in patterns whose occurrence is tried to be minimised or entirely prevented. The severity rating of a developer only indicates the pattern involvement rate. If further information is desired, the individual pattern matches need to be investigated.

4.3 Timeline Screen

The timeline screen features the main visualisation of the application. Several event types occurring during the development process as well as software metrics are displayed in chronological fashion. The main visualisation is divided into three layers featuring different granularity levels of displayed information. Located at the bottom of the visualisation is the range selector which represents the full data range loaded into the timeline. It offers an axis with ticks every three months and does not display any data. On top of the range selector is an overview layer which displays event types in a limited fashion. This low resolution overview is designed to show data from a few weeks up to a few months. It allows the user to get an idea which events occurred over the selected time frame and how many of them. It is possible to quickly identify clusters of events as well as event free time ranges. Using the overview visualisation layer, the user will gain a sense of orientation in the history of events. Above this low resolution overview, the main layer visualisation features the highest detail grade of showed events. The main view is what the user will mostly use to interpret and analyse the projects historical data. Supported by the control tabs on the right, it offers various interactive features and configuration options described later in this section.

The visible time range on the main layer view is represented by a blue selection frame on the second layer. The currently displayed time range of the second layer is represented again by a selection on the third layer. The selector frames are interactive and can be dragged and resized to change the displayed time range of the previous layer. This three layer approach allows the user to quickly navigate to the desired time frame. The timeline is initialised to show the newest data first. If a user is only interested in recent data, he will probably only use the second layer selector to browse the timeline or adjust the level of zoom on the main view. However, if this is not enough there is always the possibility to change the time frame of the second layer by using the third layer selector. When moving or resizing a selector, the visualisation is updated immediately. This behaviour enables the user to see changes while manipulating the selection and to stop whenever the right position is reached. Apart from selecting the time frame on the second and third layer, the main view offers some additional navigation features. The main view can be dragged in either direction to browse the content. The mouse wheel is supported as well and causes the main view to zoom in or out. The zoom and drag feature are an intuitive alternative to navigate the timeline.

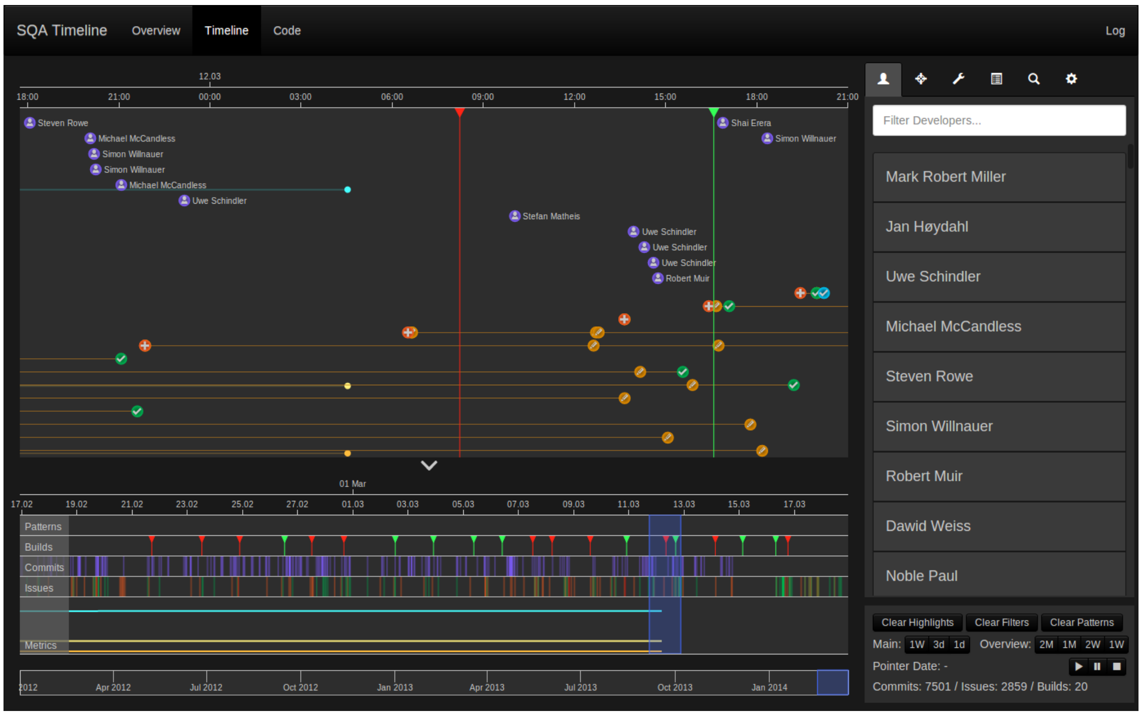


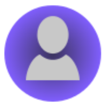
Figure 4.5: The Timeline Screen (Apache Lucene Project).

Additional functions and controls are located in the control tab section on the right side. Some controls are positioned in a persistent section at the bottom. However, the top section is organised into the usual tab layout. The functionalities of each tab will be described in detail later in this chapter. The following list represents each tab from left to right:

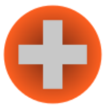
- **Developers**
  Lists the developers of the currently displayed project and offers options like developer specific highlighting and filtering.
- **Patterns**
  Lists the patterns which have been created for this project. Using the controls on this tab, the patterns can be individually displayed on the timeline.
- **Pattern Editor**
  Using the pattern editor, a user can create a new pattern for the currently visualised project.
- **Event Details**
  Shows detailed information about an event or data entity. It will always be shown if the details button is clicked on the timeline screen.
- **Search**
  Offers a search function for commits, issues, and builds. A search can be conducted using keywords and dates.
- **Options**
  Features various options to configure the timeline.

4.3.1 Visualised Data

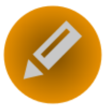
The timeline displays events and metrics in chronological fashion. Each event type has a unique appearance and consistent colour coding throughout the application. The events displayed on the main view feature various symbols to make different events distinguishable. Table 4.1 explains the meaning of symbols appearing on the main timeline view:



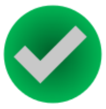
Commit
Each commit appears in the timeline with this symbol. Since a commit is a developer contribution, the symbol of a person has been chosen. Next to the symbol, the author name is displayed. This allows for a quick identification of the source. The date of the commit is the point in time when it is pushed to the main line. This sometimes result in multiple commits being displayed at the same time.



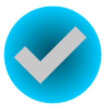
Issue created
When an issue is created, it is resembled by a plus. Since a new issue is usually something which needs the developers attention, an orange colour has been chosen. If there are other events further down the history of this particular issue, a thin brown line up to the most recent event will be displayed. This helps the user to identify events which concern the same issue as well as showing if there might be an event outside the displayed time range.



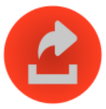
Issue updated
When an issue field is updated, a generic update event is shown. This event shows up when any field of the issue is changed. This can be a change of description or a comment of a developer. The particular changes can be viewed in the history browser of the detail view which is described in Section 4.3.4



Issue resolved
When an issue is resolved, it shows up as a green tick. This is usually the case when a developer implemented, tested, and committed the requirements of the issue. Depending on the projects issue workflow, the committer himself or a reviewer may resolve the issue.



Issue closed
Next to the resolved event, Jira offers also an issue closed event. This is useful to implement proper reviewing workflows. A workflow could for example expect the committer to resolve the issue and after that require a reviewer or issue creator to check the issue and to close or reopen it.



Issue reopened
A resolved or closed issue can be reopened, if the changes turn out to be inadequate. Many reasons can lead to an issue being reopened including errors and wrong or only partial fixes. A reopened issue usually requires the attention of the involved developers and therefore has a red symbol.

	Builds Builds are indicated by a line vertically stretching over the whole main view. The line is supported by a triangle on top which enhances visibility and simplifies selection by a cursor. The thin line allows the user to identify which events happened before and after individual builds. This may be of importance to find possible reasons for build failures. The build markers have different colours depending on the result. Successful builds are green, failed ones are red, unstable ones orange and all the other states are grey.
	Metrics Metrics are displayed as a line with dots when a measurement occurred. The height of the main view represents the Y scale. The maximum of the Y scale is dependent if the value is absolute or percent based. On the main layer, metrics are displayed partially transparent and are fully shown on mouseover.

Table 4.1: Events visualised on the timeline.

4.3.2 Dynamic Display and Scalability

A requirement for the timeline is scalability. The amount of events varies from project to project. Large scale projects may contain several commits and issue changes a day and multiple thousands a year. Small scale projects on the other hand will have significantly less activity and even contain idle time without any events at all. The timeline is very flexible in that regard. The top two layers featuring different content resolutions can be adjusted to display just the right amount of events. Some presets are available on the bottom left which allow the user to set the overview and main layer time interval to a different resolution. The presets range from one week down to one day for the main layer and from three months down to one week for the overview layer. These presets enable a quick adjustment of the content resolution. Figure 4.5 features a large scale project with the time interval on the main display layer set to about a day. Figure 4.6 on the other hand shows a significantly smaller project with the main layer resolution set to one week. As we can see, both visualisations display a reasonable amount of content without being too overcrowded or empty. The ability to freely adjust the resolution of each layer makes the visualisation potentially infinitely scalable. However, if a large amount of events happened at the same time, the main view struggles to reasonably display the content. In this case it is not possible to zoom in to minimise the amount of displayed events. Another issue could be if the density of events varies to a large extent in the project history. This would require the user to zoom in or out depending on the current event density. However, by using the mouse wheel zoom feature it is very easy to quickly zoom in or out.

The previous paragraph only discussed the density of events over time in general. But it is also possible that different event types have different densities compared to each other. A project might contain a lot more commits than issue events. There will always be variations were some intervals feature many of one type and less or none of the other type of event. As visible in Figure 4.5 the second layer visualisation is divided into lanes. These lanes are static and only display one type of event. This is intended, as it will make the density of specific event types visible. However, the main visualisation is organised using dynamic lanes which are not explicitly marked off. Commit and issue events divide most of the available space into two lanes. Each lane is divided into sub-lanes displaying individual issues or commits. Issue events belonging to the same issues will be displayed on one sub-lane. The commits are organised top to bottom and will restart at the top if all sub-lanes are filled. An algorithm analysing how many issues and commits are displayed in one time frame will set the available space for the issue and commit lanes to an appropriate size. Since every issue requires one lane can not be wrapped like commits, the issue sub-lanes are scaled as well so that every issue can be displayed. The dynamic lane sizing is conducted for every update of the main view to make sure the content uses the available space efficiently. If for example a time frame contains no issue events, the issue lane size will be set to zero and the available space is fully used by the commit lane.

The timeline is built to profit from large screen resolutions, but it also supports smaller window sizes and resolutions. Resizing the timelines width will cause the events to be pushed closer together or to move away from each other. By zooming in or out it is very easy to compensate for this. When the height of the application window is resized, the height of the main layer view will be resized accordingly. The overview layer has a static height, however it is collapsible as demonstrated in Figure 4.6. In collapsed state, the overview layer still displays the axis and the interval selector, but it will not contain any event information. The overview layer can be collapsed and un-collapsed quickly by using the arrow button above. If the application window goes below a certain height, the overview layer will be collapsed automatically. This feature helps a lot when displaying the application on smaller resolutions. The main view does not have a maximum size and can grow infinitely. The previously mentioned dynamic sizing algorithm will make sure the full screen size is used. The larger the window, the more events can be displayed. The timeline offers the best experience when using full screen mode on a monitor with full HD or even higher resolutions.

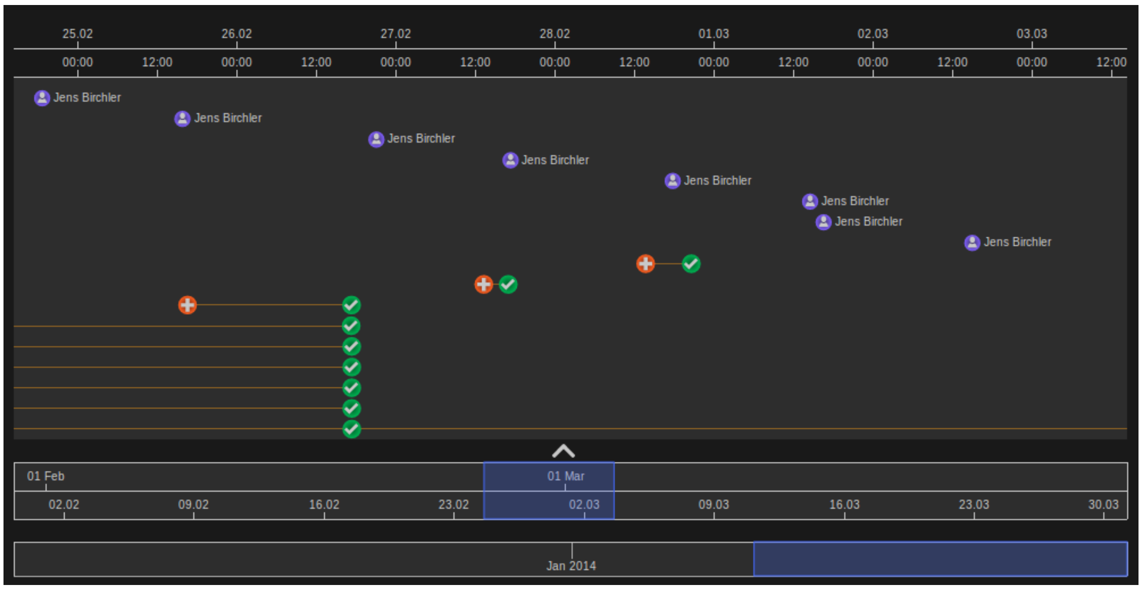


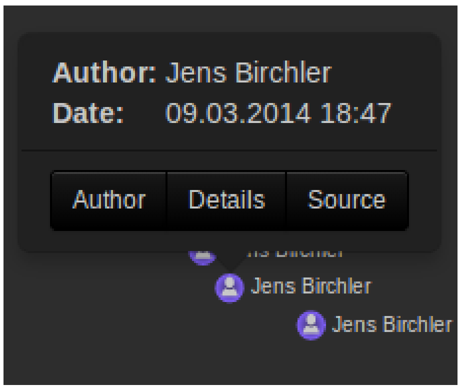
Figure 4.6: The timeline with collapsed overview layer (SQA-Timeline Project).

4.3.3 Context Information

The main view displays only limited information about events at a first glance. Different event types are symbolised by unique icons as described in Table 4.1. Only the commit events feature a label of the author next to it. The result of this mostly icon based approach is a clean and ordered interface. The user is able to quickly grasp the key event which happened in the displayed time frame. However, this approach also needs the user to be able to get further information about a displayed event. To solve this, a context sensitive pop-up is displayed on click on the event. The pop-up shows some important data fields as well as a context menu as shown in Figure 4.7 and 4.8. The content of the pop-up varies depending on what event has been clicked. The pop-up shows up for any displayed event on the timeline including commits, issues, builds, metrics, and patterns. The included context menu offers additional functionality. For most events it is possible to display further details by opening the detail view tab as described in Section 4.3.4. Several events feature additional options like looking up the author or source code for commits or highlighting contained items for pattern instances.

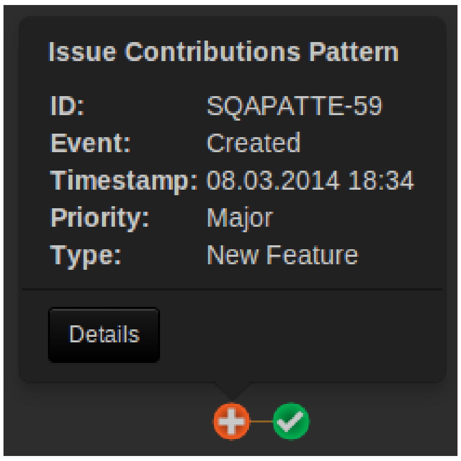
By only displaying the data when the user requests it, an information overflow in the timeline is prevented. Given that different stakeholders will require different data fields, it was a design choice to stick with the mostly label free approach with the exception of commits. Metrics are a special case, because the value is displayed on mouseover and on click the full pop-up is shown. To permanently display the value for metrics would make sense, however since the label may overlap with other displayed events, the mousover approach has been chosen. Any pop-up will stay open until the user performs a click anywhere else on the screen. This makes sure the information is displayed long enough, but it will close when the user decides to move on.

Another tool providing the user with context information is the insertion of a vertical line displaying the current date of the mouse cursor position. By simply hovering over an event, the user may see the approximate date and time of the event using this information. The vertical line at the position of the mouse cursor is also very useful to find out the order of events when they are separated by a larger vertical space. An example would be to find out if a commit happened before the issue was resolved. By hovering over the commit symbol, it will become easy to see how other events are positioned in relation to it.



Author: Jens Birchler
Date: 09.03.2014 18:47

Figure 4.7: Commit Pop-Up.



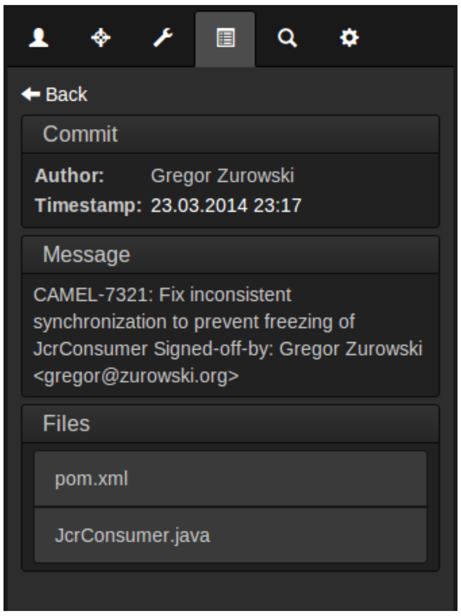
Issue Contributions Pattern

ID:	SQAPATTE-59
Event:	Created
Timestamp:	08.03.2014 18:34
Priority:	Major
Type:	New Feature

Figure 4.8: Issue Pop-Up.

4.3.4 Detail View

The detail view is part of the control tabs. Details about events or other data entities are displayed in this tab as shown in Figure 4.9. The details of the displayed item are categorised into sections. These sections can be collapsed or shown by clicking the header. Depending on the displayed item type, several less important sections will be collapsed by default. This saves display space and ensures the user will not have to scroll down to see each available category.



Back

Commit

Author: Gregor Zurowski
Timestamp: 23.03.2014 23:17

Message

CAMEL-7321: Fix inconsistent synchronization to prevent freezing of JcrConsumer Signed-off-by: Gregor Zurowski <gregor@zurowski.org>

Files

pom.xml

JcrConsumer.java

Figure 4.9: Details Tab showing commit details.

The detail view is designed to closely interact with the timeline visualisation. Users are able to navigate back and forth using different ways. A possible case is that the user is not satisfied with the information shown on the context pop-up and chooses to click the details button. This will cause the details tab to display detailed information regarding this item organised in several categories. The detail view does not only show static information. The user is able to interact with some data items by clicking it which will cause the display of a new item in the details view or have an impact on the timeline visualisation. For example the click on a date will cause the timeline to centre on this event. This is helpful if the event is out of view as this is commonly the case with issue events. One part of the issue can be outside the displayed time frame and a click on the desired date will instantly navigate to this event.

The details view provides links to other data entities for several event types. This is the case for issues, which have a list of associated commits. Another example are patterns, which feature a list of instances. If the user clicks on items in these lists, he is able to reload the detail view showing details of this item. In this situation the back button on top of the displayed content comes in handy. The user will be able to navigate back and forth trough the details view without even using the timeline. The event whose details are currently displayed in the detail view is marked on the timeline with a light blue colour. If a user has navigated through the details view he will always be able to recognize it because of this unique selection colour.

The information available about each item type as well as navigation functions are described below. Each item type features a main section at the top containing the most important data. To be able to identify the currently displayed item type, it is featured in the header of the main section.

Commits

Data sections (as shown in Figure 4.9):

- **Commit**
  The main section contains *Author* and *Timestamp* fields.

- **Message**
  This section contains the commit message. To be able to track the commits for an issue, the issue-id is usually specified in the commit message. In Figure 4.9 we can see the issue-id 'Camel-7321' in the beginning of the message. Direct navigation from commits to associated issues is currently not supported, but using the search function the issue could be found quickly if needed.

- **Files**
  This section displays the involved files in list form. Changed files are displayed in the usual grey colour. Added files appear green and deleted ones red. If the user needs specific information about the source code changes, he can click any file in the list and the code screen described in Section 4.4 will display it.

Issues

Data sections:

- **Issue**
  The main section contains *Summary*, *ID*, *Type*, *Priority*, *Status* and *Resolution* fields.

- **People**
  This section contains *Reporter* and *Assignee* fields.

- **Dates**
  This section displays a list of the events of the particular issue. Each entry features an event type and date. The event types reflect the ones displayed on the timeline. A click on a specific date will centre the main view of the timeline on the event. This is very useful to quickly navigate through the event history.

- **Description**
  This section contains the issue *Description* field.

- **Commits**
  If an issue has associated commits, they are listed in this section. Commits are linked by putting the issue-id in the message as described before. The list features date and author of each commit. The user has the choice to click on the commit date to navigate to it on the timeline or click the whole item to reload the detail view with the information about the commit.

- **History**
  This section contains the history of every update ever made to this particular issue. Individual entries feature date, author and the old and new field values. If an update event symbol is displayed on the timeline, the changes will be visible in the history section.

Builds

Data sections:

- **Build**
  The main section contains *Full Name*, *Number*, *Built-on*, *Result*, *Duration* and *Timestamp* fields.

Developers

The developer is a special data entity which is not an event featured on the timeline. More information about this topic is given in Section 4.3.5.

Data sections:

- **Developer Details**
  The main section provides information about the name and email addresses of the developer. Additionally, it contains information about overall and recent activity as described in the Section 4.2.2. It is also possible too look up the severity rating, without navigating to the overview screen.

- **Involved Patterns**
  This section contains a list of patterns the developer is involved in. The list items contain information about the pattern name, the number of found matches as well as severity ratings. Clicking a list item causes the detail view to be reloaded with the details of the pattern.

Patterns

The pattern is a parent entity which is represented by pattern instances on the timeline. More information regarding patterns is given in Section 4.3.6.

Data sections:

- **Pattern**
  The main section contain *Name*, *Type* and *Severity* fields.

- **Description**
  This sections contains the user defined pattern description.
- **Instances**
  This section lists all the pattern matches which are called instances in this application. The list items feature start and end date, which can be clicked for navigation purposes. When clicking the list item, the detail view reloads with the patterns instance specific details.

Pattern Instances

Pattern instances are a single pattern match and are therefore part of a pattern entity described above. When the pattern is shown in the timeline, its actually the set of all pattern instances which are displayed. This means any interaction with the pattern displayed on the timeline is with specific instances and not the general pattern entity.

Data sections:

- **Pattern Instance**
  The main section contains *Pattern Name*, *Pattern Type*, *Pattern Type*, *Begin Date* and *End Date* fields.

- **Events**
  This section lists all events which are involved in the pattern instance. This can be any event displayed on the timeline. The list items contain some event specific data. On click on a list item, the detail view is reloaded with the details of the particular event.

4.3.5 Developers

The developers tab of the controls section is active by default when initialising the timeline. It features a list of developers which have contributed to the project with at least one commit or had at least one issue assigned to them during project history. The developer entity is created in the back end by merging the user information contained in the source code repository and the issue tracker [Hil14]. If the developer uses different names and email addresses he wont be merged into a single entity and may show up twice. For large projects, the list of developers can be quite extensive. The list is sorted by activity and shows the last active developer first. If a specific developer needs to be found, the text box above the list can be used to filter the developers by name.

Each list entry is expandable as demonstrated in Figure 4.10. The expanded list item shows the email addresses used by the developer as well as the number of commits and issues. In addition, there is the possibility to highlight or filter both event types individually. If the highlight check box is ticked, the specific event type will be highlighted in the timeline. The highlight colour can be specified by clicking through the available colours in the selector box right behind the check box. The highlighted items will show up in the main view of the timeline as demonstrated in Figure 4.11. The example shows commits and issues in different colours. This makes only limited sense for the same developer, but it is a powerful tool to compare the activity of multiple developers. The highlighting is also performed in the second layer visualisation of the timeline. This is helpful to localise the commits of an individual or multiple developers.

While the highlighting function enables the analysis of events in comparison to all other present events, the filter function removes all events which are not filtered. It is possible to combine filters across different developers and for example only display the activity of two developers on the timeline. Highlighting is still possible when using filters. Builds and metrics are not affected by the filters and always visible.

If a user has applied filters or highlights across multiple developers, he may want to use the buttons to remove all filters or highlights in the static control box below. This is especially useful if the developer list is extensive and the user does not recall which developers where involved.

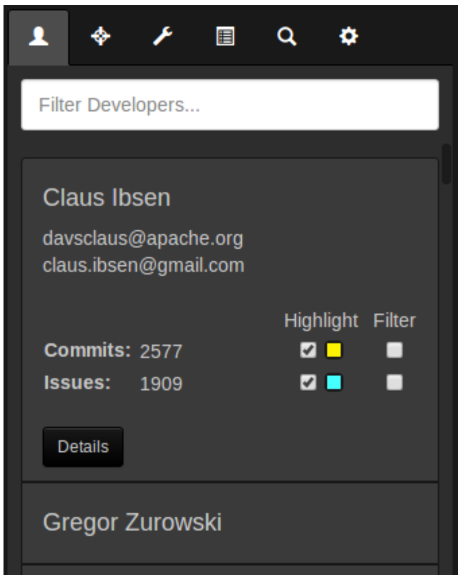


Figure 4.10: Developers Tab.

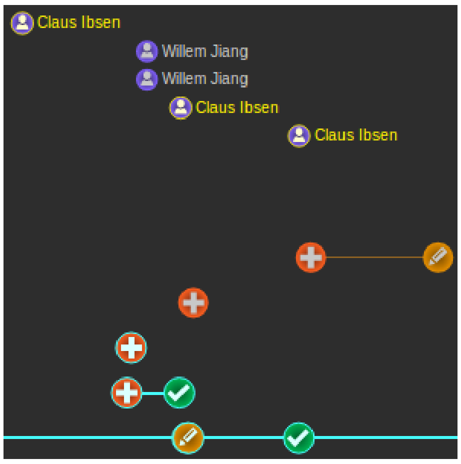


Figure 4.11: Event highlighting.

4.3.6 Patterns

The ability to display patterns supports the stakeholder in identifying situations of interest. The reasons why a pattern may be used is dependent on the project and its development environment as well as the stakeholders using the application. The following sections introduce the available pattern types and how they are visualised on the timeline.

Pattern Types

The SQA-Timeline contains a set of predefined pattern types. These patterns can be interesting for various different reasons. Certain patterns can, for example, be used to detect convention or policy breaches. Others might show critical situations which are known to lead to errors. Depending on stakeholder and project, the requirements for patterns can be very different. Lead developers and project managers are probably going to focus on different aspects than a one time contributor. Patterns can be set up and configured in an editor in order to match these individual needs.

An initial set of patterns and its exchange format was agreed upon with the developer of the back end project. The SQA-Timeline is responsible for the visualisation and configuration of patterns. The recognition and feasibility is part of the back end project SQA-Pattern [Hil14]. The defined pattern types and potential use cases are described in the following list:

- **Cluster Pattern**
  The cluster pattern allows the identification of a certain density of events. The user specifies a time interval and the number of events for the pattern to match. Various different event types involving commits, builds, and issues are available and can be combined in a single pattern. A potential use case can be the identification of days on which particularly many issues were created. An irregular increase in issue creations could be related to a single incident and would need the attention of the development team. It could also be used to track the activity of the project which involves identifying peak commit times.

- **Event Pattern**
  The event pattern provides a simple way to identify certain events such as a build failure or when an issue is reopened. If a type of event is hard to find or its occurrence is of a certain importance for the development process, it can make sense to create a pattern. It enables the user to keep track of events and to quickly navigate to occurrences.

- **Regular Expression Pattern**
  This pattern executes regular expressions on specified data fields. Supported event types are issues and commits with various fields such as description, message, or author. The specified regular expression will be tested on the target field of every event. Two matching modes are available, the normal mode detects a pattern match when the regular expression is satisfied. The complement mode matches when the expression is not satisfied. If a development process relies on a certain format or content of a field, this pattern can be used for validation. A prominent use case for this pattern would be to find commits which do not have a corresponding issue assigned to them. Assigning commits to issues using a certain format in the commit message is a convention for many projects using an issue tracker. It helps to keep track of which code has been changed for a particular issue. The regular expression pattern can now be applied to check if the issue-id is added to the commit message in the right format in order to be properly linked to the issue tracker. The regular expression pattern is very versatile and has various other use cases like checking if an issue description has a certain length. If an issue is not described properly and does not include the right examples it could result in duplicates and misunderstandings.

- **Issue Contributions Pattern**
  This pattern provides the means to track issues with a certain amount of associated commits. The pattern finds a match, if a specified number of commits is overstepped for a particular issue. An optional time interval can be set to only match the pattern if the commits happened within a certain amount of time. Another option is to find only commits by the same author. If an issue has several associated commits, this could indicate that the fix did not work as intended and needed additional work. Another reason could be that the scope of the issue is too large and should be broken down into several issues. Chances are high that some problems occurred during the solving process of the issue which might be worth a look of a lead developer or project manager. If the issue is still open, further action can be taken. If it is closed now, valuable learnings could be extracted from it.

- **Time Pattern**
  The time pattern can be used to identify events which happened within a specified interval before or after a certain event. A possible use case is the identification of commits which happened after an issue was resolved. This indicates that the fix was not properly tested before resolving and needed further work. A quick check of the issue state and if it has been reopened or properly fixed in the meantime might be necessary. Another use case for the pattern could be the identification of commits which happened before a build failure. Chances are high the error which caused the build failure is caused by one of these commits.

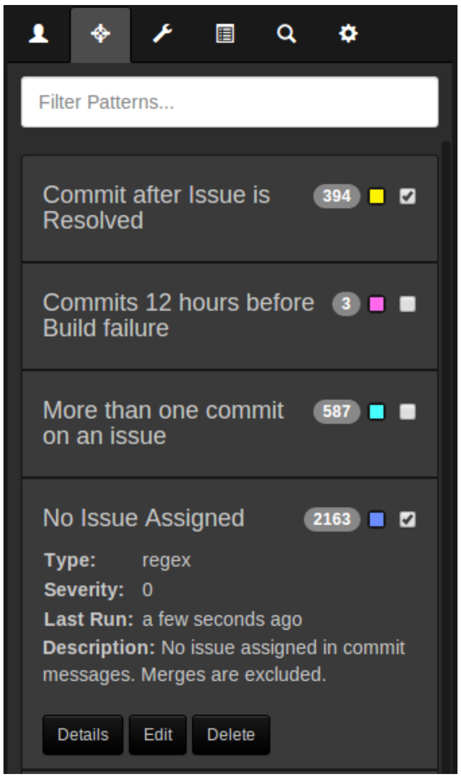
- **Metric Pattern**
  This pattern is used to identify if a source code metric falls below or exceeds a certain value. Multiple metrics can be combined in one pattern to identify more detailed situations. The pattern is pretty straight forward and can also be used for policy enforcement. It could for example be used to check if the unit test success rate falls below 100%. Which thresholds are to be set and which metrics to test is very project and policy dependent. The metric pattern allows to keep an eye on the important metrics and identify the reason for value changes.

Pattern Visualisation

The pattern functionality has two dedicated tabs in the controls section of the timeline screen. One tab provides a pattern list and the other one implements a pattern editor to create and edit patterns. The pattern tab shown in Figure 4.12 displays a list of patterns, which have been created for this project. The list can be filtered using the text box above. Each list item shows the name of the pattern as well as the amount of matches found in the entire data set. The individual items are expandable which adds some more information as well as buttons to show details, edit, or delete the pattern. Each item has a check box, which displays the found pattern instances (matches) on the timeline. Positioned on the left side of the check box is a colour selector, which can be clicked to change the colour the pattern appears with in the timeline.

The pattern editor tab shown in Figure 4.13 enables a user to create or edit patterns. The editor is dynamic and depending on the selected type, different data can be entered. For some patterns, such as the metric pattern shown in Figure 4.13, it is possible to add an unlimited amount of rules by using the *Add Rule* button. When a user tries to save a pattern using the *Save* button, the application will perform a client side check if all the required data is entered correctly. If this is the case, the pattern is transmitted to the server and executed. In its current configuration, the server will execute all the patterns and pass the results back to the front end. Depending on amount of patterns and types it can take up to a minute to rerun the algorithm. As soon as the new updated data is available on the server, the pattern list will be updated on the front end. The *Last Run* field indicates how old the shown pattern data is. If the pattern has never been run or an error occurred, an orange or red frame will be shown around the list item.

When the checkbox in the pattern list is checked, the respective pattern will be displayed in the timeline. Figure 4.14 shows the timeline displaying two patterns. Found instances are visible in both, the main and overview layer. Depending on the pattern type, an instance may mark a point in time or a duration as visible in the example. The blue pattern represents the *No Issue Assigned* pattern which marks commits that have no issue assigned in the commit message. The pattern instances are represented by a thin line in the main layer, respectively a dot in the overview. The *Commit after Issue is Resolved* pattern however involves multiple events in one instance, namely at least one commit and one issue. Its instances are therefore represented by a duration which en-


Filter Patterns...

Commit after Issue is Resolved 394 🟨 ✓

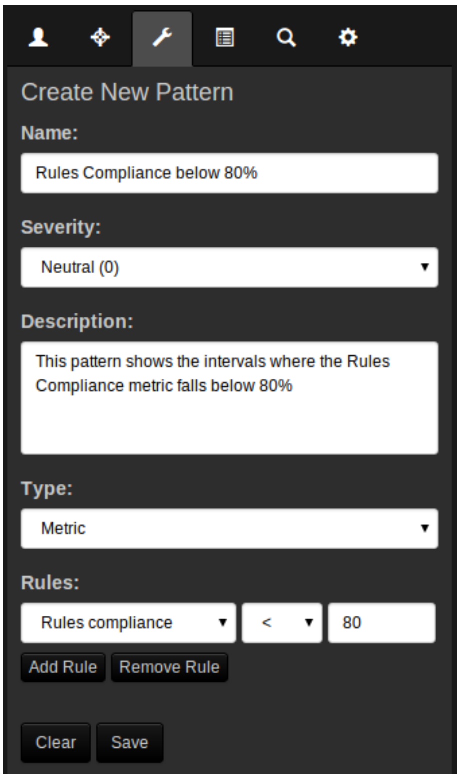
Commits 12 hours before Build failure 3 🟪 ☐

More than one commit on an issue 587 🟦 ☐

No Issue Assigned 2163 🟦 ✓

Type: regex
Severity: 0
Last Run: a few seconds ago
Description: No issue assigned in commit messages. Merges are excluded.

Figure 4.12: Pattern Tab.


Create New Pattern

Name: Rules Compliance below 80%

Severity: Neutral (0)

Description: This pattern shows the intervals where the Rules Compliance metric falls below 80%

Type: Metric

Rules: Rules compliance < 80

Figure 4.13: Pattern Editor Tab.

closes all the involved events. This should make it easier for the user to spot the temporal scope of the pattern instance. The example shows four instances of the blue pattern and one instance of the yellow one in the main layer. The instances are interactive as any other event and a pop-up is displayed on click. The pop-up provides basic data about the found instance as well as the possibilities to show more details in the detail view and to highlight the involved events.

The visualisation of patterns is designed to be of generic nature. The same visualisation is used for any pattern type. This way it was possible to include a wide range of pattern types with the possibility to easily add new types in the future. When designing the pattern visualisation, it was hard to foresee how it will look like with actual data. Each project has different properties combined with the many possibilities offered by the pattern configuration. Two identified problems are the display of overlapping instances and the display of very long instances. Overlapping pattern instances can cause confusion which element belongs to which instance. To prevent this, the user may use the event highlighting function which will clearly indicate the events which are involved in the particular instance. When dealing with long pattern instances, the user may use the details view and the provided events list. Using this list he can easily navigate between involved events without having to scroll long distances. Using the detail view the user is able to navigate all pattern instances and the involved events without even using the timeline. Given this alternative, the user may choose the navigation path depending on the circumstances provided by the project properties and pattern definition.

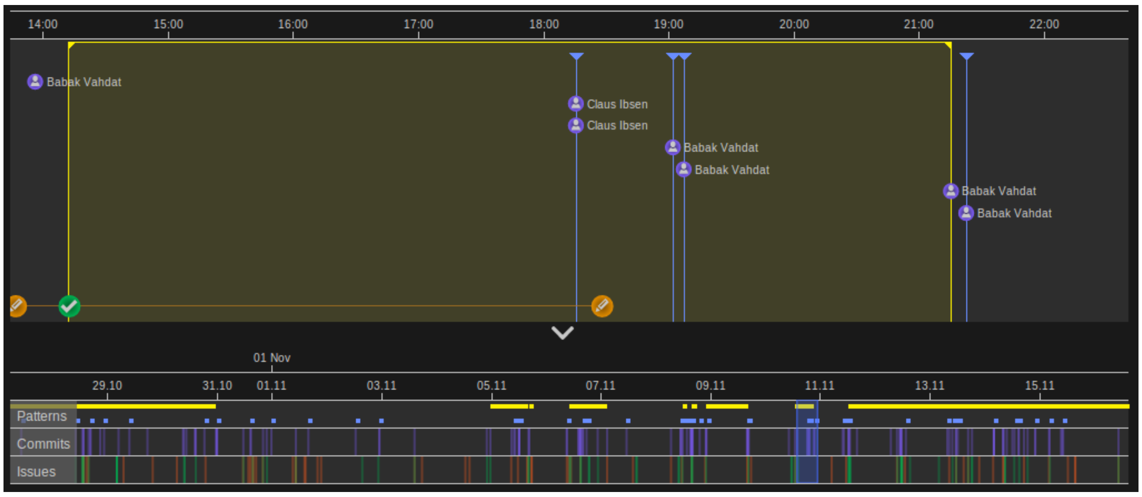


Figure 4.14: The timeline displaying two patterns.

4.3.7 Search

Located in the timeline controls is a tab dedicated to searching the project data. To be able to look up events based on different search properties adds substantial value to the application. The search function may not only be used to find a single item, but can also filter items by certain criteria as described later. The following event types may be looked up via the search function:

- Commits
- Issues
- Builds

Each event type has its own list which is initially collapsed. When a search is conducted, the results will show up categorised by type in the mentioned lists. A number next to the list header indicates the amount of found results for the specific event type. The search is executed entirely in the front end. When loading the project, the search indexes are created, which causes a wait time of a few seconds until the search is ready. As soon as the indexes are created, it offers good performance and has an almost immediate response time.

For the time being, the search function only supports fields which are available in the timeline data. Large fields like the commit message and the issue description are not included, because of performance reasons. A full text search over all fields would require a server side approach. Nevertheless, the search includes the most important fields. A valuable aspect of the search is the ability to look up dates in almost any format. The search terms ’13.03.2014’ or ’13.3.14’ will both lead to the same result. When looking up a date, it will display all events which have happened that day. Apart from dates the search also allows text input, as for example issue summary, commit author, or build number. The search terms do not have to match exactly and when searching for the keyword ’tests’, it will still find issue summaries containing the word ’test’. Search terms and dates can be combined freely to further reduce the amount of found elements.

Each search result contains some basic data about the event. For example a found build will feature the number, the result and the date. Dates are displayed as clickable links and allow a quick navigation to the event on the timeline. By selecting the header of a search result, the user opens up a context menu containing further functionality. For any event, the user is given the possibility to open the details view described in Section 4.3.4. When opening the context menu on a commit, it is also possible to directly navigate to the code screen to look at the source code changes.

4.3.8 Audio Playback

The SQA-Timeline is primarily a visualisation of data and therefore provides a visual experience for the user. However, the timeline also includes an experimental feature to enable audio playback of the project history. The approach to turn software engineering data into an audible experience is not a new idea and has been implemented before. Boccuzzo and Gall presented their approach as an extension to their tool *CocoViz* in 2009 [BG09]. The tool visualises the structure and evolution of a code base. The visualised content is supported by an ambient audio concept which is defined by a mapping of the data to different sounds and audio parameters. McIntosh et al. introduced an approach to map repository changes to musical motifs in 2014 [MLH14]. Their goal was to create a musically pleasing experience for the user, which required some manual work and was not completely automated.

The approach implemented in SQA-Timeline offers the possibility to play a sound effect for most events visualised on the timeline. Commits are mapped very simple and just play one sound effect when they are occurring. Issue events are distinguished as they are in the visualisation. Each event has its own sound mapping. The sounds for builds are dependent on the result. For example successful builds may have a different sound than failed ones. If a pattern is displayed on the timeline, the beginning and end of an instance are also mapped to a sound.

The playback is controlled using the audio buttons on the bottom of the controls section. Pressing the play button will start the playback at the beginning of the currently visualised content in the main view. The current position is marked by a red line including a label showing the exact date and time. Using the other available control buttons the playback can be paused or stopped. If no action is taken, it will continue until the end of the data set. The playback will always advance in the same pixel per time pace. The zoom level of the main view determines the density of events. This gives the user the possibility to adjust the playback to different event densities.

4.3.9 Options

To make sure the timeline is fit for various project environments as well as different stakeholders, some options are available in a dedicated tab of the timeline controls section:

- **Displayed Content**
  The content of the timeline visualisation is customisable. Each data category representing a lane in the overview layer can either be displayed or hidden in both visualisation layers individually. It is for example possible to turn off issues in the main layer but still display them in the overview or the other way around. If a data source is not available it makes sense to turn it off in the overview layer in order to save space. The main layer will automatically adjust the content anyway in this case. The ability to hide and show data easily allows stakeholders to adjust the timeline to their personal needs.

- **Displayed Metrics**
  Featuring a list of available metrics for the project, the user is given the possibility to select which ones he wants to display. If a large amount of metrics is available, the visualisation might get overcrowded if they would all be displayed at the same time.

- **Filter Inactive Issues**
  By default, all the issues which do not have an event on the current time frame of the main view are not displayed. This means if an issue is open during the time frame but has no events it will be hidden. Otherwise a line would be shown but no events. This option is crucial for large projects which have many issues open at the same time. Inactive issues would fill up most of the main view but not contain any events. The filtering of inactive issues can sometimes be confusing when scrolling along the line of an issue. As soon as an inactive time frame is encountered, it will be hidden. This option gives the user the choice to turn off the filtering and accept a larger amount of issues displayed simultaneously.

- **Extended Issue Overview**
  In the overview layer all issue events are displayed mixed together in one lane by default. It is not possible to follow one single issue over its history. By turning the extended issue overview on, the overview layer will show the same visualisation for the issues as in the main view. The extent of an issue will be clearly visible, but with a large amount of issues the readability as well as the performance decreases drastically. It also requires more space than the default view which might be a problem on smaller screen resolutions.

All the changes performed in the options menu have an immediate effect. It is therefore possible to use the options to adjust the visualisation depending on situational needs. If a user is only interested in commits for the time being, he can quickly turn off the other content to avoid distractions and free up display space. The same is valid for the metrics he might wants to see. He can go through the list and just turn on the ones he is interested in.

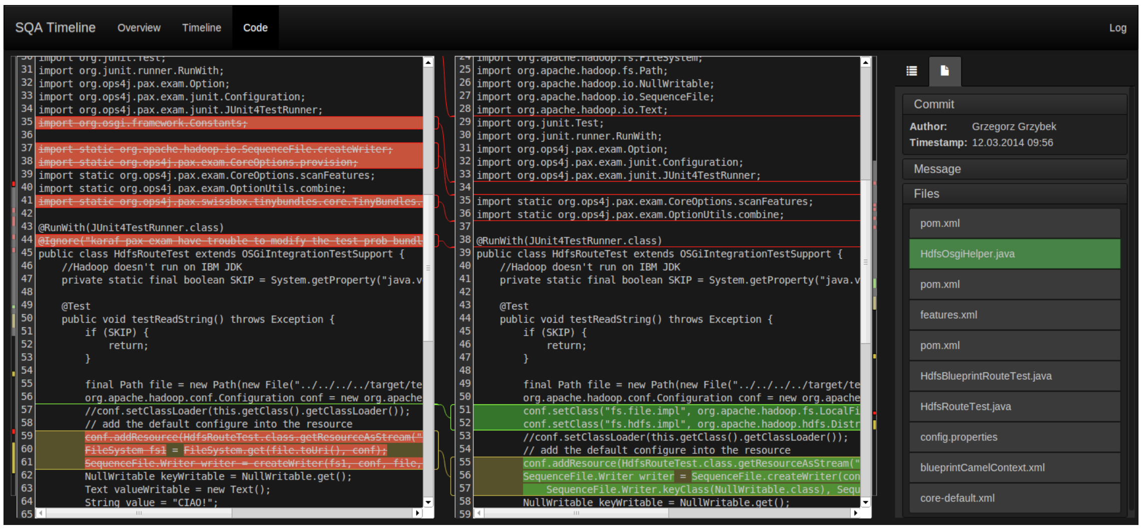
In its current set up, SQA-Timeline always initialises the default options on start up. However, in the future it would be feasible to save user specific settings on the server which will be reloaded on login. In this case each user could specify his own preferred view and its state would persist across sessions.

4.4 Code Screen

The code screen is the third screen of the SQA-Timeline application. It focusses on commits and the associated source code changes. The code screen offers a diff view for each file which has been changed in a commit. Two control tabs are available to the user. The first tab shows a list of the last hundred commits. It offers a fast entry to the screen if a developer wants to analyse a recent commit without having to look it up on the timeline. The second tab displays the details of the currently selected commit. These include author, timestamp, message, and a list of changed files. A green or red list entry means the file has been newly added or deleted respectively. If a list item is clicked, the diff viewer will show the old file on the left and the new file on the right. Additions of full lines are marked green, and deletions are red. If a section has only been changed partially, it will be marked yellow. The inline changes are again colour coded in red and green. When scrolling one side of the diff view, the other side is scrolled to the same part in the code to make changes visible quickly.

Figure 4.15 shows how the code screen looks like in action. The control tab displays the information regarding the selected commit. The shown commit could have been chosen on the timeline or the recent commit list of the code screen. The diff viewer displays the changes of a file which has been selected in the files list. Changes are clearly marked and small lines connect the associated markings across the two windows to avoid confusion.

The code screen itself does not provide any advantages compared to tools offered by other coding platforms like GitHub or Bitbucket. They have similar ways to analyse code changes of a commit. The reason why it has been included in SQA-Timeline, was to provide information across all information granularities from an overview down to the source code level. This way, the user will not have to leave the platform if he needs information about the source code.



```
import org.junit.runner.RunWith;
import org.ops4j.pax.exam.Option;
import org.ops4j.pax.exam.junit.Configuration;
import org.ops4j.pax.exam.junit.JUnit4TestRunner;
~~import org.osgi.framework.Constants;~~

~~import static org.apache.hadoop.io.SequenceFile.createWriter;~~
~~import static org.ops4j.pax.exam.CoreOptions.provision;~~
import static org.ops4j.pax.exam.CoreOptions.scanFeatures;
import static org.ops4j.pax.exam.OptionUtils.combine;
~~import static org.ops4j.pax.swissbox.tinybundles.core.TinyBundles.~~

@RunWith(JUnit4TestRunner.class)
~~@Ignore("karaf pax exam have trouble to modify the test prob bundl~~
public class HdfsRouteTest extends OSGiIntegrationTestSupport {
    //Hadoop doesn't run on IBM JDK
    private static final boolean SKIP = System.getProperty("java.v

    @Test
    public void testReadString() throws Exception {
        if (SKIP) {
            return;
        }

        final Path file = new Path(new File("../../../../target/te
        org.apache.hadoop.conf.Configuration conf = new org.apache
        //conf.setClassLoader(this.getClass().getClassLoader());
        // add the default configure into the resource
        ~~conf.addResource(HdfsRouteTest.class.getResourceAsStream("~~
        ~~FileSystem fs1~~ = ~~FileSystem.get(file.toUri(), conf);~~
        ~~SequenceFile.Writer writer = createWriter(fs1, conf, file,~~
        NullWritable keyWritable = NullWritable.get();
        Text valueWritable = new Text();
        String value = "CIAO!";
```

```
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.SequenceFile;
import org.apache.hadoop.io.Text;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.ops4j.pax.exam.Option;
import org.ops4j.pax.exam.junit.Configuration;
import org.ops4j.pax.exam.junit.JUnit4TestRunner;

import static org.ops4j.pax.exam.CoreOptions.scanFeatures;
import static org.ops4j.pax.exam.OptionUtils.combine;

@RunWith(JUnit4TestRunner.class)
public class HdfsRouteTest extends OSGiIntegrationTestSupport {
    //Hadoop doesn't run on IBM JDK
    private static final boolean SKIP = System.getProperty("java.v

    @Test
    public void testReadString() throws Exception {
        if (SKIP) {
            return;
        }

        final Path file = new Path(new File("../../../../target/te
        org.apache.hadoop.conf.Configuration conf = new org.apache
        conf.setClass("fs.file.impl", org.apache.hadoop.fs.LocalFi
        conf.setClass("fs.hdfs.impl", org.apache.hadoop.hdfs.Distr
        //conf.setClassLoader(this.getClass().getClassLoader());
        // add the default configure into the resource
        conf.addResource(HdfsRouteTest.class.getResourceAsStream("
        SequenceFile.Writer writer = SequenceFile.createWriter(con
            SequenceFile.Writer.keyClass(NullWritable.class), Sequ
        NullWritable keyWritable = NullWritable.get();
```

Commit
Author: Grzegorz Grzybek
Timestamp: 12.03.2014 09:56

Message

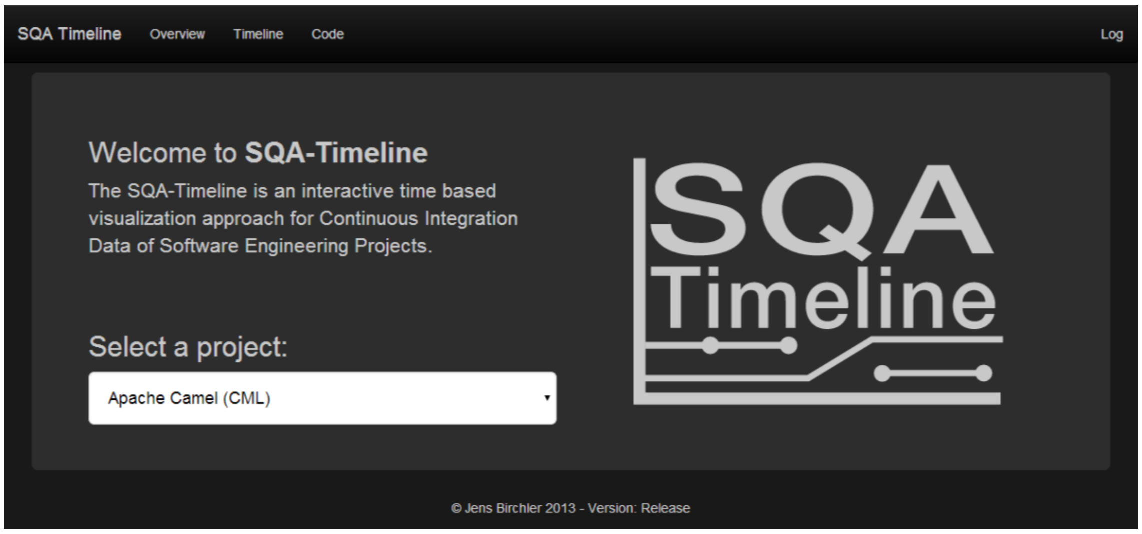
Files

- pom.xml
- HdfsOsgiHelper.java
- pom.xml
- features.xml
- pom.xml
- HdfsBlueprintRouteTest.java
- HdfsRouteTest.java
- config.properties
- blueprintCamelContext.xml
- core-default.xml

Figure 4.15: The SQA-Timeline Code Screen (Apache Camel Project).

4.5 Project Selection

When loading the application, a start page with a login is displayed. The passed credentials are transmitted to the server and verified. The projects are only initialised if the login is successful. Currently the login is only a security feature, however this could be extended to persist user specific options and views in the future. After a successful login, a page as shown in Figure 4.16 is displayed. All the available projects can be selected here. On selection of a project, the application automatically initialises the individual screens. The initialisation process happens instantly on selection and not on navigation to a specific screen. This ensures a smoother navigation experience and minimises wait times for the user. Available projects are received dynamically from the server. The application currently features several projects of various sizes. The scope of a project may vary and depends on the back end configuration. It is currently possible to limit the amount of data by setting a start date. All events before the start date are discarded. This limits the amount of traffic and required processing power. The SQA-Timeline simply displays the data which has been put together by the back end. Breaking down projects into sub-projects or combining multiple projects in a single view would be possible if supported by the back end.


SQA Timeline Overview Timeline Code Log

Welcome to **SQA-Timeline**

The SQA-Timeline is an interactive time based visualization approach for Continuous Integration Data of Software Engineering Projects.

Select a project:

Apache Camel (CML)

© Jens Birchler 2013 - Version: Release

Figure 4.16: The SQA-Timeline project selection page.

4.6 Data Sources

The SQA-Timeline currently visualises data elements from the following sources:

- Source Code Repository (Git[3], SVN[4])
- Issue Tracker (Jira[5])
- Continuous Integration Platform (Jenkins[6])
- Code Quality Platform (SonarQube[7])

The list features the software engineering tools which are currently integrated into the visualisations of SQA-Timeline. The tools listed in brackets are the actual data sources supported by the back end so far. The exchange format between back end and SQA-Timeline is specified according to the needs of the respective visualisation. It would therefore be possible to extend the support for additional data sources which fit the type of one or multiple tools featured in the list. If the required data is available, there would be no visual impact for SQA-Timeline. With Git and SVN there are already two Source Code Repositories supported. However, there is no visual difference how the data is presented on the front end application. In the future it would be possible to support more tools like GitHub Issues without making major changes to the front end.

The data displayed by SQA-Timeline is not real time data. Each time the front end is loaded in the browser, it will fetch the most recent data available on the server. However, the back end does not forward this request to the data sources but will use preloaded data. Updating the data in the back end from the different sources is quite costly and always gets the whole available data

[3]http://git-scm.com
[4]http://subversion.apache.org
[5]http://www.atlassian.com/en/software/jira
[6]http://jenkins-ci.org
[7]http://www.sonarqube.org

set. This process takes up to 15 minutes depending on project size and connection [Hil14]. How frequent these updates should happen depends on how up-to-date the data in the application needs to be and how the traffic limits are specified. If the application is only used as a review tool, daily updates might be enough. If the tool is used by active developers, they will mostly be interested in the very recent data. This means the updates would need to be close to real time.

Chapter 5

Implementation Details

The following chapter covers several implementation details of SQA-Timeline. The scope of SQA-Timeline involves only the front end application. The back end was developed in close collaboration but is part of a separate project called SQA-Pattern [Hil14]. The contents of this chapter will not cover every detail of the implementation, but rather focus on some notable design choices. The application is built with modern web technologies. The visualisation of content is achieved using HTML5, CSS3 and SVG. The logic of the application has been programmed in JavaScript. A closer description of the used technologies and why they have been chosen can be found in Chapter 3.5.

5.1 Modular Design

For more complex web applications like SQA-Timeline, it is crucial to introduce a certain structure into the code. JavaScript is a script language and was initially not designed to create larger applications. To achieve proper encapsulation of logical code elements, there exist various design patterns in JavaScript [Osm12]. SQA-Timeline uses a module-based pattern, which enables the creation of modules using a constructor. Each module contains its own name space. Functions and variables can either be exposed publicly or kept private inside the module. Most screens of the application are based on multiple modules. Following list features all modules of the application and gives a short description of its purpose:

- **Overview Chart Module**
  This module is used to initialise the *events* tab visualisation of the overview screen. It creates an interactive bar chart featuring a summary of events. It is also responsible to update the control tabs with the currently selected event type.

- **Overview Bubbles Module**
  The *developers* and *patterns* tab of the overview screen are initialised by this module. It is responsible for the bubble cloud and treemap representation of content as well as the display of details in the controls section.

- **Timeline Visualisation Module**
  The timeline visualisation is created by this module. It is the biggest module of the application and contains all logic to draw and update the timeline.

- **Timeline Controls Module**
  Because of the complexity and size of the timeline module, the logic for the controls section was encapsulated in its own module.

- **Code Controls Module**
  This module contains the logic for the controls section of the code screen. Since the diff view is created by an external library it is the only module of the controls section.

The modules are responsible for one part of the application and expose an API which is used to control the represented visual part. Each screen contains a main script which initialises the modules. Most modules are only initialised once, only the *Overview Bubbles Module* is instanced twice because the visualisation is featured on two tabs. The *Timeline Visualisation Module* and the *Timeline Controls Module* both keep a reference of each other because they interact frequently. However, it would only require minor effort to completely detach the visualisation and reuse the module as an external library in another application. The changes would involve a few event handlers needing a new target function.

When a module is initialised, a settings object is passed to the constructor. This object contains the configuration parameters for the specific instance of the module. A default value will be used if a parameter is missing, hence the settings object only needs to specify deviating parameters. The following example shows how the initialisation of an *Overview Chart Module* works when specifying all the available parameters. In the actual implementation, most of these parameters are left at its default value.

```
var moduleInstance = new ChartModule(
   {
      container: 'body', //the target node in the DOM
      width: 1000,
      height: 720,
      rangeSelectorHeight: 40,
      rangeSelectorTextMargin: 120,
      chartPadding: 100,
      startDate: 0,
      margin: {
         top: 50,
         right: 50,
         bottom: 50,
         left: 50
      },
      data: {} //object containing the visualised data
   }
);
```

Listing 5.1: Example module initialisation.

Most settings in the example shown in Listing 5.1 concern the sizing and spacing of the visualisation. However, the visualised data as well as a node selector which specifies the location of the graph are also passed via the settings object. The *Timeline Visualisation Module* contains 43 configurable parameters which allows for customisation of appearance and display options without changing the source code of the module.

5.2 Screen Navigation

Each screen of SQA-Timeline is represented by a separate HTML page. This ensures the encapsulation of each screen and gives the possibility to easily add or remove screens in the future. A drawback of this concept is that a browser is only able to display one page at a time in a single tab. Navigation between screens would result in a reinitialisation of the new page which causes a wait time. Another drawback is that the state of the origin is lost and if the user navigates back to the screen it will be in its initial default state.

To work around the mentioned drawbacks, a parent document called `app.html` was introduced. This page is loaded once when initialising the application. Login and project selection are part of this page. When initialising a project, the application pages will be loaded as child pages of `app.html` using an inline frame (`iframe`). The inline frames are initially hidden and the specific frame is shown when the user navigates on an application page. Using this concept the user is able to navigate between screens quickly. When going back to a previously visited page, it will be shown in the exact same state as when it was left. There is also the advantage that the pages are initialised in the background as the application is loaded. After a few seconds there wont be any more load times until the user decides to look at another project. However, the approach also has its drawbacks. The application is considered to be a single page by the browser, which means the back and forward buttons will not work between screens. Furthermore, the URL will stay constant during navigation, which has the effect that specific screens can not be shared by URL or saved as favourites.

5.3 External Libraries

SQA-Timeline is built using various libraries and a UI framework. The purpose of each library, as well as how it was used in the application, is described in the following list:

- **Bootstrap**[1]
  Bootstrap is a popular UI framework for the web. It includes many UI components to create a modern and intuitive user experience. It is basically a large collection of CSS style classes supported by a JavaScript library. The styling of each component is completely customisable to fit the needs of the application. Bootstrap was used in SQA-Timeline for the header and control tabs design. However, the main visualisations of the three screens do not rely on bootstrap.

- **D3**[2]
  As described in Chapter 3.6, D3 is a data driven graphics framework for JavaScript. With its low level approach, the full feature set of modern web standards like HTML and SVG can be used to create visualisations which run in a web browser. D3 was used in SQA-Timeline to create the timeline as well as the visualisations on the overview screen.

- **jQuery**[3]
  The jQuery library provides a set of tools which performs tasks often used in web devel-

[1] http://getbootstrap.com
[2] http://d3js.org
[3] http://jquery.com

opment. Its primary features are DOM selection and transformation, but it also provides support in other areas like AJAX. The features of jQuery are used frequently in all the sections of SQA-Timeline.

- **Lunr**[4]
  Lunr is a library which enables full-text search in a JavaScript environment. By creating a search index on desired fields, it offers a fast client-side search experience. Lunr was used in SQA-Timeline to create the search features on the timeline screen. The search functions are divided into multiple threads to prevent UI blocking. A more detailed description regarding this topic can be found in Section 5.6.

- **Mergely**[5]
  Mergely is a fully featured diff viewer. It takes two files as input and calculates and visualises the differences as featured on the code screen of SQA-Timeline. The library itself is built on CodeMirror[6]. By customising the included CSS file, the styling of the diff viewer was adapted to match the rest of the application.

- **Crossfilter**[7]
  Crossfilter enables fast filtering of data sets. It creates an index on the filtered attribute to increase performance. In SQA-Timeline it was used in the overview section to filter the data when moving the time range selector.

- **Moment.js**[8]
  This small library simplifies the handling of dates in JavaScript. Calculating dates as well as parsing and formatting is much simpler using this library than using the native JavaScript functions.

- **Parsley**[9]
  Parsley is a form validation library. It is used to validate the data entered in the pattern editor form before it is sent to the server. It immediately marks the form elements which are missing or contain the wrong data format.

- **Less**[10]
  Less is a CSS pre-processor which extends the CSS language. Various features like variables, functions, and mixins support the development of CSS. The LESS files are compiled into CSS before it is deployed. The transformation happens either during the build process or at runtime when running the development version of SQA-Timeline. This means when running the pre-built production version of the application, the less library will not be used, since all the LESS files have already been converted to CSS.

[4]http://lunrjs.com
[5]http://www.mergely.com
[6]http://codemirror.net
[7]http://square.github.io/crossfilter
[8]http://momentjs.com
[9]http://parsleyjs.org
[10]http://lesscss.org

5.4 Client-Server Communication

The communication between back end and front end is achieved through a RESTful API exposed by the server. REST stands for Representational State Transfer and is an architectural style of client-server communication. A RESTful web service uses the HTTP protocol to perform an action on a specific URI [RR08]. In order to manipulate resources on the server, the appropriate HTTP functions are used. For example to save a pattern on the server, a PUT request with the data containing the pattern specification is sent by the client. The server will then reply with the result of the request using the correct HTTP error code. Each resource has its own URI and is invoked by the client using AJAX (Asynchronous JavaScript and XML) to asynchronously load the content using XHR (XMLHttpRequest).

The format used for the data exchange is JSON (JavaScript Object Notation). JavaScript natively provides the means to parse the format. It is therefore often used for the communication with web applications. Each web service offers a specific data format which enables the client to parse and visualise the data. The example shown in Listing 5.2 shows the data format which is received to initialise the timeline visualisation. The format specifies arrays for builds, commits, issues and metrics. To keep the example small, only one commit and one issue is shown. Each individual item contains an ID, which can be used to get additional information using another web service.

```json
{
    "builds": [],
    "commits": [
        {
            "author": "Jens Birchler",
            "id": "coma4b923d997e113d67a710cb0e185e4efcc2f7280",
            "timestamp": 1380800193000
        }
    ],
    "issues": [
        {
            "id": "issSQAPATTE-54",
            "priority": "Minor",
            "summary": "Error message when initialising application",
            "timestamps": [
                {
                    "timestamp": 1392830615000,
                    "type": "Created"
                }
            ],
            "type": "Bug"
        },

    ],
    "metrics": []
}
```

Listing 5.2: Timeline data format in JSON.

5.5 Timeline Architecture

Each code module of SQA-Timeline has its own architecture adapted to its purpose. Since the timeline visualisation module is the core part of the application, the inner architecture of the module will be described closer in this section.

When initialising the timeline visualisation module, the DOM structure is built from scratch using JavaScript. The visualisation will be inserted into a specified container node. This node has to be located in the HTML file and must not have any children, since they will all be deleted on initialisation. The complete SVG structure is built with D3 and is customisable via the initialisation settings object. Lane sizing, paddings and margins are all configurable parameters which make it possible to adapt the looks of the visualisation without changing the source code.

After the initialisation function has run, the DOM structure is ready to display data elements. The overview and main visualisation layers only show a part of the complete data set. To display the right items on each layer, a filtering structure exists which is based on four data arrays as shown in Figure 5.1. The *Data* array on the leftmost side contains the full data set. The contents of this array are never changed. Each one of the three other arrays contains a subset of the one on its left side. This means that a function filters the data with each step, reducing the number of elements based on filtering conditions. In some cases no items are removed and all of them are transferred to the next level. The first filtering step from *Data* to *Filtered Data* is performed, if a data filter is applied. Currently this is only the case when filtering items for a specific developer. If no data filter is applied, the *Filtered Data* contains the full data set. The *Overview Layer Data* and the *Main Layer Data* contain the elements which are displayed in each layer. A display filter function filters the data set according to the selected time frame. After that, the updated content will be drawn to the respective layer.

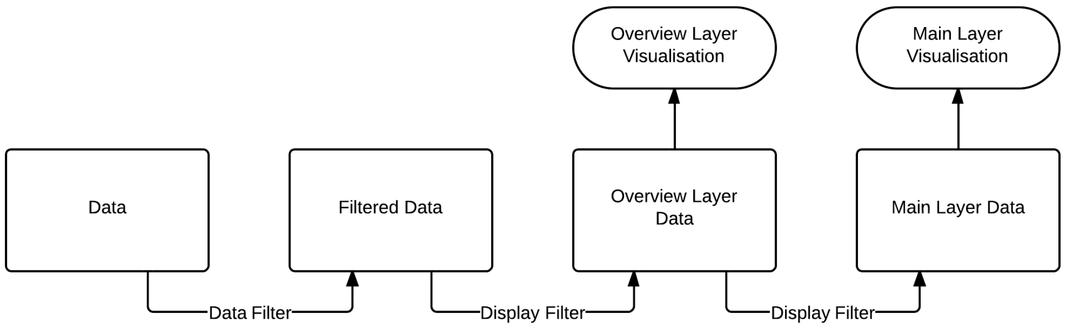


Figure 5.1: The architecture of the timeline visualisation.

For performance reasons, filtering is only performed up to the point where the parameters changed. For example if the range selector on the overview layer is moved, only the last display function is executed to update the *Main Layer Data*. This function only needs to filter the already reduced set contained in the *Overview Layer Data*. Since this update needs to happen for every pixel the selector is moved, it is crucial to achieve the best possible performance to ensure a smooth animation and instant response times. On the other hand, if the user decides to filter the displayed data, all three filter functions are executed and both layers have to be redrawn. This is a more time-consuming operation, but is not executed as repeatedly as the update of the individual layers.

5.6 Multithreading

The search feature of the timeline has been described in Chapter 4.3.7. It offers a client-side search feature for commits, issues, and builds. The search functionality is implemented with Lunr as described in Section 5.3. Each time a project is initialised for SQA-Timeline, a search index for each of the three item types is created. Depending on the size of the data set, this can take around 10-20 seconds if it is performed in a single thread. During this time the user interface would not respond, which imposes a severe restriction of the user experience. In order to work around this problem, each search index runs in its own thread which is completely independent of the main UI thread. This means the user interface remains responsive while initialising the indexes and when a search is conducted.

Multithreading has been introduced to the web by the W3C in form of the Web Worker Standard[11]. It is currently only a candidate recommendation, but is already implemented in most modern browsers. Web workers allow the execution of scripts in a separate thread. This means the main thread controlling the user interface does not have to wait for the completion of the script to perform its own calculations. This results in a non-blocking user interface while another script runs in the background. The use of multiple threads via web workers can also improve overall performance because they create a new OS-level thread which takes advantage of multi core CPUs [Hal12]. Web workers are bound to certain restrictions. They have no access to the DOM and global variables. Communication with the main thread is achieved through message passing.

When initialising SQA-Timeline with a specific project, the three web workers containing the search indexes for commits, issues, and builds are started immediately after receiving the data from the server. Each worker will create a search index based on the fields that need to be indexed without blocking the user interface. As soon as the indexing is finished, the worker sends a ready message to the main thread which will remove the displayed loading notification for the particular event type. If a search term is entered, it is sent to each individual worker which sends back the found elements to be displayed by the main thread. The advantage of using three workers for each item type is that the user may already look at the found commits while the issues are still being searched.

[11]http://www.w3.org/TR/workers

Chapter 6

Evaluation

6.1 Study Setting

In order to test the value of the approach implemented by the SQA-Timeline application, a controlled user study was conducted. A total of ten participants were divided into two equally sized groups. Both groups had to solve the same set of tasks using different tools. One group was only allowed to use the SQA-Timeline application. The other group, called the control group, were restricted to use GitHub (Source Code Repository), Jira (Issue tracker), Jenkins (CI-Platform), and SonarQube (SQA-Platform). These tools were chosen because they currently serve as data sources for SQA-Timeline. This made it possible to have both user groups solve the same tasks based on the same data. This provides the foundation to evaluate the difference between the available tools. All the tools which were part of the control group have been described previously in one of the introductory chapters.

Each group had the assignment to solve six tasks which are faced in real world development scenarios. The tasks involve different fields like Continuous Integration (CI) or Software Quality Assurance (SQA) and are mostly related to possible information needs of lead developers or project managers. The foundation of the tasks is provided by several papers describing information needs as introduced in Chapter 2.4. Why each task has been chosen and what sources it is based on will be described later in Section 6.3.2.

All the participants of the study had different experience and previous knowledge about the tools they were using. In the personal details section of the questionnaire, the study participants gave insights about their occupation and development experience which is visible in Table 6.1. Most of the participants were master students in a computer science related field, but also a PhD and a practitioner were part of the study. Software development experience ranged from 3 to 15 years and expertise with different tools varied from not available up to excellent. To avoid bias through previous knowledge, all the subjects were assigned randomly to a group. The values in Table 6.1 indicate that both groups include different levels of experience.

The evaluation took place during two weeks. Each participant worked alone using the assigned tools and was not allowed to communicate with others during the task solving phase. All the participants in the group using SQA-Timeline had no previous experience with the tool. They received a short introduction of the basic features in about five minutes. This introduction was of very general nature and did not favour the functions which are needed for the task solving. The subjects of the control group received a list of links to the publicly available tool instances. If they were not familiar with a specific tool they were given a short introduction as well.

			Experience				
Subject	**Group**	**Occupation**	Dev. years	GitHub	Jira	Jenkins	SonarQube
S1	Timeline	MA	4	poor	poor	average	average
S2	Control	MA	5	excellent	excellent	excellent	excellent
S3	Control	MA	6	good	good	average	average
S4	Timeline	MA	13	excellent	average	average	poor
S5	Control	MA	8	good	good	good	good
S6	Timeline	MA	6	good	poor	poor	n/a
S7	Timeline	MA	3	excellent	poor	poor	n/a
S8	Control	PHD	12	good	good	excellent	good
S9	Control	MA	6	good	poor	poor	poor
S10	Control	PR	8	good	good	poor	poor

MA: Master Student - PHD: PhD Student - PR: Practitioner

Table 6.1: Experience of study participants.

Each task had a time limit of five minutes. The users were asked to keep track of the used up time by themselves. After solving a task, the user had to write down the required time as well as a rating of how difficult he thought the task was. If a task could not be solved in five minutes, they had to move on to the next task and write down possible partial results as well as the five minutes of time it took them. After the task solving phase, the group using SQA-Timeline had to answer some additional questions regarding usability and were given a chance to provide written feedback regarding the application. This last section was not restricted by a time limit.

All six tasks had to be solved for the Apache Camel[1] project. The project was chosen because of its rather high activity of multiple commits and issue changes a day, as well as the available build history of around two weeks. To find a project which offered reasonable data in all four tools was important to be able to evaluate SQA-Timeline in its full potential. The Apache Camel project offers acceptable metrics data as well as build data which was sufficient for the evaluation. The project is a representation of a large-scale open source project which has been in development for multiple years.

The goal of the user study was to establish if SQA-Timeline is able to increase the solving speed or result correctness of different tasks concerning information needs of stakeholders involved in a software development project. Additionally, the usability of SQA-Timeline was evaluated. User feedback in written or oral form was gathered as well to get input about improvements and possible future work concerning the approach. The following sections will analyse the results of the study and tries to provide an unbiased assessment and discussion of findings.

[1]http://camel.apache.org

6.2 Statistical Evaluation

The resulting data from the study is evaluated using statistical hypothesis testing. The goal is to find statistically significant differences between the two user groups regarding certain values. The tested values are the following:

- **Consumed Time**
  The time used for each task is available in the data. To perform the statistical calculations, the time values were converted to seconds.

- **Correctness**
  The correctness of a task is provided by a percentage value. Most tasks were divided into sub-tasks which can either be correct or incorrect. The amount of correctly solved sub-tasks results in a percentage value which provides more granular information about the grade of correctness of a task than just a boolean value.

- **Score per Minute**
  The score per minute is calculated by dividing the correctness (%) by the consumed time (s) and then multiplying it by 60 in order to convert seconds to minutes. The conversion to minutes is performed to provide a more readable number. Since speed without correctly solving the tasks is not very desirable, this value will give insights in how much was correctly solved per used time. Because it combines both previously described values, this metric is an important factor in determining the task solving efficiency and performance of a user group.

The statistical hypothesis testing is done using the Mann-Whitney U-test [MW+47]. This non-parametric test for two independent groups does not rely on the two tested populations being normally distributed like the t-test does. It also offers more robustness which means the result is not as strongly affected by outliers as the t-test [FP10]. The two sided Mann-Whitney U-test is used to test the null hypothesis that both populations are equal against the alternative thesis that one of the populations tends to have lower or higher values than the other one. Applied to the evaluation, this means that it is tested if the population of the timeline users performs different from the one using the baseline tools regarding the tested hypothesis. The test is conducted for the three values described above. The resulting p-value is evaluated against the significance level of $\alpha = 0.05$. If the p-value is below 0.05, the null hypothesis which assumes equality is rejected and it is assumed that the two populations perform differently. After analysing the direction of the inequality visible in the two sample distributions, it can be stated that either the timeline users or the baseline users performed significantly better regarding the tested value. The testing is performed on cumulated overall values and on the task-level. The latter will indicate if there are differences between tasks.

The statistical testing and analysis is performed with R[2], which is a software environment for statistical computing. The p-value calculations for the Mann-Whitney U-test as well as graphical plotting is natively supported by R.

[2]http://www.r-project.org

6.3 Results

The following chapter presents the results of the evaluation. First, the cumulated overall performance of participants is evaluated. After that, each task will be described in detail followed by a the individual task-level results. The usability of SQA-Timeline and a discussion of results will form the final part of this chapter.

6.3.1 Overall Performance

In this section the overall performance of the two study groups is evaluated. The goal is to find out if SQA-Timeline improves overall speed and correctness of the task solving process. As mentioned before, the time, the correctness, and the ratio of the two values will be used as indicators for the performance of the participants. For this overall analysis, the values achieved in the six tasks are cumulated for each test subject. Resulting is the total time as well as the total correctness. From these two values the score to time ratio is calculated. The results of each test subject are visible in Table 6.2. Additionally, the table features median, mean, and standard deviation (SD) for each group as well as the p-value calculated from the U-test.

Subject	Group	Total Time (s)	Total Correctness (%)	Score (%) per min
S1	Timeline	1050	83	4.76
S2	Control	1100	92	5.00
S3	Control	1555	50	1.93
S4	Timeline	856	100	7.01
S5	Control	1610	35	1.29
S6	Timeline	1109	71	3.83
S7	Timeline	1107	100	5.42
S8	Control	800	47	3.54
S9	Control	1680	47	1.69
S10	Timeline	895	100	6.70
Median	Timeline	1050.0	100.0	5.420
Median	Control	1555.0	47.2	1.929
Mean	Timeline	1003.4	90.8	5.545
Mean	Control	1349.0	54.2	2.690
SD	Timeline	119.93	13.31	1.3276
SD	Control	381.94	21.78	1.5489
p-value		0.309	0.034	0.032

Table 6.2: Cumulated results for each subject.

When using the previously stated significance level of 0.05, the p-value of the *total time* accepts the null hypothesis which means there is no significant differences in the *total time* of the timeline and baseline tool users. However, the p-values of the *total correctness* and the *score per minute* reject the null hypothesis with the values of 0.034 and 0.032. This indicates with statistical significance that the timeline and baseline user groups performed differently for these two values.

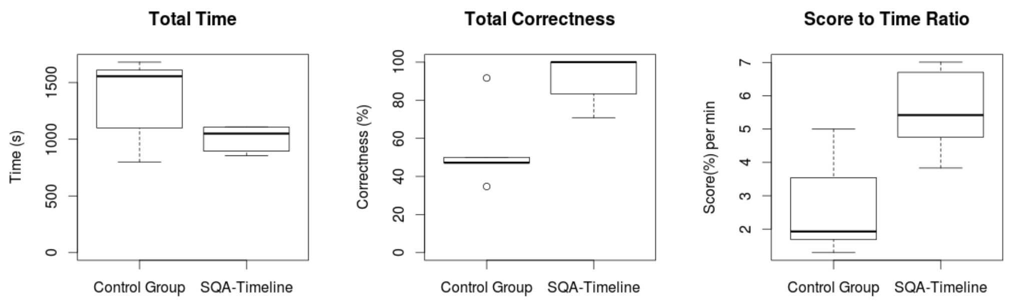


Figure 6.1: Overall results.

The individual distributions for each value of the two samples are visualised in Figure 6.1. Even though the p-value does not indicate a significant difference between the two samples for the *total time*, the box plot shows that the timeline users did have a tendency to use less time to solve all the tasks. With a median of 1555.0 seconds of the control group compared to 1050.0s of the timeline group this tendency becomes visible. The much larger standard deviation of 381.94s for the control group compared to 119.93s for the timeline group shows that the subjects of the control group had higher fluctuations in their *total time*. This may be the result of different previous knowledge about the baseline tools. However, to statistically prove this, a larger data sample with more than five subjects using the baseline tools would be needed. For the time being, it can only be stated that there is no significant difference in the *total time* of both groups. With a larger sample for both groups the tendency may become significant, but this is only speculation.

For the *total correctness* with the p-value of 0.034 it can be stated that the two sample groups are significantly different for the significance level of 0.05. When looking at the median of 100.0% of the timeline group and the median of 47.2% of the control group, the direction of the difference becomes clear. The subjects of the timeline group were able to solve the tasks with significantly higher correctness than the baseline group. In fact, three of five subjects got a 100% score which is also the reason why the median shows that value. When looking at the sample of the control group, no subject was able to score 100% *total correctness*. The maximal achieved value is 92% by subject S2 which stated to have excellent experience with all the baseline tools. The three subjects scoring 100% with the timeline all stated to have less experience with the baseline tools and obviously had no experience with SQA-Timeline. This could also be an indication that SQA-Timeline enables users with less experience to solve tasks with higher correctness. However, in order to analyse this, larger sample sizes would be needed.

As previously described, the *score to time ratio* is calculated using the *total correctness* and the *total time*. The p-value of 0.032 indicates a significant difference between the two samples. When looking at the median of 5.420 %/min for the timeline group compared to 1.929 %/min for the control group, the direction of the difference becomes evident. It can be stated that the timeline group has a significantly higher *score to time ratio*. This means the users of the timeline group where able to solve more tasks correctly per time unit they invested. In other words, they were able to use their time more efficiently towards the goal of solving their tasks correctly. The study has not differentiated wrong answers from blank answers. It is therefore not possible to make a direct statement about a higher error rate in the control group. However, by looking at the individual results on a task level in Table 6.3, it is visible that some tasks have a much lower median than the maximum time of 300s and also feature a low correctness median. Since the users were asked to try to solve the task for 5min (300s) this is an indication that the error rate was high for this task.

6.3.2 Tasks

The subsequent section will provide information about the individual tasks. The task itself and why it has been chosen will be described followed by a discussion of results. The goal is to find out if there are differences between results of individual tasks and to discuss possible reasons for it. Table 6.3 provides an overview over medians and p-values of each task categorised by *time* and *correctness*. By looking at the median *times* it stands out that the control group achieved smaller *times* than the timeline group in two tasks, namely tasks T3 and T6. For all the other tasks the timeline users have been quicker when comparing medians. A detailed task analysis will try to give a possible explanation for these fluctuations. The p-values calculated for the *times* achieved by the two user groups indicate that only task T4 has a significant difference in values with p = 0.028. This matches the findings of the overall analysis which did not result in a significant time difference for the *total time*.

	Task Time (s)			**Task Correctness (%)**		
	Median			Median		
Task	*Timeline*	*Control*	p-value	*Timeline*	*Control*	p-value
T1	55.0	280.0	0.142	100.0	33.3	0.067
T2	220.0	290.0	0.091	100.0	75.0	0.262
T3	270.0	140.0	0.056	100.0	100.00	0.700
T4	173.0	245.0	0.028	100.0	75.0	0.071
T5	73.0	300.0	0.091	100.0	0.00	0.023
T6	255.0	214.0	0.672	100.0	0.00	0.093

Table 6.3: Cumulated results for each task.

The medians of the *correctness* are in favour of the timeline user group. All the medians equal the maximum value of 100% which means that each task was solved correctly by at least three of the five group members. For the control group this is only the case in task T3. There are even two tasks T5 and T6 which were solved completely incorrect by at least three members of the control group, hence the median of 0%. The analysis of the individual tasks might reveal why there is a difference in the achieved *correctness* between the two user groups. Even though the medians indicate a difference, only one p-value of task T5 shows statistical significance.

As mentioned, there is only one case of statistical significance for *time* as well as one for the *correctness*. The *score to time ratio*, which is not shown in Table 6.3, also contains one single case of significance. These non-significant p-values are probably the result of small the sample sizes, which are not big enough to show significance on a task level as it did for the cumulated results. Therefore mostly the tendencies indicated by the medians will be used to discuss the individual tasks.

Task 1

Project activity in the last two weeks:

- *Number of commits?*
- *Number of failed builds?*
- *Number of created issues?*

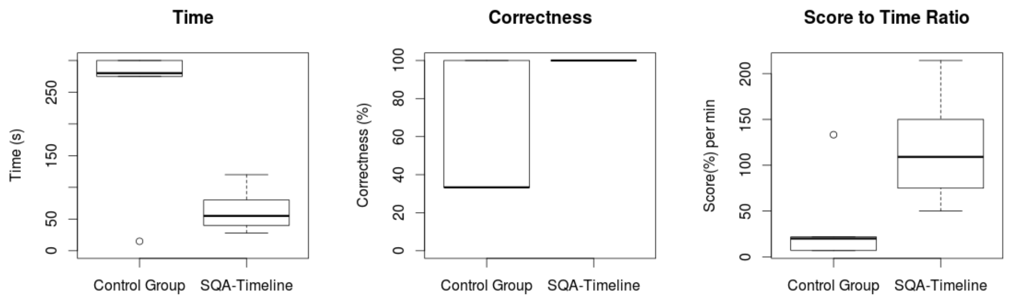


Figure 6.2: Task 1 results.

The first task wants the user to get a quick overview over the events of the last two weeks. The three sought-after events are an example for event types which could be looked for by different stakeholders. The task represents various scenarios of stakeholders wanting to get a summary of events which happened in a certain time frame. To keep the task simple, the participants only had to write down the number of events without any additional details. The task was included to show if it is possible to quickly perceive an overview over different events without having more specific information needs. The topic of project awareness has been described by Treude and Storey [TS10]. They point out the importance of dashboards and the information they represent about the project. Task 1 was designed based on their findings.

A possible real world scenario for task 1 is a project manager wanting to gather information about the project activity. If for example the number of failed builds is exceptionally high, this can indicate that faulty code was submitted and the development process was being disrupted. If it were too unpractical to get the information in the baseline tools, it might result in the stakeholder not even going through the trouble of looking it up. This could result in a deficit of project awareness.

When looking at box plots in Figure 6.2, a tendency towards lower *time* and higher *correctness* for the timeline group is visible. In fact, all the subjects of the timeline group solved this task 100% correct. The control group on the other hand has a much lower degree of correctness with a median of 33.3%. One participant in the control group only partially solved the task, but wrote down a very low time. This was probably a misunderstanding of the task, nevertheless the data was left as is in the study.

To solve the tasks, the timeline users could use the overview screen visualisation and select the right time frame. The high *correctness* indicates that this features has been understood and was used correctly by the timeline users. The control group on the other hand had to look up the information in three different baseline tools which was more time consuming and error-prone. This indicates that SQA-Timeline enables stakeholders to quickly gain awareness which event types happened in a certain time frame.

Task 2

Find these two builds and write down their result:

- *Build 1752*
- *Build 1753*

What happened between the two builds?

- *Created Issues (ID)?*
- *Resolved Issues (ID)?*
- *Commits (Author + Time)?*

Frequent builds are a core concept of proper continuous integration [FF06]. A build failure may indicate faulty code and should be investigated. Task 2 picks up on this topic. It contains two steps. First, the participants had to find two specific builds and write down their result. In the second part they had to identify events which occurred between these two builds. The sought-after event types and desired attribute(s) were specified in the task description. The task represents a scenario where a stakeholder wants to find out the reason for a build failure. Any code change since the last build success can be the source of the error. Issue events may indicate what items have been worked on during this time, which could help to isolate the error. The specific information needs which this task is based upon have been described by Fritz and Murphy [FM10]. They mention various questions developers ask regarding broken builds, including who and what caused the failure.

The box plots in Figure 6.3 displays the results for this task. When looking at the *correctness* achieved by the timeline users, it is visible that most of the participants solved the task correctly. In fact, only one was not able to solve the task with 100% *correctness*. It is the same subject which needed the full 300 seconds. The other participants seemed to be able to solve the task quicker and more correct than the control group. This tendency is also visible when looking at the *score to time ratio* chart.

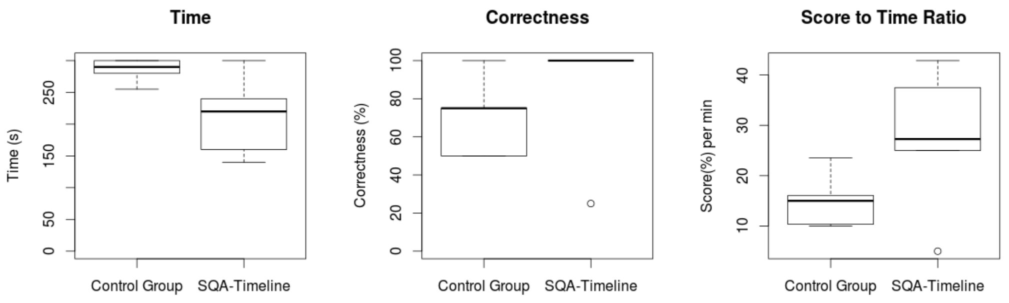


Figure 6.3: Task 2 results.

The first part of the task, where the builds had to be identified, was solved correctly by all the participants of both groups. Differences were visible in the second part, where the control group users missed several events or even wrote down false positives. This must be the case because the dates of events need to be checked manually in the baseline tools. They usually offer a list of events sorted by date, but when comparing multiple dates and times across different tools, errors can occur more frequently. For example if the date portion of an event is a match, but the time is outside the time frame, the user might overlook the time and add a false positive. This kind of error is prevented in the timeline because it is clearly visible what happened before or after a build.

Task 3

Find all commits of developer 'XY' between 01.01.2014 and 01.02.2014:

- *Number of commits?*
- *ID's of Issues he committed to in this time frame?*

Finding out what specific developers have been doing is a common task for stakeholders involved in software development. There are various possible scenarios for this task. For example a supervisor wanting to check what somebody was working on recently. Questions like "What have my co-workers been doing?" [KDV07] or "What have people been working on?" [FM10] are often mentioned in literature and are the reason why this task was included in the study.

In order to solve the task, the participants needed to identify the commits of a specific developer during a given month. The first part of the task was simply counting the amount of commits in the time frame. In the second part, they needed to write down all the issues the developer has worked on during this time. Since most commits in the Apache Camel project are associated with an issue, this will give a basic understanding what the developer has been working on in this time.

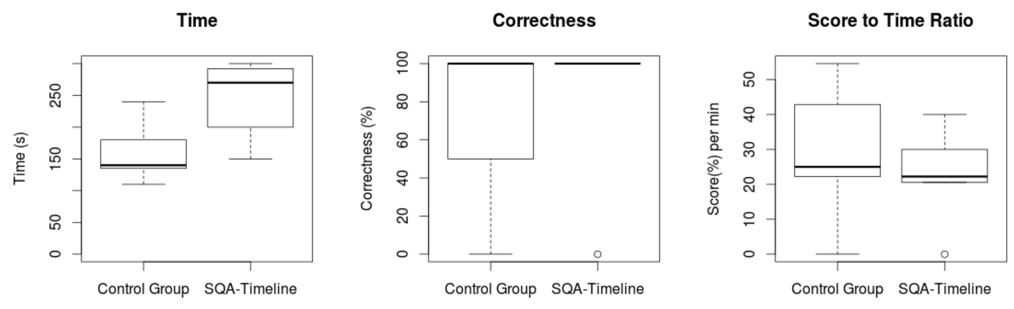


Figure 6.4: Task 3 results.

When looking at the result in Figure 6.4 it stands out that the control group tended to have less *time* to solve the task than the timeline group. The reason for this is probably that the task only involved one view of source code repository. The users just had to go through the commit list of the developer which is available in GitHub. The timeline users on the other hand had to use filter or highlight features in order to solve the task efficiently. These features might be a bit hard to find for a first time user. They have been mentioned in the introduction, but most users firstly needed some time to figure out how to solve that task. The *correctness* is quite equal for both groups even though the baseline has one more only partially correct answer than the timeline. When looking at the *score to time ratio* we can conclude that both groups were able to solve the tasks with quite equal performance with medians of 25.0 %/min for the baseline and 21.2 %/min for the timeline users.

Task 4

What happened the 20.03.14 (24h)?
Write down the information in brackets:

- *Created Issues (ID)?*
- *Resolved Issues (ID)?*
- *Commits (Author)?*
- *Builds (Result)?*

Task 4 is similar to task 1 and focusses on various events which happened during a time frame. However, the task asks for more specific information than task 1, which only asked for the number of events. The task stands for scenarios where different events need to be analysed in context of time. Like in task 1, this can be to foster project awareness [TS10] but on a more detailed level. However, awareness is only a fraction of scenarios included in this task. Information retrieval for error solving or review processes are also possible candidates. An example is if a team is close to reach a deadline for a release, a manager might want to closely monitor what is happening. Instead of just a number of how many issues have been created and reopened, he wants detailed information about the individual events in order be able to react to possible incidents. To solve the task, the participants only had to write down the attribute value in brackets. It is assumed if they were able to find this information, they would also have found additional attributes regarding this item.

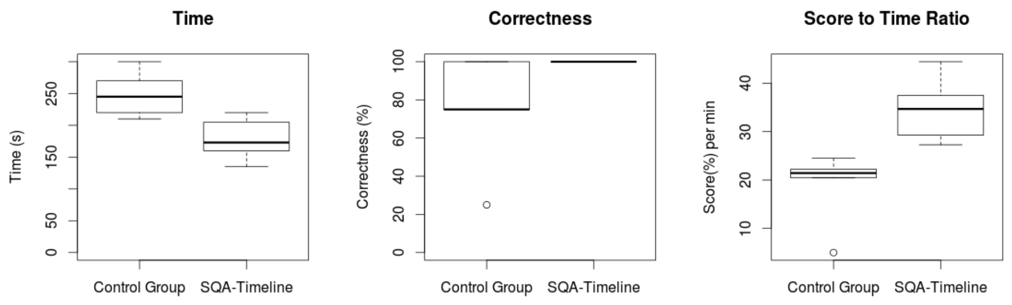


Figure 6.5: Task 4 results.

The study results for this task show that the timeline users were able to solve this task significantly faster than the control group with a p-value of 0.028. The timeline group features a 100% *correctness* for all members. The control group tends to have a lower *correctness* with a median of 75%. The *score to time ratio* indicates a significantly better performance of the timeline group with a p-value of 0.008.

The results of this task are clearly in favour of the timeline. They indicate the timeline enables the users to extract information in the context of time more efficiently than the baseline tools. The worse performance of the baseline tools is probably due to the fact that the data had to be extracted from multiple sources, all of which feature different functionality and representation of data. The timeline on the other hand has a unified design which supports less experienced users.

Task 5

Find the last 2 (newest) measurements of the 'Unit Test Failures' metric.
Write down the following attributes:

- *Date*
- *Value*

To solve task 5, the participants needed to identify the two latest measurements of a certain metric and write down their dates and values. The reason why the task asks for the second to last value as well is because the development of the metric might be of importance. Monitoring of different metrics is a common task performed in software quality assurance [Gal04]. Measurements are influenced by events like code changes or test failures. To be able to monitor these metrics and understand what is causing them is important in modern software development [BZ12].

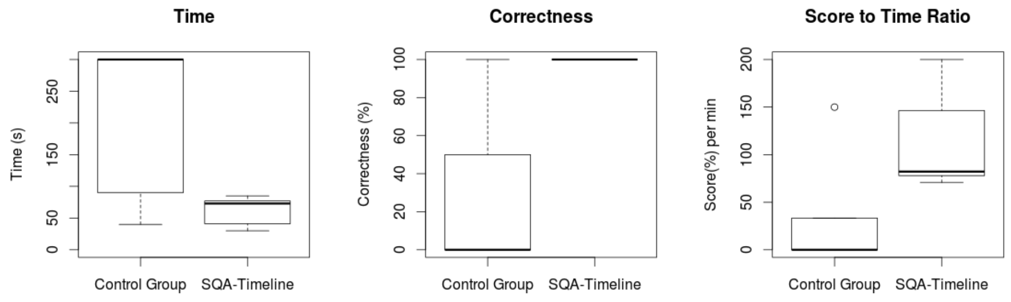


Figure 6.6: Task 5 results.

The results show that task 5 was solved correctly by all members of the timeline group. In fact, the p-value of 0.023 shows that is has been solved with significantly higher *correctness*. When looking at the *time* plot in Figure 6.6, a tendency towards a lower time of the timeline group is visible. The reasons for these results are probably the user interface of SonarQube which is the baseline tool containing the metrics. In order to look up the last two measurements, certain steps are required which need some expertise. The way they are displayed in the timeline seems to be a more convenient way to extract the data.

The timeline also enables the user to identify events which may have caused a change in a metric. This is an additional advantage which has not been part of this task because this would have made it too difficult for the baseline users. The metrics data in SonarQube is out of context and it would require quite some effort and expertise to find out what has been the cause of a value change. If for example the 'Unit Test Failures' metric has increased, the user would need to look for the related build information and the commits in two other tools. The timeline however offers this information right away in the context of time.

Task 6

How many commits between 16.03.14 00:00 and 26.03.14 24:00 have no issue assigned and are not a merge?

- *Number of commits*

In collaborative software projects, it is important to know the contents of individual commits. In order to describe source code changes contained in a commit, the developers write a commit message. It is good practice to associate a commit with an issue by putting the issue key in the message. This is the recommended way for projects of the Apache Software Foundation[3]. Many projects like Apache Maven[4] include a convention where a commit must be linked to at least one

[3]http://www.apache.org/dev/committers.html#applying-patches
[4]https://maven.apache.org/developers/conventions/svn.html

issue. If no relevant issue is existing, the committer is strongly encouraged to create one for significant changes. With a convention like this in mind, a manager might be interested in monitoring if it is followed by the developers. Developers themselves are also often interested if they followed the conventions of the team [KDV07].

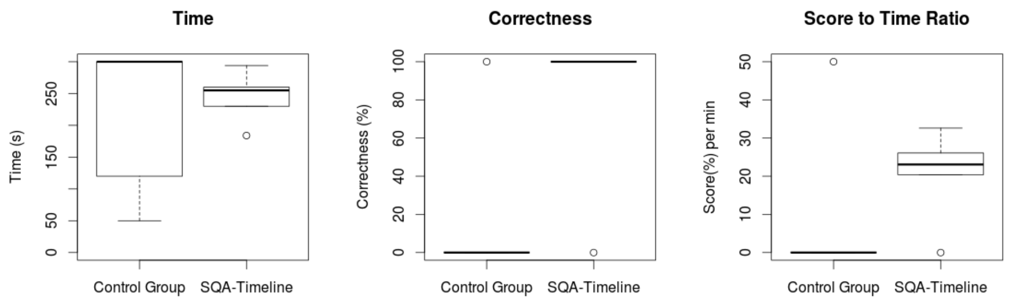


Figure 6.7: Task 6 results.

Task 6 required the participants to find out how many commits had no issue assigned. Merges were excluded, because they usually do not contain actual contributions. The time frame was heavily restricted to 10 days, because each item needed to be analysed manually by the baseline group. A larger time frame would have made the task too time-consuming. The timeline users had a preconfigured pattern available. However, they were not told that this specific pattern exists and the decision to use a pattern and finding the right one was up to them. When looking at the results in Figure 6.7 it is visible that the timeline users solved the task with higher *correctness*. In fact, all except one timeline users were able to give the right answer. On the other hand, only one member of the control group solved the task correctly. The required *time* was quite high for both groups. This is probably due to the fact that the timeline users had to find the appropriate pattern first. The control group had to manually look at every commit message which was quite time consuming as well. The manual processing was probably also the reason why many of them gave the wrong answer.

The results of task 6 leads to the conclusion that the pattern system supports the users of the timeline in identifying specific events, in this case policy violations. This does not necessarily happen quicker, but with a higher rate of correctness. In order to solve the particular task, the users still had to count the pattern instances manually. Some users suggested after the study, an automatic counter how many patterns are found in a time frame, in contrast to the available one for overall found patterns, would be useful.

6.3.3 Usability and User Feedback

The participants of the timeline group were asked to complete a usability questionnaire according to the System Usability Scale (SUS) by Brooke [Bro96]. The SUS is a set of 10 standard questions which have to be answered on a scale from strongly agree to strongly disagree. The answers can be converted to a score from 0-100. The five timeline users scored a median of 90 and a mean of 89 points. The lowest score was 82.5 and the highest one 95. These scores indicate an "excellent" level of usability [BKM09]. The high SUS score suggests that SQA-Timeline was generally intuitive to use and had no major flaws which the users were not able to overcome.

Next to the SUS questionnaire, the timeline users were also asked three open questions about positive and negative impressions as well as suggestions for improvements of SQA-Timeline. In the section about positive feedback, many users commended the intuitive and easy usage of the timeline. The multi layer presentation of the data was picked up on quickly by the users and left a positive impression. Multiple users were impressed by the fast response of the layered visualisation mechanism. The search feature received positive feedback in terms of speed and usability. In general, the users appreciated the presentation of the data in the timeline and the interaction design.

There were also some suggestions for improvements. One user missed a feature to enter a date range for the timeline. Instead of dragging the selectors to the desired range he wanted an alternative way to specify a date range for the timeline. This is a valid point which would certainly improve the navigation experience. Other suggestions where to trigger pop-ups on mouseover rather than a click. Some users also made comments about the pattern functionalities. As mentioned before, a counter feature for pattern instances occurring in a certain time frame was noted by multiple users. This is more of an advanced feature, but was mentioned because it would have been helpful to solve task 6. One user also suggested a help function giving more information about how to create patterns. This is currently not included in the application, but if the prototype were to be deployed in an actual production environment, some form of tutorial will be needed for the pattern functionalities.

6.3.4 Study Limitations

The user study was conducted in order to objectively determine the value of SQA-Timeline compared to currently available tools. However, there are certain factors which may limit the validity of the results:

- **Small samples**
  The study was conducted with 10 participants, five in each test group. These are very small sample sizes to prove statistical significance. But for most results it was enough to show a clear tendency.
- **Experience of participants**
  The general development experience and the previous knowledge about the baseline tools varied from user to user. They were randomly assigned to a group in order to prevent bias of results. However, combined with the small sample size these variations may had an impact on the results of the study.
- **Validity of tasks**
  The tasks were designed to cover a range of real world scenarios. Most tasks were based on information needs described in literature. But in order to create clear and assessable tasks, certain assumptions and simplifications had to be made.

6.3.5 Discussion of Results

The overall results of the study showed that the SQA-Timeline users were not necessarily able to solve tasks faster, but with a higher correctness than the baseline users. The results also showed a significantly higher score per time ratio. The total time used for the tasks showed only a tendency in favour of the timeline users, but no statistically significant difference. This is probably due to the fact that all the timeline users were using the tool for the first time when participating in the study. Even after a short introduction, it took some time to become familiar with the features and handling of the tool. The baseline users could profit from previous experience with the tools, which is probably also why their total times have a larger spread than the ones of the timeline users, which lie quite close together. With more experience, the SQA-Timeline users would probably be able to solve tasks faster than shown in the study.

The users of the timeline proved that they could solve the particular tasks with higher correctness than the baseline. This indicates that the timeline is a valuable addition to the tools available to stakeholders involved in software engineering projects. It will help them to solve certain tasks, especially when there are multiple data sources involved. The SQA-Timeline was never intended as a replacement for the existing tools. The baseline tools will still be needed for their primary use. However, SQA-Timeline helps to see links between events in context of time. This is beneficial for a variety of previously described tasks.

In addition to solving specific tasks such as finding a potential error source, SQA-Timeline also increases project awareness. By using the overview screen or the timeline, the users are able to quickly perceive the event history. They can browse though recent events in a matter of seconds without having to gather the data from different tools and link it together. This enables stakeholders to take a look at the recent data and gain awareness without having a specific reason to do so. Like this it might be possible that a stakeholder finds irregularities which he would not have noticed without SQA-Timeline. With the support of certain patterns, he is also able to automate the detection of events of interest.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

The goal of the thesis was to answer the following research question: *How can a timeline-based visualisation approach foster the understanding of software evolution in large-scale software systems?* In order to answer this question, a user study was conducted with a previously implemented research prototype called SQA-Timeline. During the evaluation, the performance of SQA-Timline was compared to a baseline consisting of four commonly used tools. Both user groups solved a set of tasks originating from information needs of certain stakeholders. The tasks involved different scenarios like the enhancement of project awareness, the identification of possible error sources as well as monitoring of policy compliance and metric development. The results show that the users of the implemented timeline approach were able to solve certain tasks with higher correctness and efficiency than the users of the baseline tools. All the timeline users had no previous experience and were still able to use the tool efficiently. The usability evaluation and given feedback was very positive which coincides with the increased task solving performance.

The timeline approach, extended with additional features as implemented in SQA-Timeline, can be a valuable addition to the currently available tools. However, this is very dependent on the project setting and the processes it involves. SQA-Timeline works best with projects implementing continuous integration concepts. Multiple builds a day, as well as frequent calculation of metrics provide the means to narrow down possible error sources. In addition, proper usage of commit messages and issue tracker are crucial to create awareness of project activities. If these conditions are met, the approach of SQA-Timeline shows promising results. A next step would be an evaluation of the tool over a longer duration in a real project setting. Involved stakeholders will be able to give much more of an insight about the actual value and shortcomings of SQA-Timeline.

7.2 Future Work

The current implementation of SQA-Timeline is only a prototype and still has much more room for improvements. Based on experience gathered during development and the results of the evaluation, a few suggestions are presented below:

- **Real time data**
  The data displayed in SQA-Timeline is cached on the back end. Because it has to be preprocessed, it is fetched in regular time periods. For a developer it is important to have

very up to date data available. The display of real time data would require changes in the back end. The front end is currently only refreshed on project reload. To improve the user experience, dynamic updates of the displayed data would be valuable. The concept of dynamic updating of content is included in many modern web applications. It enables the user to keep the page open and new events would be displayed without any interaction. This would open new appliances for the timeline approach, for example on a big screen in the development offices. The recent history would always be visible and therefore amplify the project awareness for the entire team in the office. In order to display important data without user interaction, changes in the way the data is displayed would be necessary. The timeline would need to display more data, since it can not rely on pop-ups. The controls section could either be omitted or contain a live feed of events with further details.

- **User Accounts**
  User accounts would enable user specific views and the possibility to save the configuration on the server. The foundation of SQA-Timeline is ready for the integration of user accounts. A login feature is already available and all the display states and options are currently initialised using a single JSON file. This configuration could in future be saved on the server associated to a user account. This would reinstate configured options on login.

- **Improved Audio Playback**
  The SQA-Timeline currently includes the experimental feature to play audio for the events history. This feature has not been evaluated and is still very basic. It leaves room for further research in this field. The resulting audio track of a development week could for example be used for a weekly review by a project manager. A stakeholder could listen to the track in the car on his way to work and already get an idea of what to expect when looking at the actual data. Ideas in this field need further development and might be worth pursuing.

Appendix A

Evaluation Questionnaire

Evaluation of SQA-Timeline

The goal of this experiment is to evaluate *SQA-Timeline*.

There are two user groups. You are part of one group and are asked **not** to use the tool(s) of the other group:

☐ User Group 1: **SQA-Timeline**

☐ User Group 2: **GitHub, JIRA, Jenkins, SonarQube**

All the task are concerning the *Apache Camel* project. The relevant tools can be found at the following locations:

- **SQA-Timeline:** https://seal-students.ifi.uzh.ch/sqa-pattern-timeline
- **GitHub:** https://github.com/apache/camel
- **JIRA:** https://issues.apache.org/jira/browse/CAMEL
- **Jenkins:** https://builds.apache.org/job/Camel.trunk.fulltest/
- **SonarQube:** https://analysis.apache.org/dashboard/index/37401

Both user groups are asked:

- To solve the tasks as fast as possible. For each task the available time is 5 minutes. Do not use more than the amount of time assigned to each task. In case that you cannot finish a task within the allotted time, you have to proceed to the next one.
- To write down how much time you have spent on each of the tasks (including the time needed for writing down the answer) in the according fields (even if you could not complete a task). If you have any comments regarding a task, use the comment section.

Thank you for participating in our evaluation.

Personal Details

We kindly ask you to share below some personal information about you. It is used exclusively for statistical purposes.

1. **What is your current occupation?**
   - ☐ Bachelor's student
   - ☐ Master's student
   - ☐ PhD student
   - ☐ Practitioner
   - ☐ Other: ____________________
2. **How many years of software development expertise do you have?** _____ years.
3. **Please tell us about your skills in the following areas:**

		excellent	good	average	poor	n/a
a.	**My familiarity with *GitHub* is:**	☐	☐	☐	☐	☐
b.	**My familiarity with *JIRA* is:**	☐	☐	☐	☐	☐
c.	**My familiarity with *Jenkins* is:**	☐	☐	☐	☐	☐
d.	**My familiarity with *SonarQube* is:**	☐	☐	☐	☐	☐

Please respect the time limit of 5 minutes given for each task.

4. **Task 1: Project activity in the *last 2 weeks*:**
   - Number of commits: ____________
   - Number of failed builds: ____________
   - Number of created issues: ____________

Overall, this task was?	Very easy □—□—□—□ Very difficult
Time spent on task (mm:ss):	________________________
Comments:	

5. **Task 2: Find these two builds and write down their result (failed/successful/aborted...):**
   - Build 1752 Result:__________________
   - Build 1753 Result:__________________

   List what happened between the two builds:

Created Issues (ID)	Resolved Issues (ID)	Commits (Author + Time)

Overall, this task was?	Very easy □—□—□—□ Very difficult
Time spent on task (mm:ss):	________________________
Comments:	

6. **Task 3: Find all commits of developer 'Babak Vahdat' between 01.01.2014 and 01.02.2014 (only Apache Camel project):**

   Number of commits: __________________

   ID's of Issues he committed to in this time frame:
   (Hint: ID's are in commit message with format 'CAMEL-xxxx')

   ____

Overall, this task was?	Very easy □—□—□—□ Very difficult
Time spent on task (mm:ss):	________________________
Comments:	

Please respect the time limit of 5 minutes given for each task.

7. **Task 4: What happened the 20.03.14 (24h)?**
   Write down the information in brackets:

Created Issues (ID)	Resolved Issues (ID)	Commits (Author)	Builds (Result)
________	________	________	________
________	________	________	________
________	________	________	________
________	________	________	________
________	________	________	________

Overall, this task was?	Very easy □—□—□—□ Very difficult
Time spent on task (mm:ss):	____________
Comments:	

8. **Task 5: Look at the 'Unit Test Failures' metric.**

   Find the last 2 (newest) measurements:
   - Date: ________________ Value: ________
   - Date: ________________ Value: ________

Overall, this task was?	Very easy □—□—□—□ Very difficult
Time spent on task (mm:ss):	____________
Comments:	

9. **Task 6: How many commits between 16.03.14 00:00 and 26.03.14 24:00 have no issue assigned and are not a merge?**

   - Number: ____________

Overall, this task was?	Very easy □—□—□—□ Very difficult
Time spent on task (mm:ss):	____________
Comments:	

Only anwer these questions if you used the SQA-Timeline tool.

Debriefing

Usability

Usability questions according to the System Usability Scale (SUS) by Brooke 1986.

10. Usability Feedback

a.	**I think that I would like to use this system frequently**	Strongly disagree	□—□—□—□—□	Strongly agree
b.	**I found the system unnecessarily complex**	Strongly disagree	□—□—□—□—□	Strongly agree
c.	**I thought the system was easy to use**	Strongly disagree	□—□—□—□—□	Strongly agree
d.	**I think that I would need the support of a technical person to be able to use this system**	Strongly disagree	□—□—□—□—□	Strongly agree
e.	**I found the various functions in this system were well integrated**	Strongly disagree	□—□—□—□—□	Strongly agree
f.	**I thought there was too much inconsistency in this system**	Strongly disagree	□—□—□—□—□	Strongly agree
g.	**I would imagine that most people would learn to use this system very quickly**	Strongly disagree	□—□—□—□—□	Strongly agree
h.	**I found the system very cumbersome to use**	Strongly disagree	□—□—□—□—□	Strongly agree
i.	**I felt very confident using the system**	Strongly disagree	□—□—□—□—□	Strongly agree
j.	**I needed to learn a lot of things before I could get going with this system**	Strongly disagree	□—□—□—□—□	Strongly agree

Your Impression

Thank you for participating in our evaluation. To conclude our experiment, we would like to ask you for some feedback on our tool.

11. What did you like most about *SQA-Timeline*?

____

____

12. What did you like least about *SQA-Timeline*?

____

____

13. Are there any suggestions for improvement or other comments?

____

____

____

Appendix B

Evaluation Results

		Participant					
Subject #	**Group**	**Occupation**	**Dev. Years**	**GitHub**	**JIRA**	**Jenkins**	**SonarQube**
2	Control	MA	5	excellent	excellent	excellent	excellent
3	Control	MA	6	good	good	average	average
5	Control	MA	8	good	good	good	good
8	Control	PHD	12	good	good	excellent	good
9	Control	MA	6	good	poor	poor	poor
1	Timeline	MA	4	poor	poor	average	average
4	Timeline	MA	13	excellent	average	average	poor
6	Timeline	MA	6	good	poor	poor	n/a
7	Timeline	MA	3	excellent	poor	poor	n/a
10	Timeline	PR	8	good	good	poor	poor
p-value							

		Task 1			Task 2			Task 3		
Subject #	**Group**	**Time**	**Correctness**	**Difficulty**	**Time**	**Correctness**	**Difficulty**	**Time**	**Correctness**	**Difficulty**
2	Control	5:00	3/3	1	4:40	3/4	2	2:20	2/2	2
3	Control	4:35	3/3	4	4:15	4/4	4	3:00	0/2	3
5	Control	5:00	1/3	3	5:00	2/4	2	1:50	2/2	1
8	Control	0:15	1/3	1	5:00	3/4	2	2:15	1/2	1
9	Control	4:40	1/3	3	4:50	2/4	3	4:00	2/2	2
1	Timeline	2:00	3/3	1	4:00	4/4	2	4:30	2/2	2
4	Timeline	0:40	3/3	1	2:40	4/4	1	2:30	2/2	2
6	Timeline	0:28	3/3	1	5:00	1/4	4	5:00	0/0	3
7	Timeline	00:55	3/3	1	3:40	4/4	3	4:52	2/2	3
10	Timeline	1:20	3/3	1	2:20	4/4	1	3:20	2/2	2
p-value		0.143	0.067		0.091	0.262		0.056	0.699	

		Task 4			Task 5			Task 6		
Subject #	**Group**	**Time**	**Correctness**	**Difficulty**	**Time**	**Correctness**	**Difficulty**	**Time**	**Correctness**	**Difficulty**
2	Control	3:40	3/4	1	0:40	2/2	1	2:00	1/1	2
3	Control	4:05	4/4	3	5:00	0/2	4	5:00	0/1	4
5	Control	5:00	1/4	3	5:00	0/2	4	5:00	0/1	4
8	Control	3:30	3/4	2	1:30	1/2	2	0:50	0/1	1
9	Control	4:30	4/4	3	5:00	0/2	4	5:00	0/1	4
1	Timeline	2:15	4/4	1	0:30	2/2	1	4:15	0/1	3
4	Timeline	2:53	4/4	1	1:13	2/2	2	4:20	1/1	2
6	Timeline	3:40	4/4	1	1:17	2/2	1	3:04	1/1	3
7	Timeline	3:25	4/4	2	0:41	2/2	1	4:54	1/1	3
10	Timeline	2:40	4/4	1	1:25	2/2	1	3:50	1/1	2
p-value		0.028	0.071		0.091	0.023		0.672	0.093	

		Usability											Cumulted Values		
Subject #	**Group**	**a**	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**	**j**	**SUS**	**Time (s)**	**Correctness**	**%/min**
2	Control												1100	0.917	5.000
3	Control												1555	0.500	1.929
5	Control												1610	0.347	1.294
8	Control												800	0.472	3.542
9	Control												1680	0.472	1.687
1	Timeline	4	1	5	2	5	2	5	1	5	2	90	1050	0.833	4.762
4	Timeline	4	1	5	1	5	1	4	1	5	1	95	856	1.000	7.009
6	Timeline	5	2	3	1	5	1	5	1	3	3	82.5	1109	0.708	3.832
7	Timeline	4	2	4	1	4	1	5	1	5	3	85	1107	1.000	5.420
10	Timeline	5	1	4	1	5	1	5	1	4	2	92.5	895	1.000	6.704
p-value													0.310	0.034	0.032

Appendix C

Setup of SQA-Timeline

The application files of SQA-Timeline are structured into various folders. The following list describes each folder and its contents:

- **/audio**
  Contains the the audio files which are played for each event when using the audio playback feature of the timeline. The audio format needs to be mp3.

- **/build**
  Contains the built version of the application. After running the build script, the version which should be deployed on the web server is found in this folder.

- **/css**
  Contains the LESS files which are compiled to CSS during the build process. If the development version is run, they are compiled at runtime.

- **/html**
  Contains the HTML files for each page of the application. Each page runs independently, however navigation to other pages will only work when running `app.html`.

- **/img**
  Contains the image files used in the application.

- **/js**
  Contains the JavaScript files for SQA-Timeline. The files associated to a specific screen are named `[screenName].[filename].js`.

- **/lib**
  Contains a folder for each external library used in the application. All the JS and CSS files for a library are located in one of these folders. A small exception to this is the file `variables.less` which is located in the `css` folder. The file contains customised variables for bootstrap combined with other variables used in the application.

C.1 Building the Application

The folder `build` contains a pre built version of SQA-Timeline which is ready for deployment on a web server. However if changes to any of the source folders are made, the build script has to be run. The build process will completely wipe the `build` folder and rebuild the application. Write access is required for this process.

To run the build script, an installation of Node.js[1] is required. In addition, Grunt[2] needs to be installed using the following global command:

```
npm install -g grunt-cli
```

After that, all the dependencies specified in `package.json` need to be installed. This is achieved using the following command on the root folder containing the file:

```
npm install
```

Now the build process which is specified in the file named `Gruntfile.js` can be started with this command on the same folder:

```
grunt
```

C.2 Deployment

To deploy the application, all the folders in the `build` directory need to be served by a web server. The folder `js` contains a file called `config.js`. Several global configuration options such as the web service locations can be adjusted in this file. To start SQA-Timeline, `app.html` needs to be opened in a browser. An alternative is to open `index.html` in the root directory.

It is strongly encouraged to deploy the built version on the server rather than the development version. The development version is much slower because the CSS will be compiled at runtime. In addition, the minification and combination of source files during the build will minimise traffic and load times.

[1]http://nodejs.org
[2]http://gruntjs.com

Appendix D

CD Contents

- **master_thesis.pdf**
  The final version of the thesis in PDF.
- **abstract.txt**
  The abstract in english language.
- **zusfsg.txt**
  The abstract in german language.
- **evaluation/**
  - **evaluation-results.odt**
    An open office table containing the evaluation results.
  - **evaluation-questionnaire.pdf**
    The evaluation questionnaire.
- **sqa-timeline/**
  The folder containing the complete source code of SQA-Timeline.
- **latex/**
  The folder containing latex source files of the thesis.

Bibliography

[ANM$^+$07] R. Agarwal, P. Nayak, M. Malarvizhi, P. Suresh, and N. Modi. Virtual quality assurance facilitation model. In *Global Software Engineering, 2007. ICGSE 2007. Second IEEE International Conference on*, pages 51–59, 2007.

[BG09] S. Boccuzzo and H.C. Gall. Cocoviz with ambient audio software exploration. In *Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on*, pages 571–574, May 2009.

[BGG14] Martin Brandtner, Emanuel Giger, and Harald Gall. Supporting continuous integration by mashing-up software quality information. In *IEEE CSMR-WCRE 2014 Software Evolution Week (CSMR-WCRE)*, pages 109–118, Antwerp, Belgium, FEB 2014. IEEE.

[Bin07] David Binkley. Source code analysis: A road map. In *2007 Future of Software Engineering*, FOSE '07, pages 104–119, Washington, DC, USA, 2007. IEEE Computer Society.

[BKM09] Aaron Bangor, Philip Kortum, and James Miller. Determining what individual sus scores mean: Adding an adjective rating scale. *Journal of usability studies*, 4(3):114–123, 2009.

[BOH11] M. Bostock, V. Ogievetsky, and J. Heer. D3 data-driven documents. *Visualization and Computer Graphics, IEEE Transactions on*, 17(12):2301–2309, Dec 2011.

[BP84] F.J. Buckley and Robert Poston. Software quality assurance. *Software Engineering, IEEE Transactions on*, SE-10(1):36–41, 1984.

[Bro96] John Brooke. Sus-a quick and dirty usability scale. *Usability evaluation in industry*, 189:194, 1996.

[BVGW10] Dane Bertram, Amy Voida, Saul Greenberg, and Robert Walker. Communication, collaboration, and bugs: The social nature of issue tracking in small, collocated teams. In *Proceedings of the 2010 ACM Conference on Computer Supported Cooperative Work*, CSCW '10, pages 291–300, New York, NY, USA, 2010. ACM.

[BZ12] Raymond P. L. Buse and Thomas Zimmermann. Information needs for software development analytics. In *Proceedings of the 2012 International Conference on Software Engineering*, ICSE 2012, pages 987–996, Piscataway, NJ, USA, 2012. IEEE Press.

[DLR10] M. D'Ambros, M. Lanza, and R. Robbes. An extensive comparison of bug prediction approaches. In *Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on*, pages 31–41, 2010.

[DMG07] Paul M Duvall, Steve Matyas, and Andrew Glover. *Continuous integration: improving software quality and reducing risk*. Pearson Education, 2007.

[FCJV13] Carla Fernandez-Corrales, Marcelo Jenkins, and Jorge Villegas. Application of statistical process control to software defect metrics: An industry experience report. In *Empirical Software Engineering and Measurement, 2013 ACM / IEEE International Symposium on*, pages 323–331, 2013.

[FF06] Martin Fowler and Matthew Foemmel. Continuous integration. *Thought-Works http://www.thoughtworks.com/Continuous Integration.pdf*, 2006.

[FM10] Thomas Fritz and Gail C. Murphy. Using information fragments to answer the questions developers ask. In *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ICSE '10, pages 175–184, New York, NY, USA, 2010. ACM.

[FP98] Norman E. Fenton and Shari Lawrence Pfleeger. *Software Metrics: A Rigorous and Practical Approach*. PWS Publishing Co., Boston, MA, USA, 2nd edition, 1998.

[FP10] Michael P Fay and Michael A Proschan. Wilcoxon-mann-whitney or t-test? on assumptions for hypothesis tests and multiple interpretations of decision rules. *Statistics surveys*, 4:1, 2010.

[Gal04] Daniel Galin. *Software quality assurance: from theory to implementation*. Pearson education, 2004.

[Hal12] Wesley Hales. *HTML5 and JavaScript Web Apps*. O'Reilly Media, Incorporated, 2012.

[HBP09] E. Hossain, M.A. Babar, and Hye-young Paik. Using scrum in global software development: A systematic literature review. In *Global Software Engineering, 2009. ICGSE 2009. Fourth IEEE International Conference on*, pages 175–184, July 2009.

[Hil14] Stefan Hiltebrand. Sqa-pattern. Master's thesis, University of Zurich, April 2014.

[HLSN13] Nicole Haenni, Mircea Lungu, Niko Schwarz, and Oscar Nierstrasz. Categorizing developer information needs in software ecosystems. In *Proceedings of the 2013 International Workshop on Ecosystem Architectures*, WEA 2013, pages 1–5, New York, NY, USA, 2013. ACM.

[JS91] B. Johnson and B. Shneiderman. Tree-maps: a space-filling approach to the visualization of hierarchical information structures. In *Visualization, 1991. Visualization '91, Proceedings., IEEE Conference on*, pages 284–291, Oct 1991.

[KDV07] Andrew J. Ko, Robert DeLine, and Gina Venolia. Information needs in collocated software development teams. In *Proceedings of the 29th International Conference on Software Engineering*, ICSE '07, pages 344–353, Washington, DC, USA, 2007. IEEE Computer Society.

[KMB04] Cem Kaner, Senior Member, and Walter P. Bond. Software engineering metrics: What do they measure and how do we know? In *In METRICS 2004. IEEE CS*. Press, 2004.

[MLH14] S. McIntosh, K. Legere, and A.E. Hassan. Orchestrating change: An artistic representation of software evolution. In *Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week - IEEE Conference on*, pages 348–352, Feb 2014.

[MS11] Silvia Miksch and Heidrun Schumann. *Visualization of time-oriented data*. Springerverlag London Limited, 2011.

[Mur13] Scott Murray. *Interactive Data Visualization for the Web*. O'Reilly Media, 2013.

[MW$^+$47] Henry B Mann, Donald R Whitney, et al. On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics*, 18(1):50–60, 1947.

[OK09] Dawn M. Owens and Deepak Khazanchi. Software quality assurance. *Handbook of Research on Technology Project Management, Planning, and Operations*, pages 242–260, 2009.

[Osm12] Addy Osmani. *Learning JavaScript Design Patterns*. O'Reilly Media, 2012.

[RPH$^+$11] Foyzur Rahman, Daryl Posnett, Abram Hindle, Earl Barr, and Premkumar Devanbu. Bugcache for inspections: Hit or miss? In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ESEC/FSE '11, pages 322–331, New York, NY, USA, 2011. ACM.

[RR08] Leonard Richardson and Sam Ruby. *RESTful web services*. O'Reilly, 2008.

[SHK98] Sandra A. Slaughter, Donald E. Harter, and Mayuram S. Krishnan. Evaluating the cost of software quality. *Commun. ACM*, 41(8):67–73, August 1998.

[SM99] Heidrun Schumann and Wolfgang Muller. *Visualisierung: Grundlagen und allgemeine Methoden*. Springer DE, 1999.

[TS09] Christoph Treude and M-A Storey. Concernlines: A timeline view of co-occurring concerns. In *Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on*, pages 575–578. IEEE, 2009.

[TS10] C. Treude and M. Storey. Awareness 2.0: staying aware of projects, developers and tasks using dashboards and feeds. In *Software Engineering, 2010 ACM/IEEE 32nd International Conference on*, volume 1, pages 365–374, May 2010.