Development of a dashboard as part of PinDown’s new graphical user interface

LUND UNIVERSITY
Campus Helsingborg

LTH School of Engineering at Campus Helsingborg
Department of Computer Science

Bachelor thesis:
Mai Linh Nguyen
Erik Samuelsson

© Copyright Mai Linh Nguyen, Erik Samuelsson

LTH School of Engineering
Lund University
Box 882
SE-251 08 Helsingborg
Sweden

LTH Ingenjörshögskolan vid Campus Helsingborg
Lunds universitet
Box 882
251 08 Helsingborg

Printed in Sweden
Media-Tryck
Biblioteksdirektionen
Lunds universitet
Lund 2016

Abstract

Dashboards are used in many different situations where it is necessary to be able to display information on a single screen, in a way so that the information can be monitored at a glance. This thesis explore the possibility of using a dashboard for the new graphical user interface of PinDown, an automatic debugging tool developed by Verifyter. The dashboard will be used to monitor projects from a single view.

The thesis investigate the current implementation of the interface in terms of usability as a reference point for what needs to be improved with the new design. We present three different frameworks for web application development, Ruby on Rails, Django and Play, and compare them against each other with respect to backwards compatibility, ability to obfuscate, compile to WAR and server-side thread safety. We evaluated the dashboard design through user tests of the developed proof of concept.

The result of this thesis was a prototype of the new graphical user interface which can be further built upon. The prototype was designed according to reliable design principles and the underlying architecture was carefully chosen, to make a robust basis for further development. For back end development we concluded that any of the mentioned frameworks could be used. The decision to use Play framework was based on our own experience with the Java programming language and the current implementation being written in Java. To visualize data in graphs we used NVD3, a charting library written in D3.js and to make the application responsive we used Bootstrap, a front end framework that makes the design respond to resize and different screen types.

Our conclusion is that the design met Verifyter's needs and that the dashboard should improve the usability of PinDown. The evaluation did not lead to any redesign but we found a few usability issues that were corrected.

Keywords: Dashboard design, dashboard development, usability, framework comparison.

Sammanfattning

Dashboards används i många olika situationer där det är nödvändigt att kunna visa information på en skärm, på ett sätt så att informationen kan ses över med enkelhet. Detta examensarbete utforskar möjligheten att använda en *dashboard* för PinDowns nya användargränssnitt. PinDown är ett verktyg för automatisk debugging skapat av Verifyter. Dashboarden kommer användas för att bevaka projekt.

Examensarbetet undersöker den nuvarande implementationen av användargränssnittet i form av användbarhet för att använda som referenspunkt för vad som bör förbättras i och med den nya designen. Vi presenterar tre ramverk för webbapplikationsutveckling, Ruby on Rails, Django och Play, och jämför de mot varandra med avseende på bakåtkompatibilitet, möjligheten att fördunkla (eng. obfuscate) kod, kompilera till WAR och trådsäkerhet på serversidan. Vi evaluerade *dashboardens* design genom användbarhetstester av den implementation som gjordes i syfte att bevisa konceptet.

Resultatet av examensarbetet var en prototyp av det nya användargränssnittet som sedan kan användas för vidareutveckling. Prototypen följer beprövade designprinciper och den underliggande arkitekturen är noga utvald, för att skapa en robust grund för fortsatt utveckling. Vi kom fram till att vardera av de tre nämnda ramverken kunde användas för back-end-utveckling. Valet att använda Play framework baserades på vår egen erfarenhet av programmeringsspråket Java, samt att den nuvarande implementationen är skriven i Java. För att visualisera data i grafer använde vi NVD3, ett grafbibliotek skrivet i D3.js. För att få designen att reagera på förändringar av fönsterstorlek och olika skrämtyper användes Bootstrap.

Vår slutsats är att designen möter de krav som Verifyter ställer och att dashboarden bör förbättra användbarheten av PinDown. Evalueringen ledde inte till någon omkonstruktion av dashboarden, däremot hittades några problem med användbarheten som åtgärdades.

Nyckelord: *Dashboarddesign*, *dashboardutveckling*, användbarhet, jämförelse av ramverk.

Foreword

First of all, we would like to thank Daniel Hansson and Patrik Granath for the opportunity to write this thesis.

We would also like to thank our supervisor Dr. Emelie Engström for her guidance in writing this thesis.

List of contents

1 Introduction 1
1.1 Motivation 1
1.2 Goal and questions 1
1.3 Delimitation 2
1.4 Terminology 2
1.5 Approach 2
1.6 Outline 2
2 Background 3
3 A brief introduction to PinDown 4
3.1 What is PinDown? 4
3.2 Usability of the current GUI 5
4 Dashboard architecture 8
4.1 Front end 8
4.1.1 JavaScript libraries 8
4.1.2 JavaScript libraries comparison 11
4.2 Back end 11
4.2.1 Web frameworks 12
4.2.2 Frameworks comparison 16
5 Dashboard development 17
5.1 Methodology 17
5.1.1 Requirement elicitation 17
5.1.2 Dashboard type selection 17
5.1.3 Dashboard design 19
5.2 Dashboard goals 20
5.3 Dashboard selection model 20
5.4 Dashboard design 21
5.4.1 Dashboard design principles 21
5.4.2 Dashboard components 23
5.4.3 Project view mockups 24
5.5 Presenting the design 28
6 Evaluation 29
6.1 Scenarios 30
6.2 Follow-up questions 31
6.3 Improving the design 32
7 Conclusion 34
7.1 Questions 34
7.1.1 What are the pros and cons with the current implementation in terms of usability? 34

7.1.2 What web frameworks are of interest, and how are they different from each? 34
7.1.3 What needs does Verifyter have when it comes to design and functionality of the new GUI? 35
7.1.4 How should the project dashboard be designed? 35
7.2 Future work 35
References 36
Appendix 38
A. PinDown 38
B. Mockup 40
C. User tests 41

1 Introduction

This thesis was commissioned by Verifyter AB, a software startup company which has developed a verification tool called PinDown that automatically debugs test failures. The goal of this thesis was to design a dashboard for one of the features of the graphical user interface. Based on a set of criteria we looked into which frameworks would be good options for implementing the design and for future development of the web application.

1.1 Motivation

Verifyter was founded in 2010. With PinDown, the company offer a tool that automate the processes of diagnosing soft- and hardware failures, thus allowing developers, who might spend valuable time doing the diagnosis manually, to spend their time on development.

Users interacts with PinDown through a web application, it has four panes: Test, Results, Open Bugs and Settings (See Appendix A). Under the Test pane the user can view information about the latest test run. Results and Open Bugs present additional information and under the Settings pane the user can add new projects, edit or remove projects and change system settings.

We think PinDown is a powerful tool, as it free up developers from doing manual work, but it could benefit from a new GUI. As an example we could look at graphs as they are useful to convey a lot of information in one area of the screen and as such they are an important component for this kind of user interface. In the current GUI there is the option to see graphs, but the user has to navigate to the Result pane and click on the graph icon in the upper-right corner. The graphs also doesn't function properly as it happens sometimes that the user need to refresh the page in order to show graphs. With a new GUI Verifyter is looking to address some issues as well as to update the overall look and feel.

1.2 Goal and questions

In this thesis we made a design for the project dashboard. We also investigated different frameworks that can be used for developing web applications and implemented the dashboard as a proof of concept.

The thesis consisted of two problems. The first one was to design a new project dashboard and the second was to implement this design as part of PinDown's new graphical user interface. For the second problem we chose frameworks for web application development to work with. To better understand the problems we worked with these questions:

Question 1: What are the pros and cons with the current implementation in terms of usability?
Question 2: What web frameworks are of interest, and how are they different from each other?
Question 3: What needs does Verifyter have when it comes to design and functionality of the new GUI?
Question 4: How should the project dashboard be designed considering the following questions: What are Verifyter's needs? How will user experience be improved?

1.3 Delimitation

In this thesis we've made a comparison between three frameworks, Ruby on Rails, Django and Play.

Only the overview and the project dashboard were implemented.

1.4 Terminology

Throughout the thesis we will refer to the *overview* and the *dashboard*. The overview is a view of a list of current projects, listed individually or as part of a group. From the overview the user can select a project and explore it further with the dashboard. The dashboard is a view with graphs, logs and history of a specified project. The dashboard is also referred to as the *project view* or *project dashboard*. We think that the reader should have no trouble to understand what we mean from the context.

Furthermore in chapter 3 and 6 we mention *labs*. With labs we mean a form of training in which the user will work with a set of tasks.

1.5 Approach

For chapter three and four we made analysis based on different sets of criteria. To give our opinion on these criteria we researched literature, documentation, forums and mail groups. For chapter three we worked with PinDown to build our own opinion of it and for chapter four we completed some of the basic introductory examples for each of the frameworks we researched to get an initial point of view. For chapter five we followed a process described in 5.1 Methodology.

1.6 Outline

In the first chapter the purpose and problems are introduced. The second chapter introduce the theoretical background. Chapter three and four discuss question one and two, and the result of their respective pre-study. These questions aim at understanding the current implementation and what frameworks to use for the new GUI. In chapter five question three and four are discussed and the methodology used to answer them, followed by the result and a presentation of the new GUI. In chapter six the work is evaluated and in chapter seven we'll discuss our conclusion. Figures and tables are numbered based on their respective chapter.

2 Background

There are four questions that we have focused on in this thesis, the first three can be thought of as sub-questions leading up to the fourth question, which can be thought of as the main research question. The first question aim at investigating the usability of the current implementation to find out what needs to be improved. The second question aim towards which framework to use and the third question investigate what requirements Verifyter has of the new GUI. The goal of the fourth question is to find out how the dashboard should be designed. The development of a dashboard requires careful planning and consideration. Before the development begins it is useful to have a clear definition of what a dashboard is, or at least should be. We have chosen this definition by Stephen Few:

“A dashboard is a visual display of the most important information needed to achieve one or more objectives; consolidated and arranged on a single screen so the information can be monitored at a glance.” (Few 2006)

This definition gives a robust foundation from which the development can take place. Once it has been decided what type of information the dashboard should display, one needs to decide how. There are many design principles that can be used when designing a dashboard. In *Information Dashboard Design* (Few 2006) Stephen Few give his deliberate opinion, state facts and argument over design choices, to give the reader knowledge in how to design to make an optimal dashboard. However, design principles are not exclusive to dashboards, and most of them origin from other fields e.g. psychology and the study of human visual perception. In *Information Visualization* (Ware 2000) Colin Ware presents, explains and discuss scientific research of human perception which has led to some of the design principles that we in turn have used in this thesis.

For the graph components we’ve made use of Tufte’s principles of data-ink ratio, found in *The Visual Display of Quantitative Information* (Tufte 2001). A concept of min-maximizing the area used for valuable information.

When investigating usability concerns we’ve worked with Lauesen’s definition given in *Software Requirements – Styles and Techniques* (Lauesen 2002). It consist of five factors; ease of learning, task efficiency, ease of remembering, subjective satisfaction and understandability. When setting up our user tests we had found *Usability Testing Essentials* (Barnum 2011) to be relevant and insightful, as she is very experienced in the field.

On the subject of developing a dashboard we found that *Dashboard Development Guide* (Staron 2015) had a process of working which we followed with a few modifications to make it fit better for our thesis. When working with design goals we found that Flesch’s thesis *Design, Development and Evaluation of a Big Data Analytics Dashboard* (Flesch 2014) was relevant to us, even though his dashboard had a different purpose to ours.

As for source criticism. All of the decisions that have had an impact on the thesis has come from a source which we consider to be trustworthy, e.g. from authors who are respected in their field or peer reviewed. Regarding technology choices we’ve had to read a lot of forum threads to further our knowledge, and either tried it out ourselves or researched code documentation.

3 A brief introduction to PinDown

In this chapter we'll give a brief introduction to PinDown and then with Lauesen's definition of usability we'll look at each factor and give our opinion. We'll suggest what we think are some pros and cons of the current implementation and how we can apply this knowledge in our own design.

3.1 What is PinDown?

In short, PinDown is a tool for automatic debugging of regression failures. This means that when a bug is introduced to a feature, PinDown is able to track when and by whom the faulty code was committed. By making a failing test pass, PinDown proves that the faulty code was correctly identified.

PinDown is integrated to a regression test system in two steps. The first step is to give PinDown information about the output from the regression test script, that is, the structure of the test results. Next, PinDown is provided with scriptable steps which let it run individual tests it wishes to debug. These are the same steps that an engineer would do during manual debugging, to reproduce a specific failure.

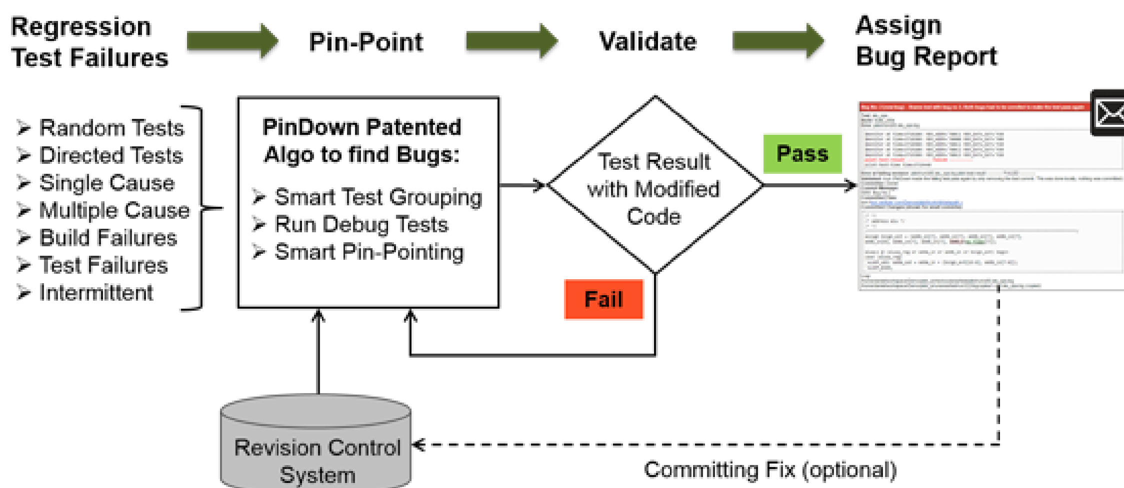

Figure 3: The automatic debug process of PinDown. *Used with permission from Verifyter.*

Figure 3 shows the process that PinDown follows. It starts with the regression test results produced by the user's test scripts. Then, PinDown will find when and with which commit the regression bug occurred. PinDown finds when the test started to fail by running older snapshots of the revision control system. To make sure that this is the only commit that causes this test or build to fail, it patch the code locally in a way such that the bad commit is undone. If the test that was failing now passes, a bug report is issued to the person who made the bad commit. Else, if the patch causes a test or build failure, e.g. if there are two or more bad commits that affect the test results, the debug process continues.

3.2 Usability of the current GUI

To identify the pros and cons in terms of usability of the current GUI we used the definition of usability given in *Software Requirements* (Lauesen 2002, p. 256). The definition is made out of five factors:

"

1. *Ease of learning: How easy is the system to learn for various groups of users?*
2. *Task efficiency: How efficient is it for the frequent user?*
3. *Ease of remembering: How easy is it to remember for occasional user?*
4. *Subjective satisfaction: How satisfied is the user with the system?*
5. *Understandability: How easy is it to understand what the system does?*

"(Lauesen 2002)

For each factor in Lauesen's definition of usability we'll discuss our experience with the tool and also what some pros and cons are that we found while trying it out. When discussing these factors we'll do it in the context of PinDown.

Factor 1: Ease of learning

When talking about ease of learning, you also have to consider who the user is. What sort of background does the user have? Someone who is familiar with the terminology might be able to pick up on how to work the tool as they explore it, while someone who does not might consider having a look at the manual first, read through the documentation and work with labs to acquire knowledge. Exactly how someone will learn to operate a tool is hard to predict, and the goal of this factor is not to determine how, but rather how to make it as easy as possible for users to learn in a way suitable for them.

We would argue that a user will most likely need to work with the labs as a minimum to be able to configure the tool properly, this is because PinDown is integrated into an existing system and therefore it needs to know a few things before it can start its work. But what if we only consider the interface, how easy is it to learn? Can someone without training complete a given task by intuition? Depending on the task we think that it could be done because some parts of the interface are self-explanatory, but for more complex tasks we think that the user needs to work with the labs first to grasp the concept. However, we are intrigued by this question and in our design we would like to see if looking up information in our design is intuitive or not.

Factor 2: Task efficiency

When isolating tasks, i.e. when looking up a value in a graph, or read a log. We found that the current interface is efficient. It is a straightforward process and it doesn't change over time which means that the time it takes to complete a task is likely to be reduced with experience. However we found that performing a combination of tasks is not as efficient. E.g. if you wanted to look up a value from a graph, and also look at the PinDown debug log, you would first need to go to the Results pane, then to see the logs you would have to change pane to Test, and access the PinDown log via the button in the lower-right corner. See appendix A.

In our design we have tried to make the combination of tasks more efficient by having all of this information present within the dashboard and as such eliminate the need to switch between different views.

Factor 3: Ease of remembering
This is an important factor and we would argue that the two previous factors depend on it. The reason we say this is because if it is hard to remember how to complete a task, it doesn't matter how efficient it is or how easy it is to learn, since progression would be either too slow or non-existent. We think that ease of remembering can also be thought of in terms of a learning curve, and how steep it is. The steeper the curve is, the more important it is that it is easy to remember how the interface works, where to find information and how to interpret it. However, Lauesen adds to this factor, "*for occasional user*" and because PinDown is not likely to have occasional users we think that it is fine to expect some practice. That said we don't think that the learning curve is very steep as reflected in factor 1.

Factor 4: Subjective satisfaction
From our own brief experience it is hard to draw any conclusion when it comes to satisfaction. While the experience wasn't unpleasant in any way, we think that the problem with task efficiency discussed in factor 2 could be a source for dissatisfaction. If we only consider esthetic qualities we think that it does its job well and it is something we've also considered in our own design. We've moved away from things that are flashy and purely esthetic as these are things that users will get tired of eventually, something that Stephen Few comes back to time and time again in *Information Dashboard Design (*Few, 2006).

Factor 5: Understandability
This is also an important factor, perhaps the most important factor of them all. What good is a system that is easy to learn and very efficient, easy to remember and pleasant to use, if you have no idea of how and why to use it? It should make sense to complete a task, or a combination of tasks. The interface needs to be intuitive to use and there needs to be a clear connection between e.g. pressing a button and displaying a message. If the users doesn't know what the button is doing, or why the message is displaying when the button is pressed, then it's a reason for concern. It doesn't mean that every user needs to understand everything at first glance, but after some exploration of the interface it should become apparent. We think that a way to support this factor is to design the interface so that it can be easily understood with a brief presentation, through documentation or labs. It shouldn't be a worry that things aren't understood at first glance, but if things are still confusing even after the user have worked through labs then perhaps the design need some rework.

Table 3: A summary of our conclusion of the usability of the current GUI.

Factor	Pros	Cons	Reasoning
Ease of learning	X		We found that learning the system is easy, but it does require the user to take some time to do labs or read documentation. This is not a bad thing as it will give opportunity to show the user the potential of the tool.
Task efficiency		X	We found that for doing individual tasks it wasn't a problem, but a combination of tasks require the user to switch between views, making tasks more complex than they need to be.
Ease of remembering	X		Most of the components of the interface are self-explanatory e.g. the PinDown log button brings up the PinDown log. This helps with remembering.
Subjective satisfaction		X	Because it is subjective we can only really speak for ourselves, and we weren't dissatisfied by any means. However, because of the task efficiency we count it as a drawback and this is something we'll try to address with the new design.
Understandability	X		We had no trouble understanding the current interface as we found it intuitive, and also well explained in training documentation.

4 Dashboard architecture

Before we could start designing the GUI we had to decide which front and back end frameworks to use in our development, the reason to do this before starting the design was so that we could be certain that we could achieve backwards compatibility with the technology we would choose.

4.1 Front end

In this section we'll first present the JavaScript libraries that we've used, and then in the comparison we'll give our reasoning to why. For the front end side Verifyter made a list of criteria that they expect to be satisfied. The criteria were:

- Backward compatibility: The client-side shall support older web browsers. Internet Explorer 11 and later version, Firefox 16.0.1 and later version.
- Responsiveness: The client-side should have responsive design and cross-browser rendering.
- Offline JavaScript libraries: All of the JavaScript libraries that are used for implementation of client-side shall be hosted locally or on an intranet.

To make the client-side responsive we needed a HTML framework. From early meeting with Verifyter they recommended that we made a research about Bootstrap which is a HTML, CSS and JavaScript framework for development of responsive design.
When it came to JS libraries, we could make use of any JS library, as long as it met the client-side's criterion of working offline.

There is a fair amount of JS libraries that achieve the same goal, but a set of popular JS libraries were chosen for data visualization to be compared against each other. They were Google charts, D3.js and Van charts. We made comparisons between these libraries against our criteria to select the most appropriate one.

4.1.1 JavaScript libraries

Bootstrap (http://getbootstrap.com/)
In order to make the web application responsive for different devices we used Bootstrap which is a front-end library/web framework that facilitates the development of dynamic websites and web applications. Bootstrap supports responsive web design which means that the layout of the web page or web application regulates dynamically depending on the properties of the platform.

Since the components of the dashboard are organized in a grid pattern (see 5.3.3), it was easy for us to apply Bootstrap in our design, because Bootstrap makes responsive layout based on grid systems. Figure 4.1 show what the grid system looks like.

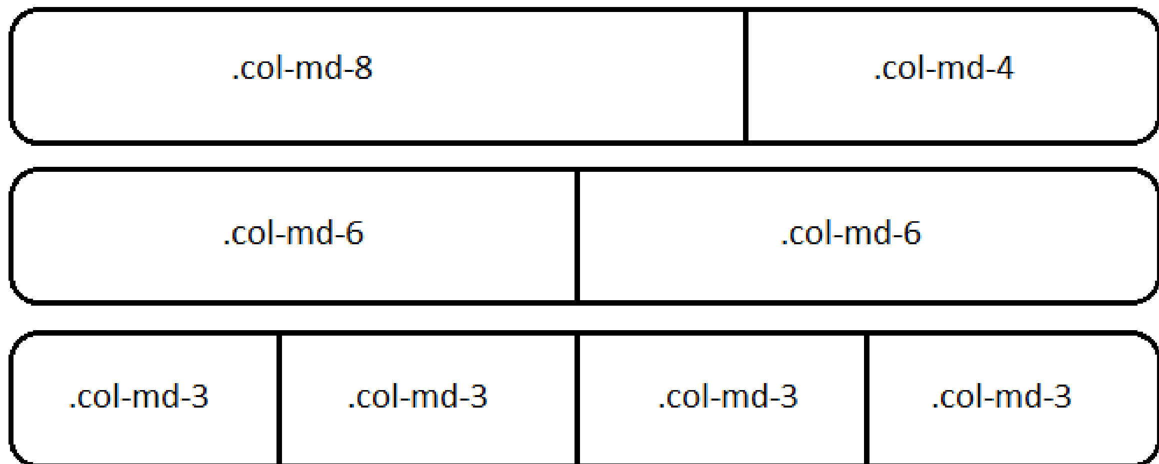

Figure 4.1 Bootstrap's grid system.

It uses the grid systems to create layouts through a sequence of rows and columns. Bootstrap contains predefined classes for different layout options, e.g. "md" in figure 4.1 indicates that this is the layout displayed on a desktop. For layouts that should be displayed on phones, the class "xs" would be used instead. For tablets "sm" and for larger desktops "lg". These classes can then be used in combination e.g. to make the layout change depending on the device used you could write:

```
<div id ="tableDiv" class="col-md-6 col-xs-12"> some code </div>.
```

In addition to support of responsive design Bootstrap also has many reusable interface components like buttons, typography, forms etc. Which we used to created our user interface.

D3.js and NVD3
To visualize data in graphs we used D3.js (Bostock, M. et al, 2011). It is a JS library that make use of SVG, HTML5 and CSS to create visualizations. When creating a graph with D3.js one usually follow these steps:

- Selection: The D3.js functions `select()` and `selectAll()` return an element or a set filtered on elements from current document. These elements could be filter like in JQuery (a JavaScript library) by unique identification (#id), by class (.class), by tag, by attribute or by place in hierarchy. Once an element has been selected the content can be manipulated by other operators e.g. `append('svg')` adds a svg-element to the selected element and `attr('width','35')` defined the width of the svg-element to 35 pixel.
- Data joins: D3.js binds input data to element by the `data()` operator which takes data arrays in different formats as parameter. The data formats can e.g. be JSON, Comma-Separated Values (CSV) or geoJSON. In order to read other formats one could write a JavaScript function for that format.
- Appending nodes depending on data: Once the data is bound to selected elements, modifying content based on input data will follow these steps: appending new nodes, updating existing nodes, destructing nodes.
    1. Appending nodes: The `enter()` function and chained functions after it will be invoked for each item in the dataset that aren't already represented by a DOM node.
    2. Updating nodes: The `update()` function will be invoked for all nodes in in the selection that have a corresponding item in the dataset. This step updates the nodes when the value of the item has been changed.
    3. Destructing nodes: The function `exit()` will be invoked for all nodes in the selection that doesn't have a corresponding item in the dataset. The step remove nodes from the document.

Putting the above steps together could look something like:

```
var svg = d3.select('#piechart')
        .append('svg')
        .attr('width', 200)
        .attr('height', 200)
        .append('g')
        .attr('transform', 'translate(100,100)');

var path = svg.selectAll('path')
        .data(pie(dataset))
        .enter()
        .append('path')
        .attr('d', arc)
        .attr('fill', function(d, i) {
          return color(d.data.label);});
```

We wanted to make our stacked area charts interactive, so that when the user mouse-over the graph they would see a tooltip with information about the node they're currently holding the mouse over. For this we used NVD3 which is a library that provides re-usable charts for D3. It simplified the code a lot by replacing most of the native D3 code with function calls to pre-built NVD3-functions.

JQuery Treetable
To organize projects in the overview in a tree structure we used a plug-in for JQuery called JQuery Treetable.

4.1.2 JavaScript libraries comparison

After researching Bootstrap we were confident that bootstrap met our first criterion of backward compatibility. Bootstrap supports Firefox, Internet Explorer and Google Chrome on Windows. It behaves well even on web browser for Linux though they are not officially supported. Bootstrap let us achieve the criterion of responsiveness and the library can be hosted locally which means that bootstrap meet all of our criteria.

Table 4.1 shows that each of the chart libraries we looked into met our criteria expect for Google Charts (https://developers.google.com/chart/) that failed on one criterion: to be able to host locally. To render charts with Google Chart it is required that the machine has live access to Google's jsapi, through their website. This means that we're not permitted to host it locally or on an intranet.

Table 4.1 The result of the comparison of JavaScript libraries against the criteria.

Library	Backward compatibility	Responsive	Hosted locally
Google Charts	Yes	Yes	No
Van Charts	Yes	Yes	Yes
D3.js	Yes	Yes	Yes

Our decision to go with D3.js was based on two things. First of it is open source whereas Van Charts (http://www.vancharts.com/) is not. Second, it has a large community and is very well documented. Mike Bostock, one of the developers of D3.js have created many useful examples and tips on how to create many different types of charts. D3.js is a very large and complex library and it takes a lot of time to get comfortable using it. We strongly recommend Curran Kelleher's screencasts to get to know the basics of D3.js, found on: https://github.com/curran.

4.2 Back end

The decision to implement the new web application using a web application framework was based on the idea that a framework makes development easier by simplifying things. There is a vast amount of frameworks available and it can be difficult to choose between them. To give guidance in the matter a set of criteria were expressed after a meeting with Verifyter, where we discussed their user base, their current implementation and their requirements. Among the most important was backward compatibility with Java virtual machine (JVM). Another criterion to consider was the ease of potentially having to learn a new language like Ruby or Python. This aspect meant that a large community and good documentation would also have to be considered.

Ruby on Rails (http://rubyonrails.org/), Django (https://www.djangoproject.com/) and Play framework (https://www.playframework.com/) were chosen because of their popularity and strong communities, to be compared against each other on the following criteria:

- Backward compatible with JVM.
- Compile to WAR.
- Able to obfuscate.
- Server-side thread-safe.

For both the front end and back end, each criterion was written after a discussion with Verifyter about the new GUI. To fit within our time frame each criterion would have to be briefly researched through documentations, discussion forums and mail groups. The goal of this stage was to find suitable frameworks to use in development of the overview and project dashboard. For each framework we tried a simple “Hello world!” example. After we had decided upon frameworks we made a small feasibility study to ensure backwards compatibility by putting together a small example that we compiled on a machine with Java 6 and then ran it on another machine that was offline.

In this section we’ll describe the architecture of Django, Ruby on Rails and Play framework. Since each of these frameworks follow the Model-View-Controller (MVC) design pattern, we will start with a brief introduction of the pattern and then describe the frameworks individually. Then we’ll present the result of the comparison.

4.2.1 Web frameworks

The MVC design pattern splits a given software application into three interconnected components which is Model, View, Controller.

- Model: The model contains all the data and the state of the application domain. It manages all the business logic and have no knowledge of the user interface.
- View: The view presents data to the user by generating the user interface. It is passive and doesn’t do any processing. Different views can access the same model for different purposes.
- Controller: The Controller contains classes which are used for communication between the models, the views and the user. It receives events from the user, interact with the model and display the appropriate view to the user.

The purpose behind separating these components is to distinguish the user interface logic from business logic. This means that each component can be as independent of the other as possible. Changes that are made to one component don’t affect changes that are made to the others.

Play framework
In *The main concepts* of Play framework's documentation the MVC pattern applied to web architecture is described as follows.

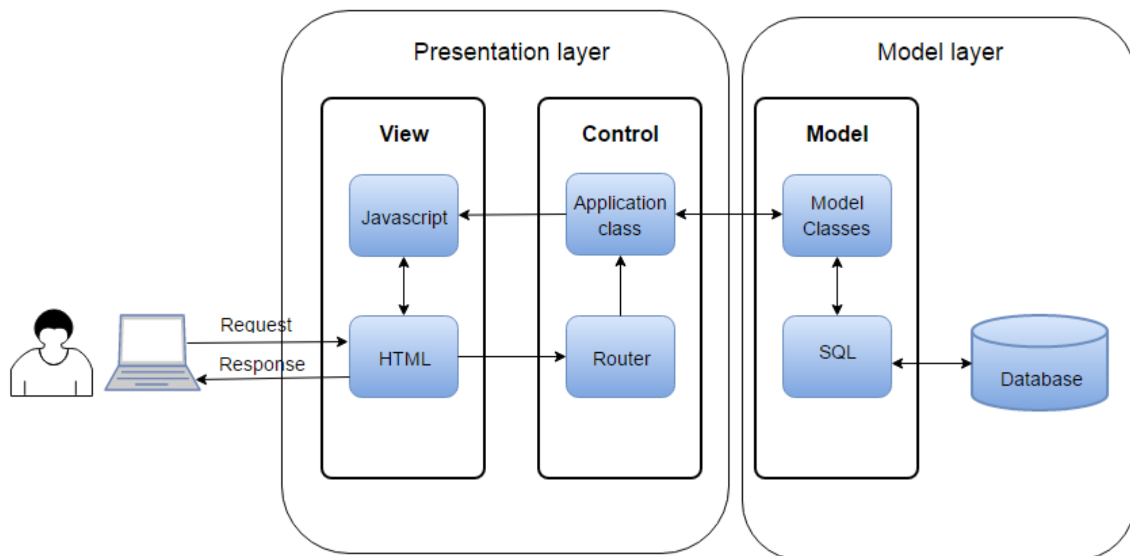

Figure 4.2 Play framework's web architecture.

The MVC pattern separates the play application in two layers: The Presentation layer and the Model layer. The Presentation layer contains the View and Controller layer. See figure 4.2.

- The Model layer: contains classes that describes behaviors and property of objects.
- The View layer: The function of the view layer is to handle the data displaying. It generates the data received from the model into views which is user interface. The view is produced in a "Web Format" like HTML, XML or JSON.
- The Controller: The controller listens to events (HTTP request) from user action, extracts the data encapsulated in the event, processes them, and applies changes to the database by the model classes if needs and then responds to the event by sending view with encapsulated data in a HTTP response.
-

In figure 4.2 you can see a HTTP request life cycle which clarifies how it works between layers.

- The user sends a HTTP request by interacts with the user interface (HTML)
- The router tries to map the specific route for this request. Once the specific route was found, the corresponding action method in application class will be invoked.
- The invoked method code is executed and makes required changes to the database by invoking methods from model classes. Data sends then back to application class.
- The action method sends data to corresponding JavaScript function which handles view rendering.
- A HTTTP response contains the HTML, XML or JSON file sends back to the client as response.

Ruby on Rails

The working principles of Ruby on Rails (RoR) framework is very similar to Play framework, since the RoR framework also follows the MVC design pattern. The definitions of these components Model, View, Controller in RoR is identical with the definitions on Play framework except the name of the components are called different.

- Model (ActiveRecord): The model manages relationship, validation, association, transactions between the objects and the database. This Model is implemented in the ActiveRecord library.
- View (ActionView): This subsystem is implemented in the ActionView library. The view defines how the data should be presented. Which view should be displayed is triggered by the controller's decision.
- Controller (ActionController): This subsystem is implemented in ActionController and its work tasks are direct traffic, invoke the action methods for requested URLs, retrieves data from database through the models and organizing that data and calls the specific view to display that data.

The framework treats a HTTP request in steps describes below:

- A HTTP request comes in through user input.
- Rails redirects the incoming request to a specific controller's method by using the routers.rb file (the routes.rb file contains a list of available action and type of action - get, post and patch in the controller).
- The method in controller that mapped to the specific route will be invoked. Controller is a Ruby class which have methods corresponds to each specific route.
- The controller retrieves data in the database through the models
- All retrieved data is saved in a temporary memory location.
- The controller calls the specific view, passes in the retrieved data to that view and renders HTML file.
- The controller sends the HTML/XML and metadata to the browser to display on the screen.

Django

Although Django appears to be a MVC framework, there are some things that are different with the MVC pattern used in Django, compared to Play and Ruby on Rails. In *The Django Book* (Holovaty and Kaplan-Moss, 2009, chapter 5) they describe how the three components are divided in Django:

- Model: The model has a data-access role and are managed by Django's database layer
- View: The view handles which data to display and how to display it. This is managed by the views and templates.
- Controller: The controller's role is to decide which view function will be invoked and send HTTP Response to user. This is conducted by the framework itself.

Since the controller is managed by the framework itself and developers work most on the template, view and model, Django has been referred to as an MTV framework.

- Model: The model layer contains everything about the data: the behaviors and property of the data, relationship between the data and how to access and validate it.
- Template: The template is presentation layer. It contains all the decision on how thing should be displayed
- View: The business logic layer. View is referred as Controller in other web framework. This layer manages the access to the model and which template will display the retrieved data from model.

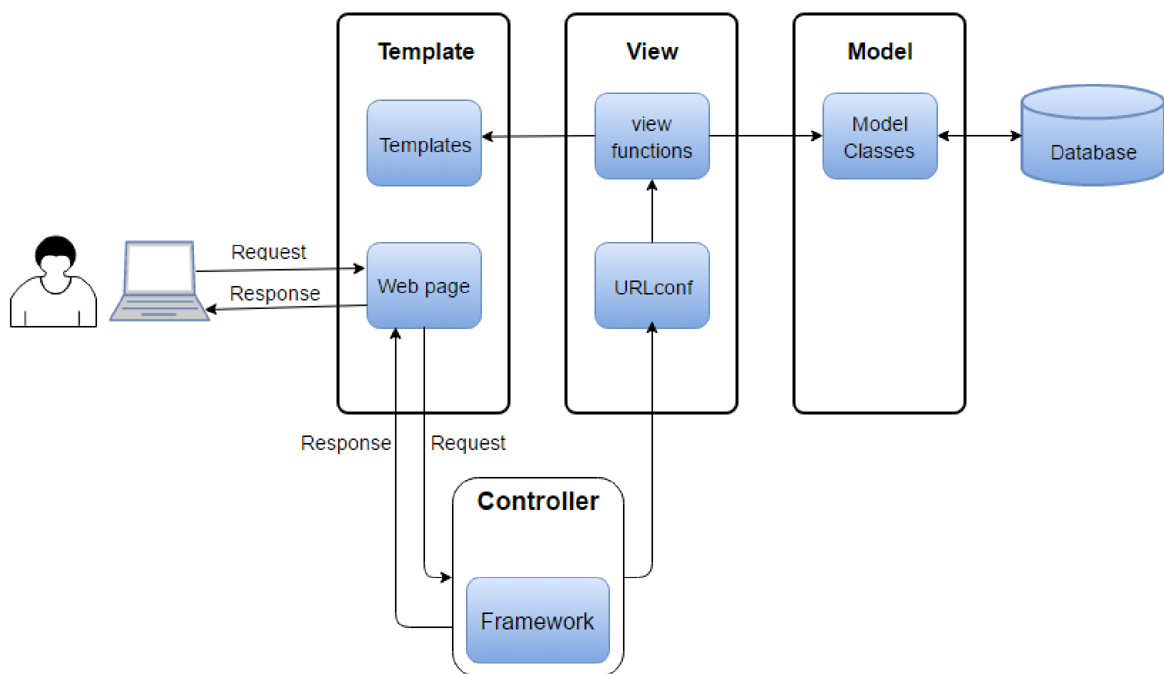

Figure 4.3 Django's web architecture.

Figure 4.3 show a HTTP request life cycle which clarifies how it works between components.

- The user sends a HTTP request by interacts with the user interface (Webpage)
- When a request comes in the framework determines the root URLconf by checking the ROOT-URLCONF setting (All this happens in the controller part which the developer don't need to handle)
- When the controller has found the root URLconf, it will check all of the URLpatterns in the URLconf in order to find the one that matches the requested URL.
- If it finds a match it will invoked the corresponding view function.
- The view function invokes the model to get requested data, mapped the data with associated template (which can be HTML or XML) and return an HTTP Response
- The framework transforms the View's HTTP Response into proper HTTP Response which result a web page.

4.2.2 Frameworks comparison

As mentioned in 2.2.2 we chose Ruby on Rails, Django and Play because of their large communities and because all of them are well documented. This aspect is important because it makes searching for information much less of a hassle. When making comparisons against the criteria, see table 4.2 we found that any of the three frameworks could be used. These criteria alone would not be enough to decide which framework to use, therefore we consider an additional factor, preference.

Table 4.2. The result of the comparison of frameworks against the criteria.

	Backward compatible with JVM	Compile to WAR	Able to obfuscate	Server-side thread-safe
Ruby on Rails	Yes (with JRuby, a Java implementation of Ruby)	Yes	Yes	Yes
Django	Yes (with Jython, a Java implementation of Python)	Yes	Yes	Yes
Play	Yes	Yes	Yes	Yes

To have preferences require experience and to be experienced with each framework would take a lot of time. After trying out the basic “Hello world” example we browsed different blog posts, articles and forums to try and see what other people had found and what their thoughts were. For this we set up a few guidelines for an opinion to be considered:

- The person should have experience with more than one framework.
- Claims made should be reasoned for and/or in some way backed up by a source.
- There must be no obvious bashing of other frameworks.

After reading through many forum threads and blog posts we came to the conclusion that while a lot of opinions where valid, it was hard to get away from one's own bias. As an example, reading through the blog post *Rails vs Django vs Play framework* by a software developer named Diogo Nunes, in which he compared each of the frameworks we investigated, we found ourselves wondering how well Play would hold up against the other two, rather than which framework he would actually recommend.

Yevgeniy Brikman, a software engineer who previously worked at LinkedIn, provides lots of useful links, as well as sharing his own knowledge, in this blog post *The Ultimate Guide to Getting Started with the Play Framework*. But at this point in time we had already decided to go with Play.

Reading what other people thought was interesting but it was our own experience with the Java language that led us to use Play. It would also be easier for Verifyter since there current solution is Java based, and any of the other options would require more installations on customers end.

5 Dashboard development

In this chapter we'll describe the process we worked with and then the result of each of the process steps will be presented.

5.1 Methodology

In *Dashboard development guide* (Staron 2015) there is a description of a dashboard development process consisting of five stages:

1. Requirement elicitation
2. Dashboard type selection
3. Dashboard design
4. Impact evaluation
5. Dashboard maintenance

For this thesis we found that the first three stages were of particular interest, as they are within our scope. The following is a brief description of the steps, as well as how and when we chose to differ from them in our process.

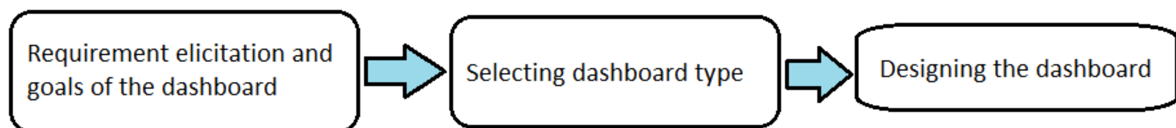

Figure 5.1 The first three steps of the *Dashboard development guide* (Staron 2015)

5.1.1 Requirement elicitation

In the first stage of the development process the goal is to identify stakeholders, information providers and users by making interviews in the organization, and then collect high level expectations. During this stage we worked out the goals of the dashboard as suggested by Staron (2015) by finding out what the information needs were and how it should be satisfied and visualized. In the *Dashboard development guide* (Staron 2015) there is a clear distinction between the different stages, however because part of our thesis was to compare different frameworks that would be used to develop the dashboard, we had already done so before moving on to the next stage, which was to find the technology to be used. In retrospect this meant that we could more freely move between stages and even if they were separated we could treat the first and second steps of the process as one step. The result of this stage were the goals of the dashboard, the elicited requirements and a paper mockup of the visual content.

5.1.2 Dashboard type selection

In the second stage we used the dashboard selection model (Staron et al. 2015) to identify the purpose of the dashboard in terms of what information it should display and how. The dashboard selection model is a model that helps with selecting properties of a dashboard depending on stakeholder's information needs. The model consist of seven dimensions where each dimension has two alternatives and stakeholders grade which one should have focus, from equal to full. Because part of the thesis was to evaluate different options of frameworks for front- and back end we were helped in the second stage as to which technology to choose based on our research. In turn this meant that we could use the

dashboard selection model to validate our choice, as well as elicit requirements that were missed in previous stage.

Staron (2015) suggest this stage to be concluded with a prototype as a feasibility study of the chosen technology. During the evaluation of different frameworks we had already set up a feasibility study of the technology that we had selected which meant that we could conclude this stage still with a simple paper mockup which we could use when moving on to the designing of the dashboard. Table 5.1 shows the different dimensions, their alternatives and a brief description of each.

Table 5.1: Dimensions and their respective alternatives as described in the *Dashboard selection model* (Staron et al. 2015).

Dimension	Description	Alternatives	Description
Type	What kind of visualization is needed?	Report / Dashboard	Report: require flexible format. Dashboard: require same structure every update.
Data acquisition	Input of data into the tool.	Manual / Automated	Manual: user enter data into the tool. Automated: data is imported.
Stakeholders	Defining the stakeholders.	Individuals / Group	Individuals: used by an individual. Group: used by a team.
Delivery	How data will be provided to stakeholders.	Fetched / Delivered	Fetched: user actively seek information. Delivered: information is delivered, e.g. through email.
Update	How data is updated.	Periodically / Continuously	Data can be updated periodically or continuously.
Aim	The aim of the dashboard.	Information / Decision support	Information: designed to spread information. Decision support: designed for specific decisions.
Data flow	How much processing of data to be done in the dashboard.	Raw data / Indicators	Raw data: no additional interpretation is done. Indicators: analysis models applied.

5.1.3 Dashboard design

During the third step of the process the dashboard design was created according to the requirements elicited during the first step and implemented using the selected technology from step two. The design process started with the use of paper mockups of the components of which the dashboard should consist. This made it possible to get quick feedback on the placement of the components. In *Information Dashboard Design* (Few 2006 p.113) Few describe two categories of which important data can be divided into; information that is always important and information that is only important at the moment. These two categories can be emphasized by different means, static and dynamic respectively. Few says: *"Because location is static, this is a variable that we can leverage to highlight information that is always important."*

In order for us to know which information should be regarded as important we asked that each component was ranked by Verifyter in order of importance. While it is true in most cases that each component in itself is important, it is necessary to establish an internal hierarchy to know which component should be emphasized over another. We allowed for components to be equal in importance, except for the most important component which could only be graded to one item. This was done so that the components could be arranged according to the different emphasis regions (Few 2006, p. 114). The reason to only let one item be the most important even though there are two regions with greater emphasis, we argue, is because it is much harder to utilize two regions efficiently than it is to utilize one region efficiently.

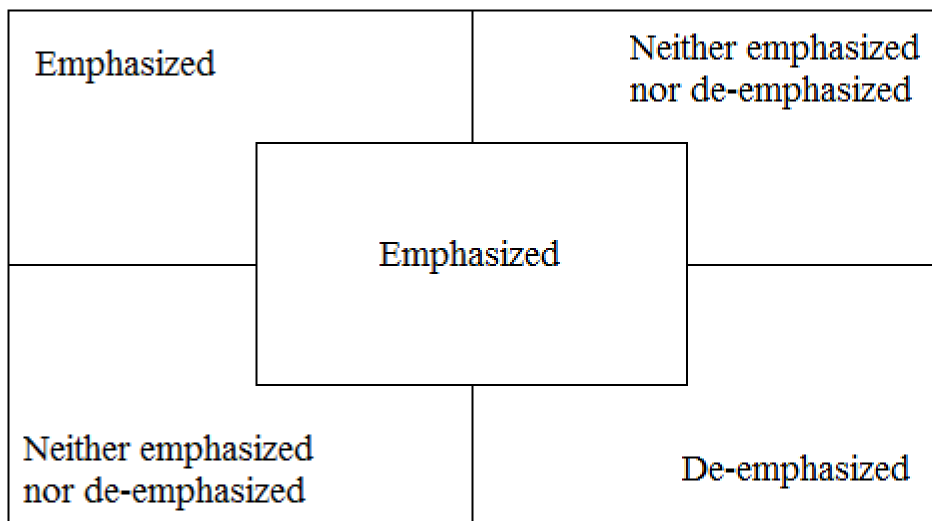

Figure 5.2: Different regions of the dashboard have different emphasis.

5.2 Dashboard goals

Following the process described in *Dashboard development guide* (Staron 2015) the first stage is dedicated to eliciting requirements and working with the goals of the dashboard. The goals can be split into two parts; the design and the functionality. Where the design is not just focusing on the look and feel but also the realization of it. The goals of the design were identified as:

- Responsiveness: in order for the dashboard to work well on different screen sizes and different devices types, e.g. laptops, phones, tablets, it needs to be responsive.
- Accessibility: our definition of accessibility is the same as the one expressed by Flesch (2014) which says: "The dashboard should be accessible as easily as possible for users. It should therefore have as few hard dependencies in terms of installed software, operating system or device type as possible." This also reflect the need of backwards compatibility. There is also another type of accessibility to consider and it is one in terms of usability for people with disabilities as mentioned in *Usability Testing Essentials* (Barnum 2011, p. 108), however this has not been considered in this thesis.
- Ease of use: the dashboard should be designed in a way so that users can work with it without any training. A way to achieve this is to design based on the concept of *Don't make me think* (Krug 2006).

The goals of the functionality derived from the requirements that we elicited. The dashboard should show trends in projects through graphs, and there should be logs that convey reports of bugs and PinDown log messages. The user should be able to look up previous runs and get details about them.

5.3 Dashboard selection model

Requirements for the dashboard was elicited through a continuous discussion and also from a presentation by Verifyter of their vision of the new GUI. While the purpose of the dashboard selection model is to find characteristics of the dashboard, we also used it as a way to elicit requirements missed by the discussion, and as a way to validate the requirements that we had elicited.

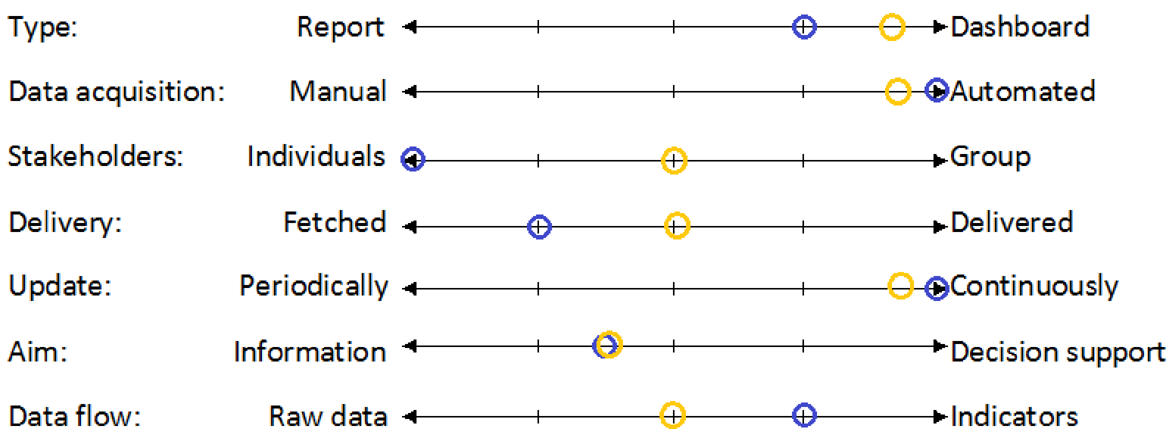

Figure 5.3 Graphical representation of the dashboard selection model, as described in *Dashboard Selection Model* (Staron et al. 2015). The blue and yellow rings indicates two stakeholders we asked to participate.

The result of the dashboard selection model is shown in figure 5.3. For the most part both participants had similar answers, which was somewhat to be expected. The interesting dimensions to investigate further with discussion were the ones that they were leaning towards different alternatives on, although there was not one dimension on which they both picked the opposites. Because the answer depends on how the participant interpret the question, we asked them to describe to us with a few sentences what they thought about each dimension. Table 5.2 shows a summary of what their expectations of the dashboard were.

Table 5.2 Summary of the answers to each dimension.

Dimension	Summary
Type	The dashboard should display data visualization with graphs and make use of logs to display messages. The layout should be fixed but responsive meaning that the layout will differ depending on the screen size, but should be the same among equal sizes.
Data acquisition	Data input is fully automated, imported from a database.
Stakeholders	The dashboard should be used by individuals for the most part, but could also be used by a team of developers to discuss a project.
Delivery	Stakeholders will seek the information they require but to some extent it would be good to have information sent by email.
Update	Data should be updated continuously.
Aim	The dashboard should provide information about the status of the project.
Data flow	Data should be visualized in graphs and in logs. The user is not expected to look at raw data but rather see trends in the graphs.

5.4 Dashboard design

The initial design of the dashboard was first laid out with the help of paper mockups, then it transitioned into a digital mockup which became a working prototype. The design principles which we have been researching focus on the placement of components and their graphical representation. First we will present the design principles and then we will present the design and the reasoning behind it, according to said principles.

5.4.1 Dashboard design principles

If we know what the user is looking for, we can help them find it by making it stand out from its surrounding. For example if you wanted to draw attention to a specific word within a text the easiest way to accomplish this would be to write it in bold. This is called pre-attentive processing and while words and letters aren’t pre-attentively processed, line thickness is. Pre-attentive processing is perhaps not a design principle as much as it is a phenomenon that occurs within our visual field, however, being aware of pre-attentive processing is useful when designing a dashboard. Colin Ware (2000) says: “In essence, pre-attentive processing determines what visual objects are offered up to our attention.”

Another way to look at it is if instead of drawing attention towards something, you'd wanted to draw attention away from something. In the above example when attention is drawn towards a single word, it is at the same time drawn away from the rest of the text.

If this happens unintentionally it can turn into a major concern, why it is important to factor in pre-attentive processing. In *Information visualization: perception for design* (Ware, 2000, p. 165) Ware presents four categories, each with several features which are pre-attentively processed. The categories are:

- Form, such as line orientation, length, width, size. Spatial groupings, added marks and numerosity.
- Color, different variety of hue and intensity.
- Motion, can be flicker or direction of motion.
- Spatial position, e.g. 2D position or convex/concave shape from shading.

Another phenomenon which occur in our visual field is the way we perceive form, e.g. we have a tendency to group things together based on shared characteristics. This tendency, among other perceptual phenomena are described by Max Wertheimer, Wolfgang Köhler, and Kurt Koffka as set of principles, called the gestalt principles. Gestalt meaning shape in German. The gestalt principles can be described as "rules of the organization of perceptual scenes" (Dejan Todorovic, 2008) and they aim to give understanding as to how certain objects in our visual field are grouped together. In our design we've made use of the following gestalt principles:

- The principle of proximity, objects that are located close to each other will be perceived as belonging to the same group. In dashboard design this principle can be used to direct the way users scan data, from left to right, or from top to bottom. The way to achieve this is by placing sections of data closer together, either horizontally or vertically depending on the direction it should be viewed.

- The principle of similarity, objects that are similar to each other in color, size, shape and orientation will be perceived as belonging to the same group. In a dashboard this can be used to make connections between objects that aren't placed close to each other or objects that are reoccurring.

- The principle of enclosure, objects that are enclosed by a visual border will be perceived as belonging to the same group. A visual border can be a line, or a background color.

To make sure that we didn't waste valuable dashboard space, nor distracted users from their tasks, we made use of Tufte's (2001) principle of data-ink ratio which says: *"Maximize the data-ink ratio, within reason."* Tufte explain the principle as: *"Every bit of ink on a graphic requires a reason. And nearly always that reason should be that ink presents new information."*

In *The visual display of Quantitative Information* (Tufte, 2001, p. 93) Tufte talks about "data-ink ratio", a concept in which the ink used to print a graph can be divided into ink that present data and ink that present non-data. In the example with a graph, the non-data would be the axis, and the data ink would be ticks and the values plotted. Tufte defines the data-ink ratio as:

"Data-ink ratio = data-ink / total ink used to print the graphic
= proportion of a graphic's ink devoted to the non-redundant display of data-information
= 1.0 – proportion of a graphic that can be erased without loss of data-information."
(Tufte 2001)

The definition give reason to also consider Tufte's two principles of erasing:

- *"Erase non-data ink, within reason."*
- *"Erase redundant data-ink, within reason."*

Where non-data ink fails to convey any statistical information and redundant data-ink depicts the same information over and over.

5.4.2 Dashboard components

The goal of the paper mockup described in 5.1.3 was to get feedback on the placement of the components. While all components are important externally, it is necessary for the design to distinguish between them internally and structure them hierarchical as this will help with the placement. The visual components of which the project view of the dashboard should consist of were identified as:

- Current graph – a graph that show the result of the tests that are currently running. A result can be either pass or fail. If no tests are currently run, the graph should show the last run tests for the specific project.
- Result graph – a graph that show a summary of the end results of each previous PinDown run for that specific project.
- Bug log – a log that show bugs found from the current run.
- PinDown log – PinDown debug log.
- Run history – a list of previous PinDown runs.
- Recent commits – a list of recent commits.

The visual components where ranked in the following order, with number one being the most important:

1. Current graph, it was ranked the most important because it show the most recent status of the project.
2. Bug log and PinDown log, they were ranked the second most important items as they are tied to the current graph.
3. Result graph, it is also an important component for showing trends, but it is behind the current graph in terms of what needs to be emphasized.
4. Run history, this component does not need emphasis.
5. Recent commits, this component does not need emphasis.

When the emphasis hierarchy had been established we could start to try out different layouts according to the emphasis regions discussed in 5.1.3. Because of the ease in trying different layouts with paper mockups we could try a lot of different placements, while always trying to fit according to the emphasis regions and the internal ranking. Some of the layouts could be quickly discarded as they broke the hierarchy of what should be emphasized or not. In the following we will discuss three sequential layouts and our reasoning behind them, with the third layout being the one that progressed to a digital mockup.

5.4.3 Project view mockups

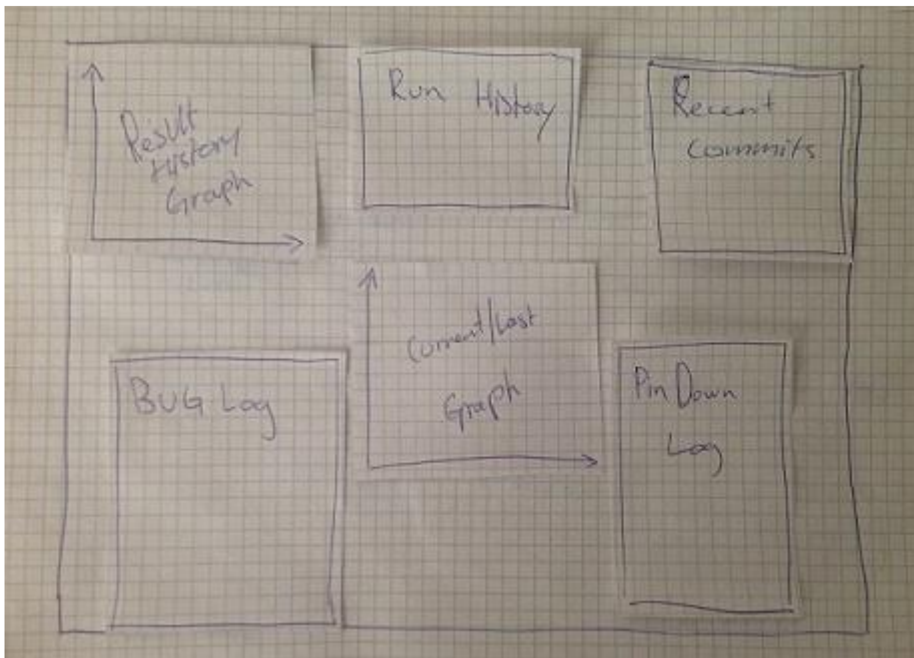

Figure 5.4: Paper mockup of layout 1.

Layout 1: The current graph was placed in the middle of the dashboard. The result graph was placed in the upper-left corner. Recent commits was placed in the upper-right corner and run history was placed between the result graph and recent commits. The logs were placed on each side of the current graph. See figure 5.4.

In layout 1 we placed the most important item, the current graph, in the center of the screen as this is an area which is emphasized. However, when placing an item in this region it is necessary to set it apart from what surrounds it by using for example white space (Few 2006, p. 114). This meant that when placing the other components, there would have to be some space between them and the center. We placed the logs on either side of the current graph, but because these logs have a strong connection to the current graph, we would have them placed closer to the current graph than the other components, utilizing the principle of proximity. In other words, the white space would originally be even around each side of the current graph, then by moving the logs closer towards the current graph we would reduce the whitespace between them thus making them appear as a unity, or group. With this layout it became clear that the lower half of the emphasis hierarchy also had a connection between them, as they all focused more on an overview rather than current events.

We discarded layout 1 mainly because of two crucial aspects. When placing the logs on either side of the current graph, their width would be too small to show the average log message without having to break it into new lines. And, because the upper-left corner also is an area which is emphasized, this meant that the result graph could possibly be distracting as its x-axis would enter the field of vision when scanning the y-axis of the current graph. While the logs were ranked higher than the result graph, we'd argue that placing the logs in areas of less emphasis than that of the result graph would work because they would gain some emphasis from being connected to the current graph through the principle of proximity. Still, having the graphs arranged diagonally of each other was a bigger issue. If the current graph would have been placed in the upper-left corner instead,

with the result graph placed in the center, it could have worked better with two diagonally placed graphs as this would aligned with the natural progression of going from left to right and from current events to a summary, but then the placement of the logs wouldn't make any sense.

We decided that a grid layout would be a better option as this would neutralize the center region. Also we were reminded of why we didn't allow for two items to hold the highest rank as it would be hard to optimize two areas of the dashboard with strong emphasis.

Figure 5.5: Paper mockup of layout 2.

Layout 2: The current graph was placed in the upper-left corner, the result graph was placed in the upper-right corner. Run history was placed between the two graphs. The two logs were placed in the lower-left corner, one on top of the other and drawn out as to fill up space and to allow log messages to be displayed without breaking the sentence into two lines. Recent commits was placed in the lower-right corner. See figure 5.5.

In layout 2 we would still have the current graph placed in a region appropriate to its rank of most important component, and with the result graph in the upper-right corner the layout became more sequential as the current graph pass on its end result to the result graph. We had learned from the previous layout that this progression from one graph to the other also meant that they had a connection to each other which would have to be considered.

This layout was also discarded. We didn't think that we could properly use the principle of proximity to connect the current graph with the logs as instead they were separated by the principle of enclosure. The principle of enclosure says that things can be grouped together by a border such as a line. While there was no visible line, the grid pattern itself organize the components into rows which are a form of natural line, which became apparent as the logs were drawn out. Also, because the logs were drawn out the upper log would enter the center region, which we had tried to stay away from. And since it would be the second half of the log that would be placed in the center region it would probably only be annoying as the logs wouldn't make any sense without reading the first part. Because we used paper

mockups it is hard to tell if this would have been a problem or not, but we argued that this would most likely be distracting in one way or another.

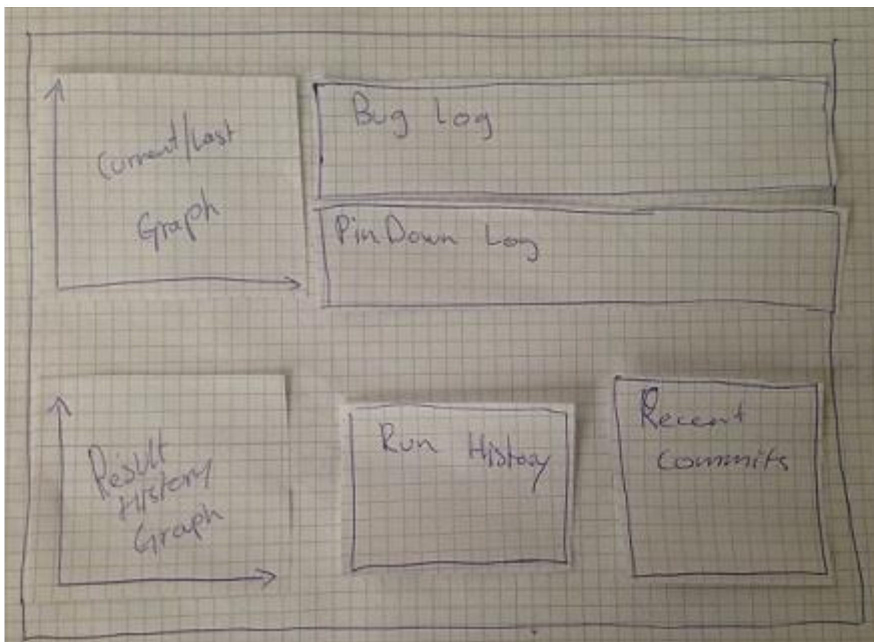

Figure 5.6: Paper mockup of layout 3.

Layout 3: The current graph was placed in the upper-left corner, the two logs were placed in the upper-right corner, one on top of the other. The result graph was placed in the lower-left corner. Recent commits was placed in the lower-right corner and run history was placed between the result graph and recent commits. See figure 5.6.

In layout 3 the grid pattern would work better because on the lower side of the screen there weren't any component that would get into the center region with enough whitespace to make it stand out. As for the upper side, the lower log was now placed with its first half in the center region, however since we could now make use of the principle of proximity again and because it is the first half of the log that is first entering the field of vision, this wouldn't be a problem. Also, we could use the principle of similarity to connect the graphs together, by using the same colors for example. The principle of enclosure also meant that the upper and lower halves were separated, which seemed ideal since they represented current events and an overview of previous runs respectively.

Once we had decided to place the components according to layout 3 we developed a digital mockup. See appendix B, figure B1. Now that we could see the components on the screen and interact with them in a more direct way, a few things became apparent. We wanted the run history component to work as navigation through previous runs, which meant that browsing through the run history would have to change the state of the current graph and logs, as these would be used to show these previous runs. However, because this new connection between the components was not supported by our design principles, it didn't feel intuitive. Another reason as to why it didn't feel intuitive can be found in *Usability Testing Essentials* (Barnum 2011, p. 86) where Barnum explains that for a website, users expect to find "site search on the upper-right corner or near the upper-left corner" and we found that this was true in our case as well. This meant that we would have to rearrange the components so as to not go against our goal of ease of use.

The initial idea of the logs was to have a button for each log which when pressed would expand the log so that it could be read without constantly having to scroll through it. This meant that if the logs weren't in their expanded state, they would be nearly useless. We decided to change the layout again before implementing a prototype.

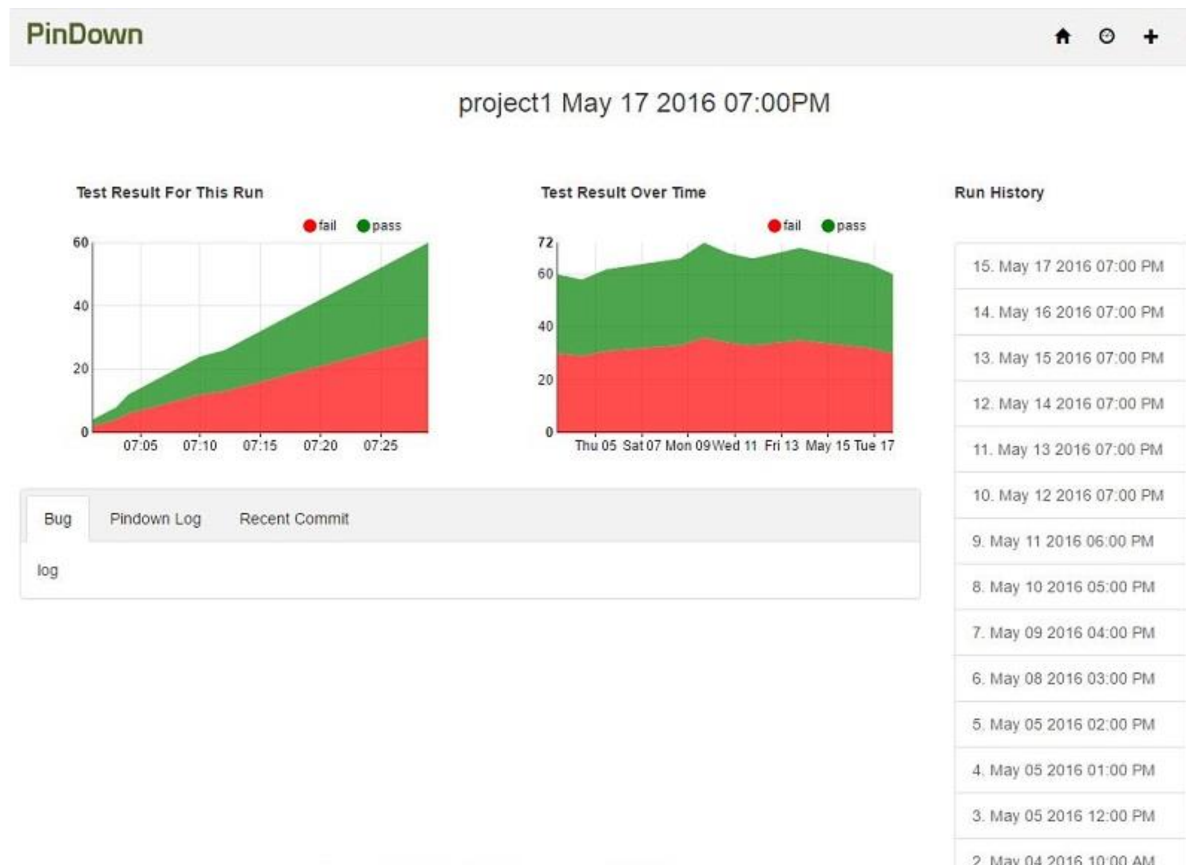

Figure 5.7: Layout 4.

Layout 4: The current graph was placed in the upper-left corner, the result graph was placed to the right of it. Run history was placed on the right side of the screen and the logs, together with recent commits were placed below the graphs, in a panel with a pane for each component. Because changing the components on the screen was a simple task, we didn't feel the need to revisit the paper mockups. See figure 5.7.

Because the logs were only useful if they were expanded, it was decided that they should always be expanded. Because of the size of the dashboard, this meant two things; the two logs could not be expanded at the same time, and when expanded the two graphs could not be both monitored easily at a glance. The decision was made to move the result graph to the upper-half of the dashboard, and move down the logs. Then it made sense to merge recent commits together with the logs and have them displayed as panes, and let the run history expand over both halves. We still made use of the principle of similarity to connect the two graphs.

5.5 Presenting the design

The placement of components is arguably the most important aspect of the design. It doesn't matter how well each component in itself is designed, if the layout doesn't make any sense or if the layout make it difficult to find information. That said, once the layout has been determined, the design of the component becomes so important that poorly designed components risk making a good layout inoperable.

In *Information Dashboard Design* (Few, 2006) Few expand on the principle of data-ink ratio by saying that for the entire dashboard the non-data pixels should be reduced to a minimum, within reason. Non-data pixels are pixels that doesn't display data, this does not include a blank background. In order to reduce non-data pixels Few present two steps.

"

1. *Eliminate all unnecessary non-data pixels.*
2. *De-emphasize and regularize the non-data pixels that remain.*

"(*Few 2006)*

The first step is simple and self-explanatory, however knowing what counts as unnecessary require some thinking and eventually it will lead to the second step of making sure that non-data pixels aren't getting in the way of data pixels. Not all non-data pixels can be eliminated without losing something valuable. In the graph example explaining the concept of data-ink ratio in 5.3.1, losing the axis would mean losing structure as the axis are fundamental to the way graphs are presented. What the second step imply is to keep the axis but de-emphasize them, making them visible enough to work as axis without interfering with the data pixels.

In our design there isn't much space of the dashboard not taken up by a component, making the first step rather easy as there weren't many unnecessary non-data pixels to begin with. As for the second step we found that for each component, except the graphs, there weren't many non-data pixels that didn't directly support structure and organization. The graphs are interesting because we decided to add non-data pixels to them instead of removing. We used stacked area charts as they are useful for showing trends over time between the attributes represented in the graph, in our case the relationship between passing and failing tests. A problem with using areas in graphs is that they can be hard to read accurately (Robbins 2005, p 61) and when the areas get close to each other in size it will become even harder to distinguish between them, but for showing trends, we'd argue that it will suffice. In order to more easily look up a value at a glance, we added grid lines to the background of the graph, as well as changing the opacity, making the areas somewhat transparent. Also the graphs are interactive, meaning that if you need to read a specific value you can do so accurately by moving the mouse over the graph.

Both graphs show trends in form of passing and failing tests over time, however, they use different time formats. By using the principle of similarity, using the same colors to present the areas, we let the user know at a glance that they show the same type of data. The colors, green for passing tests and red for failing, were chosen as they are considered standard within the industry. As mentioned earlier in 5.2 when discussing accessibility, for this thesis we didn't consider colors that would be easier to distinguish for people with color blindness.

6 Evaluation

To evaluate how well our design worked and to test our theories and design principles against the five factors of usability given by Lauesen, see 3.2, we wanted to set up a series of usability tests. Because Verifyter's user base is located in the USA, it is not feasible to run tests with users of the current system, however, we expected users to have a similar background as we do and we therefore based some of our findings on our own experience with the tool, and we set up a few tests that were completed by three students in computer science.

We led them through a total of six task-based scenarios and then asked them a few questions about the overall design and their thoughts of it. The reason we chose 3 test subjects was based on the article *Why you only need to test with 5 users* (Nielsen, 2000) in which Nielsen describe with a graph that the optimal number of users to test with is five, however with three users you are likely to find as much as 65% of the usability problems. Because of the small size of the test we argued that our three test subjects were likely to find all of the most crucial usability problems.

Before we created our scenarios that would test the new GUI we first established a few test goals based on Lauesen's definition of usability. These goals would then help us with writing the tasks, so that we knew what we wanted to test.

- Ease of learning: can a user without training complete the various tasks?
- Task efficiency: how long does it take to complete a task, is it within a reasonable timeframe?
- Subjective satisfaction: was the experience satisfying or enjoyable?
- Understandability: is it hard for the users to grasp the context of the tasks they are preforming?

We excluded one of the factors, ease of remembering, from our goals. The reason for this was because PinDown is a dedicated tool, and it is no likely to have occasional users. Even if there are occasional users, we would argue that they would be few enough for it to not be worthwhile testing. Ease of learning, task efficiency and understandability could be monitored as the test subject completed the tasks. The remaining factor, satisfaction, would require us to ask the test subject a few questions after they had completed the tasks. Task scenarios and follow-up questions can be found in Appendix C.

In this section we'll go over each of the scenarios, summarize the result of the scenario and discuss how this compared to our expected outcome. We'll refer to the test subjects as TS1, TS2 and TS3. In 6.3 we'll further discuss what we learnt from each scenario and how we made use of that knowledge.

6.1 Scenarios

Scenario 1: Exploring the overview and dashboard.
In the first scenario we asked the test subjects to explore the overview and dashboard. The starting point was the overview. They were free to click on things but they were not allowed to enter any data yet. The purpose of this scenario was for the test subject to get familiar with the interface and to see what they thought the different views were and what the information they could find was used for. We didn't have an expected outcome for this scenario as the questions that we asked were open for interpretation.

TS1 thought that the overview had something to do with different bugs, and that the dashboard looked like a test system. Neither TS2 nor TS3 visited the dashboard, and both of them thought the overview was used to show statistics for different tests. During this scenario we noticed that each test subject had difficulties with going from the overview to the dashboard.

Scenario 2: Creating a new project.
In this scenario the test subject was asked to create a new project and name it "project1". This project already existed and the purpose of this scenario was to try and see if the error message displayed was clear enough. We expected the test subject to realize that project1 already existed.

TS1 managed to complete the task, however the error message that displayed was ambiguous. TS2 and TS3 did not see any error messages at all. This was because when creating a new project there are two fields that you can enter text into, project name and project group, we only asked them to fill in the name of the project, but the error message only displayed if both fields contained text.

Scenario 3: Visiting the project dashboard.
The purpose of this scenario was to see if the test subject could move from the overview to the dashboard of a specific project. We asked them to find out more information about project1. We expected them to find the project in the overview and click on it to enter the dashboard.

All of the test subjects had difficulties with finding project1. It belonged to a project group and to find it they first had to expand the group to view projects. Once they found the project, TS1 had no problem with visiting the dashboard while TS2 and TS3 struggled at first. To visit the dashboard you need to click on the name but it wasn't made clear enough.

Scenario 4: Look up information in the graphs.
In this scenario we asked the test subjects to compare two results, the failing tests from the last time someone ran the tool, with the failing tests from the run previous to that. This scenario could be completed two ways, either by looking at the summary graph, which shows the result of all the previous runs, or by first looking up the value in the current graph, then navigate to the previous run in run history and then look up the value in the current graph which would show the result of that run.

None of the test subjects used the summary graph, we later learnt that it wasn't obvious to them that this was actually a summary. Furthermore only TS1 got the correct answer of number of fails, this was because TS2 and TS3 looked up the wrong dates in the run history. As the dates were displayed in ascending order, we assume that they expected the

last run to be at the top. Despite that TS2 and TS3 picked wrong dates, none of them had any trouble reading the graphs.

Scenario 5: Navigation.
In this scenario we asked the test subject to look up a specific date in the run history to see the status of the project that day. This date had three runs, run at different times. We were interested in seeing which run the test subject would pick. All tests subjects picked the last run of that day.

Scenario 6: The bottom panel.
In this scenario we asked the test subject to find the most recent commit. We expected them to browse to the right pane in the bottom panel to view the recent commits. All of the test subjects managed to find the most recent commit without any difficulties.

6.2 Follow-up questions

After they completed the scenarios we asked them a few follow-up questions, the purpose of these questions was to let them discuss the interface with their own words.

Question 1: What did you think of the overall design?
TS1 thought that the design was good except for some things being ambiguous, e.g. one of the icons looked like a network symbol and not a bar chart meant to indicate a shortcut to the dashboard. In the overview page it would be a good idea to let the user see names of icons when they place the mouse over them, and that the mouse change into a hand symbol when placed over a link to indicate that it can be pressed.

TS2 thought that the design was good but not very intuitive and would also like to know what things can be used to navigate through the application without having to click on it, i.e. to show names when placing mouse over icons.

TS3 thought the design was “stale” and would also like to not have to click on things to find out what they are or lead to. TS3 also confused the dashboard shortcut icon to be network settings.

Question 2: Was there any component that felt out of place? If yes, why?
TS1 thought that the button to expand projects felt out of place because of its colors. It would have been better if it was only a plus sign without any background color to it.

TS2 thought that the bug log in the bottom panel felt out of place because there wasn’t any connection between it and the graphs.

TS3 thought that the icon resembling a network symbol was confusing and therefore out of place.

Question 3: Was it hard to find the information?
TS1 thought it was difficult to find information.

TS2 thought it was difficult to track information.

TS3 did not think it was difficult to find information.

Question 4: Was it hard to interpret the information?
TS1 thought it was somewhat difficult to interpret the information.

TS2 did not think it was difficult to interpret the information.

TS3 thought it was somewhat difficult to interpret the information.

Question 5: Did you understand the connection between the two graphs?
TS1 thought that the connection between the graphs could be made more obvious.

TS2 understood the connection between the graphs but thought that it wasn't obvious at first.

TS3 could also understand the connection but thought that it could have been emphasized more.

Question 6: Did you miss any information?
TS1 would like the graphs to be more clear e.g. by adding axis labels and adding date to the title above the graph.

TS2 would like to see a connection made between the graphs and bottom panel, perhaps by adding date above the panel.

TS3 would like the start and stop button to be made more clear.

6.3 Improving the design

It is worth noting that in a real scenario the user will most likely complete labs first and thus get some formal training in how to operate the interface. Even so it is important that the interface is intuitive to use to make learning a faster experience. We think that the test gave us a lot of valuable input. First we'll address the things that we would like to improve after the tests and after that we'll discuss some usability issues found and why they're not necessarily going to be issues for the real user.

Mouse-over tooltips
We think that a tooltip on mouse-over to display icon names is a very good idea for the initial learning phase, to save users from having to click around too much to figure out where an icon lead them.

Error messages
The error messages were ambiguous because the message didn't specify exactly what the error was, and it was also displayed in a dark green color. TS1 pointed this out and added that perhaps a red color would be better suited, to which we agree.

Expanding project groups
The button that expand the project groups into a tree structure was hard to find. The reason for this was that it had a background color that made it hard to see that there was a plus sign. TS1 thought that a plus sign would be enough to indicate that this button would expand the project group.

Run history
It makes more sense to have the most recent run showed at the top of the list rather than at the bottom, something that all of the test subjects pointed out and something that we agree with.

Clarify connections
There was a few things that could be better explained within the dashboard e.g. the connection between the graphs and date above graphs and logs.

Things that won't be an issue
TS1 suggested that we switch places on the graphs to make it more obvious that navigation in run history changes the data showed in the current graph. We don't think that this is necessary because the navigation will change the state of the whole dashboard, i.e. both the current graph and logs will change to that run. Since navigation is something the user is likely to do after a run has been completed, we think that our reasoning for placing the graphs in order, current to summary, is still valid.

When creating a new project the user has to browse the menu in the upper-right corner. TS1 thought that it could be a good idea to have a shortcut to creating a new project visible in the overview. We don't think this will be necessary since the creation of new projects happens infrequently.

The confusing icons will also not likely be an issue because they are placeholders for Verifyter's custom icons.

7 Conclusion

In the closing chapter we'll present each of our questions and the result of answering them. Then we'll provide some ideas for future work.

7.1 Questions

Our conclusion is that the design met Verifyter's needs and that the dashboard should improve the usability of PinDown.

7.1.1 What are the pros and cons with the current implementation in terms of usability?

The most obvious drawback with the current implementation is that it is not a dashboard. This means that there will be a lot of switching views to achieve some tasks, and this lack of unity means that tasks take longer to complete and in some sense they are also more difficult than they have to be, to complete. With a dashboard you collect all the necessary data in one area.

When evaluating the current implementation against five factors of usability it is important to remember that these factors are balanced off each other. That means that while you should consider all of them, sometimes you have to make compromises. A dashboard while simple to use, might take more time to learn than the current implementation that has its content divided over different panes. But once you've learnt the basics then you will want to complete tasks at a faster phase. The dashboard is more equipped to allow users to complete tasks faster, at the expense of needing a bit more training.

We found that current implementation is easy to learn, easy to remember and easy to understand. This comes with the cost of being less task efficient and as consequence of this also less subjectively satisfying. In our design of a new GUI we wanted to improve task efficiency and subjective satisfying while maintain ease of learning, ease of remembering and understandability. A way we did this was by trying to make the dashboard as intuitive as possible.

7.1.2 What web frameworks are of interest, and how are they different from each?

The frameworks that we researched in this thesis were Ruby on Rails, Django and Play framework. We found that either of them could be used in our case. Both Ruby on Rails and Django can be run on JVM with JRuby and Jython respectively. JRuby and Jython are Java implementations of the two languages Ruby and Python. Our decision to use Play as our web framework was based on our knowledge of the Java language, but also because the current implementation of PinDown is Java based, which would mean an easier transition.

The main difference between the frameworks really is the language used. They all follow the same pattern of an MVC architecture, however it is slightly different in Django.

7.1.3 What needs does Verifyter have when it comes to design and functionality of the new GUI?

To answer this question we elicited requirements from Verifyter and worked with goals of the dashboard. The goals were focused on responsiveness, accessibility and ease of use. These goals concern both the design and the functionality. This was part of the first stage in the development process we followed, suggested in *Dashboard development guide* (Staron, 2015). The next step is to find out which technology to use when realizing the design. In our case this had already been done when answering the previous question, but we still used the proposed *dashboard selection model* (Staron et al. 2015), however we used it as a way to validate our choice.

The previous two questions progressed into this one and as a result we concluded that Verifyter's needs of the dashboard were:

- Backward compatibility.
- A design that is responsive, accessible and easy to learn and use.
- The functionality of the dashboard should improve task efficiency by removing some of the usability issues in the current design.

7.1.4 How should the project dashboard be designed?

This question had two sub-questions to it: What are Verifyter's needs? And, how will user experience be improved? By answering the previous three questions we hoped to gain knowledge to design the dashboard so that it met the requirements set by Verifyter and improved the user's experience. What we mean with improving user's experience is an assumption that with improved usability, the general experience will be improved. To answer the question we worked with two different types of mockups, paper and digital, and then we conducted a set of user tests to evaluate the design.

The user tests aimed towards usability and we learnt a lot from them. However, we think that most of the usability problems found by our test subjects either had to do with bad code, or quality of life changes. What we mean with quality of life changes in this context is that the problems found didn't force us to rework the design, but rather improve on it. There is difference there which we think is worth pointing out. A rework would e.g. mean to move components to new locations, while a quality of life change means to add or remove something from existing components. The latter often being easier to fix. An example of bad code was the error messages which only displayed when an error occurred and both of the text fields in "Create project" were filled in.

7.2 Future work

Since the implementation of Pindown followed the MVC design pattern it facilitates further development of the final product for future project. Further development can be implementation of remaining functions and more thorough usability tests before the final product is released. Due to the timeframe we were unable to make a second usability test to verify that the design changes improved usability.

References

Barnum, C. (2011). *Usability Testing Essentials.* Morgan Kaufmann, Boston.

Bostock, M., Ogievetsky, V. Heer, J. *Data-Driven Documents.* IEEE Transactions On Visualization And Computer Graphics [IEEE Trans Vis Comput Graph] 2011 Dec; Vol. 17 (12), pp. 2301-9.

Brikman, Y. (2014) *The Ultimate Guide to Getting Started with the Play Framework.* Retrieved 2 March, 2016
URL: http://www.ybrikman.com/writing/2014/03/10/the-ultimate-guide-to-getting-started/

Few, S. (2006). *Information Dashboard Design - The effective visual communication of data*. O'Reilly, Sebastopol, CA.

Flesch, B. (2014). *Design, Development and Evaluation of a Big Data Analytics Dashboard.* Copenhagen Business School and University of Mannheim, Department of IT of Management. URL:
http://studenttheses.cbs.dk/bitstream/handle/10417/4945/benjamin_flesch.pdf?sequence=1

Holovaty, A., Kaplan-Moss, J. (2009) *The Django Book*. Apress.
URL: http://www.djangobook.com/en/2.0/index.html

Krug, S. (2006). *Don't make me think! – A common sense approach to web usability*. New Riders, Berkeley, Calif.

Lauesen, S. (2002). *Software Requirements – Styles and Techniques.* Addison-Wesley, Harlow.

Nielsen, J. (2000). *Why you only need to test with 5 users*. Retrieved 9 May, 2016
URL: https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/

Nunes, D. (2014) *Rails vs Django vs Play.* Retrieved 29 February, 2016
URL: http://www.diogonunes.com/blog/rails-vs-django-vs-play-frameworks/

Robbins, N.B. (2005). *Creating more effective graphs*. Wiley-Interscience, Hoboken, N.J.

Staron, M. (2015). *Dashboard Development Guide – How to build sustainable and useful dashboards to support software development and maintenance.* Research Reports in Software Engineering and Management 2015:02 ISSN 1654-4870 Chalmers University of technology and University of Gothenburg, Department of Computer Science and Engineering.

Staron, M., Niesel, K., Meding, W. (2015). *Selecting the Right Visualization of Indicators and Measures – Dashboard Selection Model.* University of Gothenburg, IT Faculty, Department of Computer Science and Engineering. *International Conference on Software Measurement (Mensura)*. 01/01/2015

Todorovic, D. (2008) *Gestalt principles.* Retrieved 26 April, 2016
URL: http://www.scholarpedia.org/article/Gestalt_principles

Tufte, E.R. (2001). *The visual display of quantitative information*. Graphics Press, Cheshire, Conn.

Ware, C. (2000). *Information Visualization – Perception for design*. Morgan Kaufman, San Francisco.

Links

Django: https://www.djangoproject.com/

Jython: http://www.jython.org/

Ruby on Rails: http://rubyonrails.org/

JRuby: http://jruby.org/

Play Framework: https://www.playframework.com/

D3.js: https://d3js.org/

Google charts: https://developers.google.com/chart/

VanCharts: http://www.vancharts.com/

NVD3: http://nvd3.org/

Bootstrap: http://getbootstrap.com/

JQuery: https://jquery.com/

Appendix

A. PinDown

PinDown

Test Results Open Bugs Settings

lab5

Stop Start - with Schedule

run1 : 11 May 2016 14:15 CEST : Current

Project : lab5 have not been run.
Starting up pindown job
Wed May 11 14:15:26 CEST 2016
Started by Anonymous
Starting project: lab5
workfolder: /opt/pindown/lab5
Running as verifytererik
In folder /opt/pindown/lab5 owned by verifytererik
Pindown starting...
PinDown creating run area for the test phase
Using folder: /opt/pindown/lab5/runarea/test/run1
Starting shellcommand: ../../../cleantest.sh > runtest.log 2>&1
Preparing to check out these repositories at the latest revisions:
svn://localhost/training/y80e/branches/demo at 132 (latest) Earliest is 132
PinDown checking out revision for test, using a monolithic repository
Using folder: /opt/pindown/lab5/checkoutarea/test
Progress 0% - May 11 2016 14:15:30
Progress 100% - May 11 2016 14:15:32
SVN checkOutRevision: Checking out 132 from svn://localhost/training/y80e/branches/demo into /opt/pindown/lab5/checkoutarea/test/pilot
CheckOutRepositoriesCommand: checkout done.
Starting shellcommand: ln -s ../../../checkoutarea/test/pilot
Starting shellcommand: ../../../runtest.sh >> runtest.log 2>&1&
0 tests (0 pass, 0 fail -> 0 new, 0 closed). 0 builds (0 pass, 0 fail -> 0 new, 0 closed). Run 0/15 min (May 11 2:15 PM).
0 tests (0 pass, 0 fail -> 0 new, 0 closed). 1 builds (1 pass, 0 fail -> 0 new, 0 closed). Run 0/15 min (May 11 2:15 PM).
Completed: Test phase has completed. File /opt/pindown/lab5/runarea/test/run1/done.nfo which PinDown was waiting for have appeared.
Summary of job completion: All 1 job's PinDown was waiting for have completed. Waited 0 min. Timeout in 15 min. (May 11 2:15 PM)
1 tests (1 pass, 0 fail -> 0 new, 0 closed). 1 builds (1 pass, 0 fail -> 0 new, 0 closed). Run 0/15 min (May 11 2:15 PM).
Diagnosing...
Good news - all tests passed! There are no failures for PinDown to diagnose.
Preparing email with subject "lab5 regression: Success! No failing tests, no bugs.", to be sent
to recepients: verifytererik@training.verifyter.com
Email sent May 11 2016 14:15
PinDown diagnosis has finished.

Pindown log Test log

Powered by WaveMaker

Copyright Verifyter 2010-2016

Figure A1: Test pane of the current GUI.

PinDown

Test Results Open Bugs Settings

lab5

Hide always passing Hide seeds

Refresh

Show 50 entries

Search:

Config	Test	May 11 2016 14:15 2 tests bug
build_y80e	::Build Result::	
build_y80e	alu_ops	

Showing 1 to 2 of 2 entries

Previous Next

Powered by WaveMaker

Copyright Verifyter 2010-2016

Figure A2: Result pane of the current GUI.

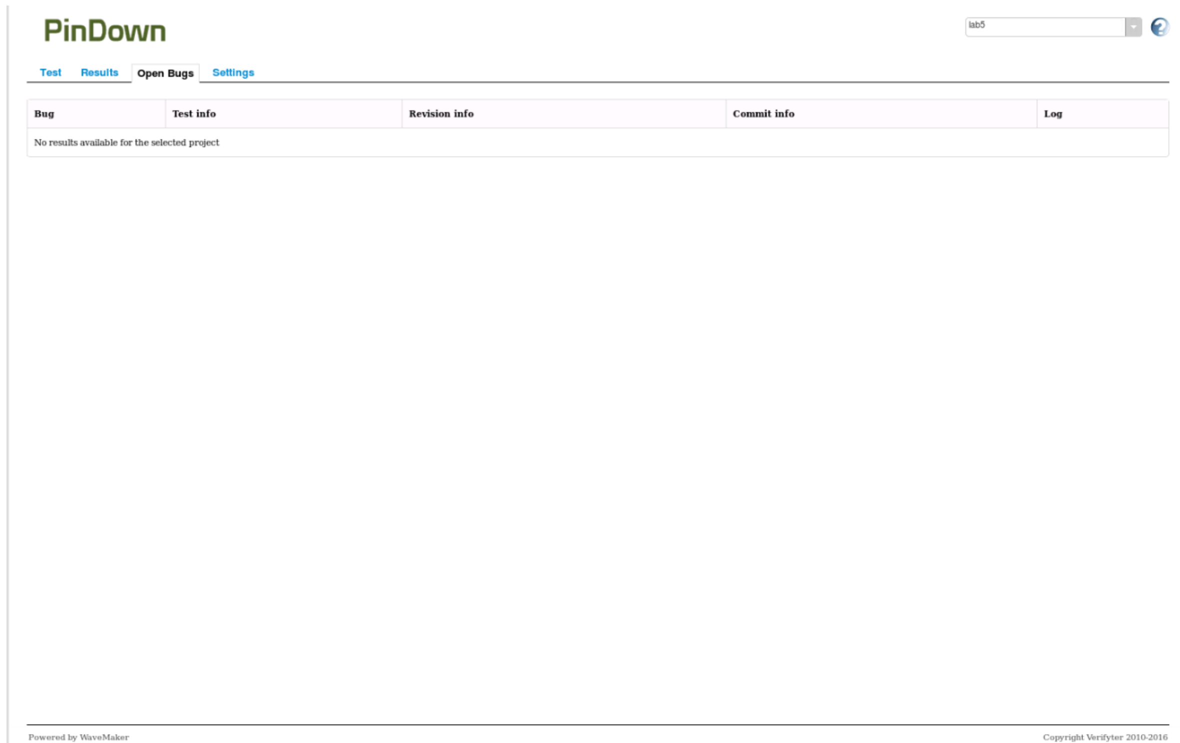

Figure A3: Open Bugs pane of the current GUI.

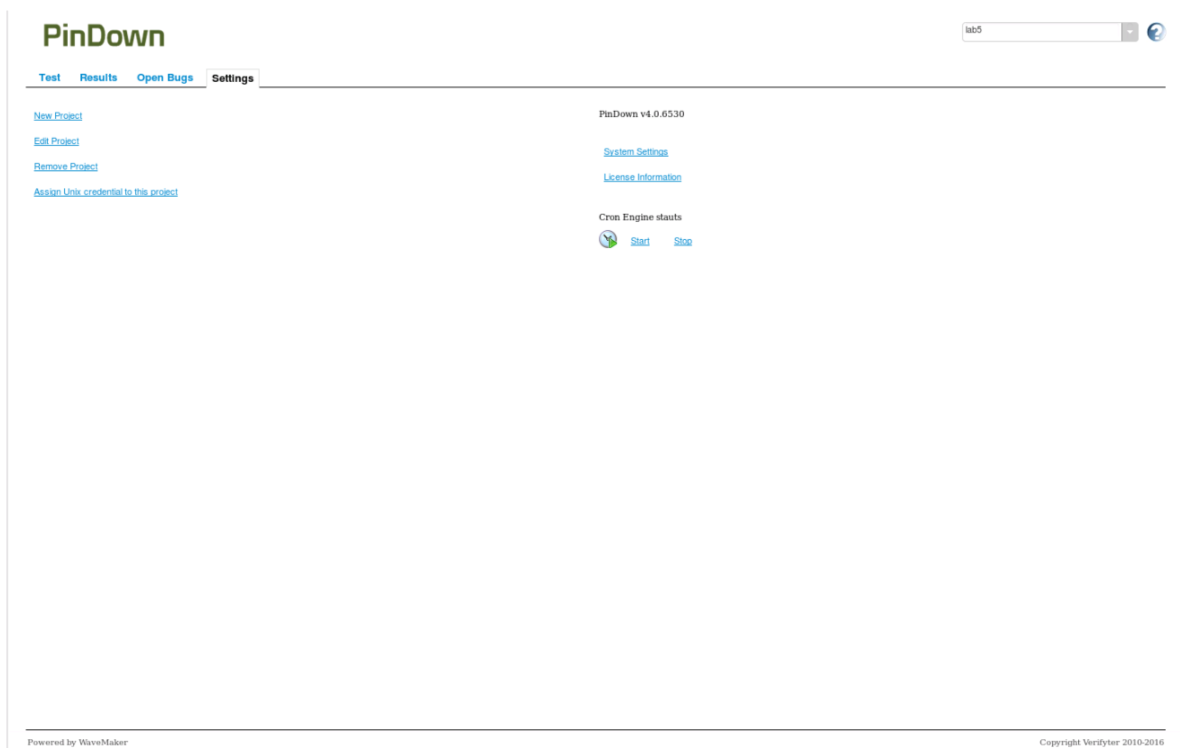

Figure A4: Settings pane of the current GUI.

B. Mockup

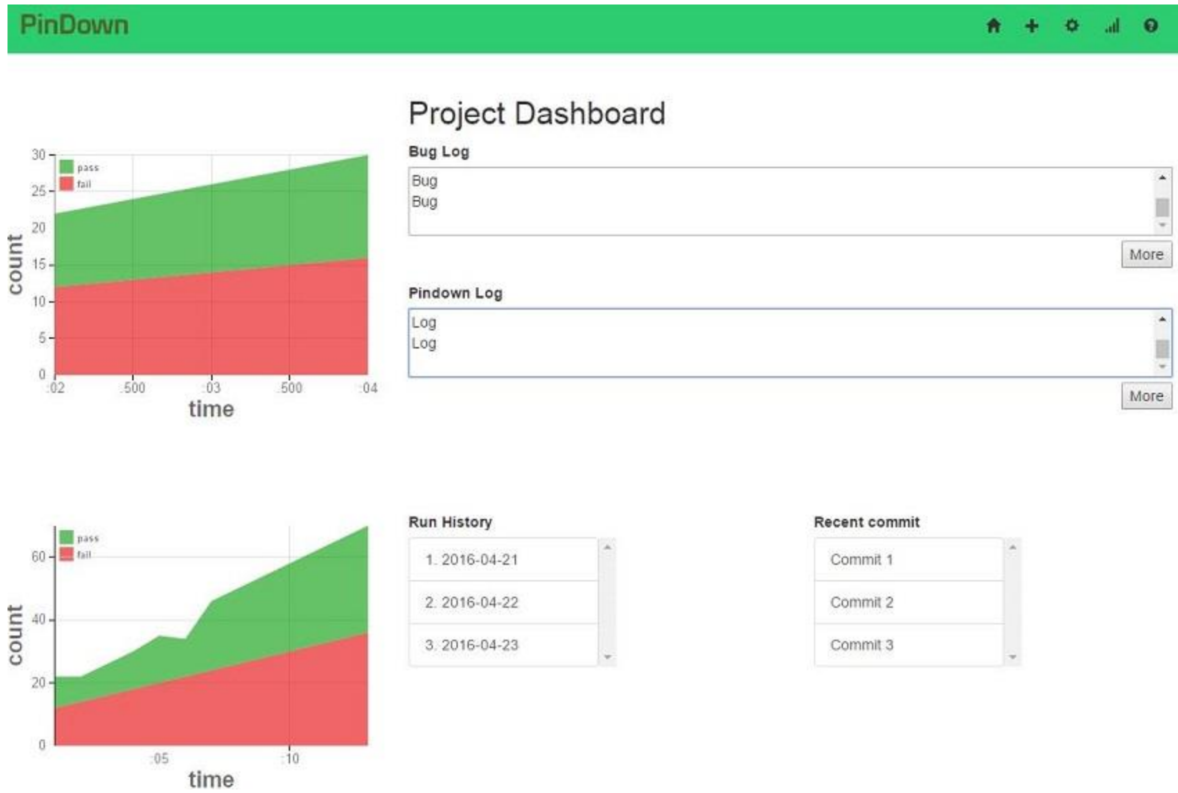

Figure B1 Digital mockup of layout 3.

C. User tests

Scenario 1: Exploring the overview and dashboard.
You are free to click around and explore the GUI, but don't enter any data yet. (The test subject is presented with the overview)

- Can you tell us what you think that this view is? (The overview)
- Can you tell us what you think that this view is? (If the test subject explore the dashboard)
- What do you think this information is used for?

Scenario 2: Creating a new project.
You're working as a microchip designer in Silicon Valley. It is Monday morning and you would like to create a new project. You want to name it project1.

Expected outcome: Because project1 already exist the test subject should realize this when notified by the system.

Scenario 3: Visiting the project dashboard.
You found out that the project already existed. After creating a project with a different name you would like know more about project1.

Expected outcome: After creating a new project with a different name, the test subject should visit the dashboard of project1.

Scenario 4: Look up information in the graphs.
You would like to compare the number of failed tests from the last time someone ran the tool, with the run prior to that.

- Can you tell us how many tests failed the last time?
- Can you tell us how many tests failed prior to that?

Expected outcome: There are two ways to answer this question. Either the test subject can look in the summary graph for a quick analysis. Or the test subject can first look at the current graph, then navigate to the previous run through run history.

Scenario 5: Navigation.
Today is the ninth of May. You called in sick last Thursday and would like to know the status of the project at that time. (The fifth of May)

Expected outcome: The test subject should find the correct date in run history and click it.

Scenario 6: The bottom panel.
You are interested in knowing what commits where made recently.

Expected outcome: The test subject should browse to the right pane.

Follow-up questions:

- What did you think of the overall design?
- Was there any component that felt out of place? If yes, why?
- Was it hard to find the information?
- Was it hard to interpret the information?
- Did you understand the connection between the two graphs?
- Did you miss any information?