Architecture Design

Daan van der Valk Ahmet Gudek Marco Boom
Leendert van Doorn Borek Beker

May 17, 2016

1 Introduction

In this document the result of the Context Project with the subject tools for software engineering will be described. First, a description and explaination of the goals of the project is described. In the next chapter, a description of the chosen tools and system architecture is given.

1.1 Design goals

In this subsection the design goals of the product are described. These are goals that summarize the aim of the project.

- **Give meaning to data**: The data that is collected from Bitbucket and Github is mainly raw data. A goal of this project is to filter and aggregate the data. The reason for this is that not all the data might be very useful. Also with the use of aggregation, data that is meaningless on it's own, will get meaning when aggregated together.
- **Usability** (User Experience): A goal that is aimed to be achieved in this project is the goal Usability. The reason why this is a goal, is that the users need to have a good experience without too much hassle when using the tool. The user need to quickly analyze the presented data. Moreover, a user needs to find his way within the app. It should be clear for a user how to use the features of the app.
- **Maintainable**: This project is part of a study course. After the end of this course this tool might be widely used among software engineers. To keep providing a great working tool the tools needs to be updated constantly. Because of this the code needs to be maintainable by future software engineers that will work on this project.
- **Scaleable**: As described before, if a lot of software engineers start using the tool, it needs to be resilient to a rapid growing user amount. If the software breaks due to high traffic, it will also hurt the usability of the tool.

2 Software architecture

To realize the goals that are described in the last chapter, the best tools and architecture need to be used.

2.1 Basic system decomposition

Octopeer basically consists of three clients. Two clients collect data from BitBucket and GitHub. These clients (Google Chrome Extensions) will collect data like mouse movement and actions of a software engineer. Data that is collected will be sent to a database that runs on the octopeer domain.

The visualization tool only connects to the database. The data that is provided by that database will be used by the tool and it will aggregated and visualized locally.

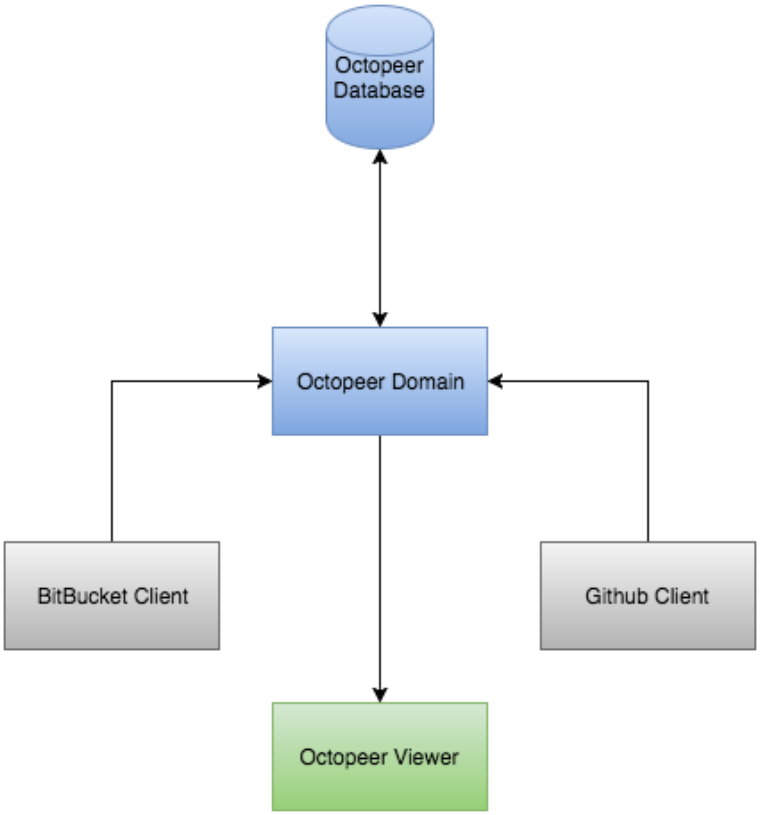

Figure 1: Basic system decomposition

2.2 Octopeer Analytics decomposition

For the client itself a basic structure will be described. The calls to the Octopeer Domain, Github API and the BitBucket API are made by the "APICallers". These callers are called by objects called "Services. These services consists of concrete definitions on what the type of objects you want to get. For example, with the GitHubService, you can call a function "getPullRequest". This functions needs a callback as parameter, and this callback will be called when the APICaller got the pullrequest from the GitHub API. The other services work the same. The front-end of Octopeer Analytics will use these services to get the data from the API's that are being used.

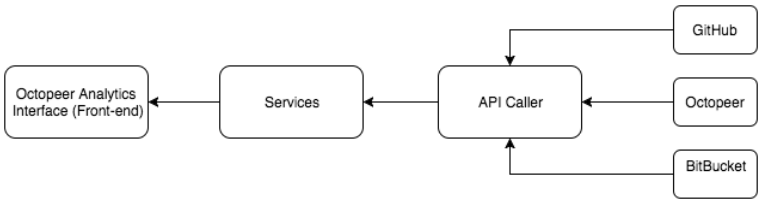

Figure 2: Ocotpeer viewer decomposition

2.3 Subsystems

The following tools and systems are used to realize the visualization tool.

2.3.1 Programming languages

The visualization tool is mainly written in the usual web development languages (HTML, Javascript and CSS). To make writing Javascript code a bit easier, a framework called jQuery is used. Also to make a large amount of different kind of graphs the framework D3 is used. This framework is an extension of Javascript and is used to create great looking and useable graphs.

2.3.2 Project management

Github is used to maintain the code repositories. By the use of pull request the tool will be improved constantly.

Waffle.io is used to manage and keep track of tasks. It gives good insights in project development and progress.

2.3.3 Continuous Integration

To make sure that every pull request that get merged won't break the tool and that a good quality is maintained, continuous integration is mandatory. A good tool that is able to handle this is GruntJS in combination with Travis. A pull request can not be merged until all the tests are passed.

2.4 Persistent data management

The database is on the external Octopeer domain. This domain handles the data with the use of Django (a Python ORM language). Because of the fact that this is another project good communication with the engineers of the project is required.

2.5 Concurrency

A goal of this project is scalability. This means that when the user base grows rapidly big, the tool should not break down. A lot of concurrent users should not be hold back. The external Octopeer domain should be able to handle a lot of traffic.

2.6 Modularity

Modularity is the concept of separating as many parts of the program as possible. This allows small changes to the code to be executed easily, as the code of the different parts are in separate locations. For the modularity of this project, RequireJS is used. This library loads in the modules of visualisations and integrates them into the main html file.

To be able to complete this process as easily and efficient as possible, a specific format has been designed for the visualisation modules. The format requires a pre-set set of fields to be defined in a JavaScript file which are read, executed and appended to the DOM by a main script. The full documentation for this function can be found in Module_documentation.pdf.

3 Glossary

1. **UX**: User Experience
2. **ORM**: Object Relational Mapping language. This is a language that maps a database to a more usable language. Django is an example of this.
3. **CI**: Continuous Integration automate tests and builds. Travis is an example of this.