

# Bachelor Project Report

## Visualizing and Monitoring Software Ecosystems in a Mobile App

Andrea G. Perozziello

Advisor: Prof. Gabriele Bavota   Co-Advisor: Dr. Csaba Nagy

### **Abstract**

Code sharing platforms such as GitHub provide the possibility to reuse existing code, and to learn from others coding activities.

While the amount of code available on GitHub is overwhelming, it is not always easy for developers to assess whether a given project implements high quality code.

High quality code is known to be easier to read, evolve and maintain. Having the possibility to easily monitor and compare the internal quality of software systems can help developers to identify high-quality code from which they can learn coding patterns and reuse code.

In this thesis we present a mobile application which allows developers to monitor and compare the quality of projects belonging to a software ecosystem of interest.

# 1 Introduction

Open Source Software (OSS) is becoming more and more popular. Software developers can access OSS thanks to code sharing platforms such as GitHub [1]. Thanks to these sharing platforms, the developers can easily access OSS in order to learn from it and reuse code.

Selecting the code components to reuse is not an easy task. Indeed, OSS projects can exhibit a substantial difference in the quality of their code [2]. Low code quality can be exhibited in many different ways: Code components can lack clarity, maintainability or may be in need for refactoring to improve their efficiency, testability, extensibility.

To make it easy for developers to monitor and compare the quality of OSS projects, we implemented an iOS app to visualize information about the internal code quality of several systems, allowing for a simple comparison. Based on this information, the developer can select the system that is best suited for her needs.

In particular, our app offers the following features:

- The user can create a software ecosystems (i.e., a collection of systems) of interest.
- The user can add to the ecosystem a number of GitHub projects she wants to compare.
- Through visualizations, the user can compare the projects in the ecosystem from different aspects (e.g., quality metrics, open issues).

## 1.1 Structure of the Thesis

In Chapter 2 we present the architecture of the software we developed. Each component is detailed in a specific subsection.

Chapter 3 gives an overview of the iOS mobile application, and a brief explanation of all the different views that compose it.

Finally, Chapter 4 concludes the thesis and highlights directions for future work.

## 2 Architecture

Our app features three main components. First, the GitHub Miner, that is in charge of retrieving data from the GitHub APIs.

Second, a module in charge of computing quality metrics for a given repository. These metrics capture the internal quality of code components and are detailed in Section 2.2.

Finally, the information retrieved by the two above-mentioned components, are stored in a MySQL database that can be queried by the mobile app through a Web service.

Figure 1 shows the described architecture.

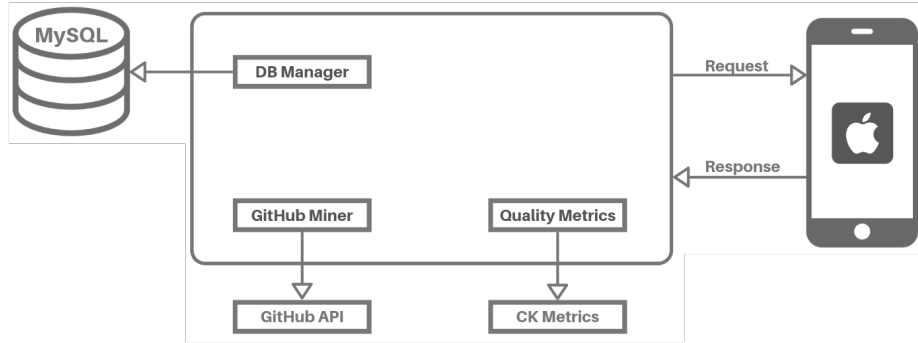


Figure 1: Content Diagram of the application

## 2.1 GitHub Miner

In order to retrieve the needed data from GitHub we developed a module using the GitHub APIs. This module is implemented as a Web application using “Apache Tomcat” [3] as container.

Through the GitHub APIs we retrieve data in JSON format regarding Repositories, Owners, Commits, Pull Requests, Authors, Issues, Files and Stars by GET HTTP Requests.

This information is then stored in a MySQL database depicted in Figure 2.

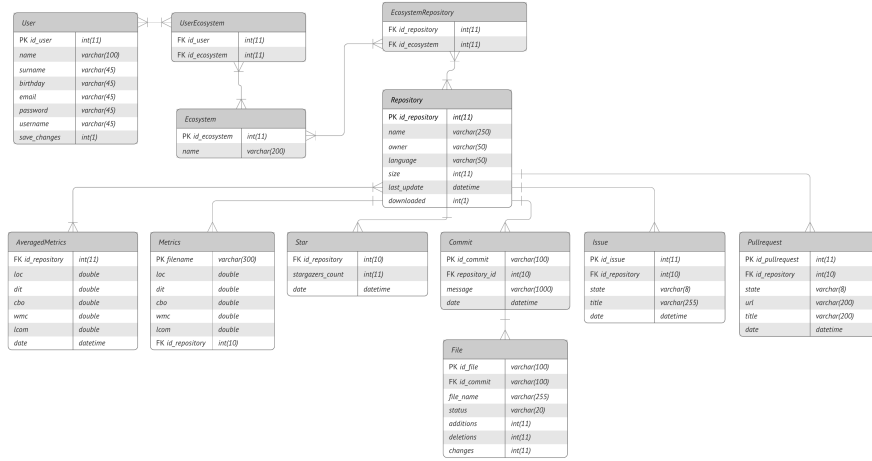


Figure 2: Database Schema

## 2.2 Computing Quality metrics

The module in charge of computing quality metrics is involved each time a new repository is analyzed. We only consider Java repositories due to the fact that the static analyzer used to compute the metrics does only support Java.

When a repository is added to a certain Ecosystem, the repository is firstly cloned inside local storage. Successively, it is fed to the metrics calculator component which is based on a Java library written by Mauricio Aniche [4]. Given a path, this library computes the metrics of all the Java files contained in it using the Java Developers Tools API from Eclipse.

The metrics computed by the tool are the ones belonging to the Chidamber & Kemerer [5] object-oriented metrics suite, described in the following.

### **2.2.1 CBO Coupling between Object Classes**

Number of classes to which a class is coupled.

Two classes are said to be coupled if methods within a class use methods belonging to another one.

Having a high CBO can in fact lead to important difficulties when reusing code implemented and it is a harm to modular design.

### **2.2.2 WMC Weighted Methods Per Class**

A proxy for the complexity of the methods implemented in a class.

The higher the WMC of a class, the more that class will be prone to faults. WMC is none at all a good predictor for how much time and effort are related with the development and the maintenance of the class.

### **2.2.3 DIT Depth of Inheritance Tree**

Maximum inheritance path from the class to the root class.

DIT underlines how deep a class is in the hierarchy from the root class. It is a direct indicator for the complexity of the class contents (methods, variables). Visual Studio .NET documentation states that classes with a DIT value  $> 5$  are indeed said to be too complex to develop.

### **2.2.4 LCOM2 Lack of Cohesion of Methods**

Cohesion metrics measure how well the methods of a class are related to each other.

A class is said to be cohesive if it performs one function. A class which performs two or more unrelated functions is said to be non-cohesive; it usually should be split into smaller classes.

Low cohesion underlines poor design and high complexity.

Although high cohesion is considered to be desirable cause contributes to encapsulation, one drawback is the high coupling between the methods of the class, which brings to as high efforts in testing.

## 2.3 Web Service

HTTP responses and requests are managed inside the web service we developed by using servlets, which act as interface with the mobile application.

Servlets have been implemented to provide the data needed for the visualizations supported in the app (i.e., repository name, #commit, etc.).

To accomplish that, three types of HTTP methods have been used inside the Web application: GET, POST and DELETE.

Quality metrics are computed for every file of a given repository. The results of each metrics is then averaged and stored inside the database with a reference for the specific repository.

In order to visualize the data on the application charts, a request is made for every specified repository by using a servlet. This servlet retrieves the averaged value of each metric.

Averaged metrics for all requested repositories are then visualized in the app by a graph chart representation.

### 3 The iOS App

The iOS application is written in Swift and developed in Xcode.

The implementation mostly consisted in connecting different “Views” which communicate among them, and underneath interact with the *Web service* we developed.

The “*Views*” types are basically two:

- Blank static views with elements (i.e., buttons, text-field, images etc.) fixed inside it, named *ViewControllers*.
- Dynamic ones with same type of elements loaded differently each time, named *TableViewController*s. An exception is made for the settings view whose contents are static.

In the following, we present and describe the different views of the app.

#### 3.1 Login

This is the root view, in other words the first view displayed to the user when it opens the application, from here the user can either log in or register.

In order to correctly login, valid username and password must be typed in. The view is shown in Figure 3.

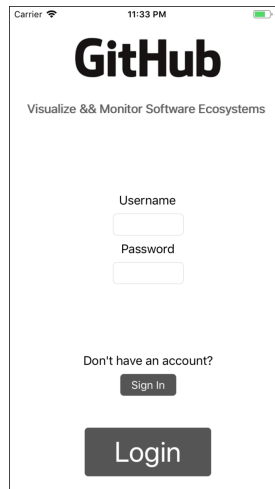


Figure 3: User Login

### 3.2 Register

The register view is visualized in the case the user wants to create a new account.

If all the text fields are filled in with the correct format, the user creates a new account and is redirected to the application home page shown in Figure 4.

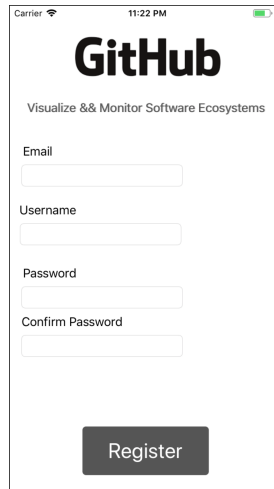
A screenshot of a mobile application interface for user registration. At the top, the status bar shows 'Carrier', signal strength, '11:22 PM', and battery level. The app header features the 'GitHub' logo in bold black text, followed by the tagline 'Visualize && Monitor Software Ecosystems' in a smaller font. Below the header, there are four text input fields labeled 'Email', 'Username', 'Password', and 'Confirm Password' from top to bottom. Each label is positioned to the left of its corresponding input field. At the bottom of the form is a dark gray button with the word 'Register' in white text.

Figure 4: User Registration

### 3.3 Home Ecosystem List View

This is the home page of the application. When entering this view all user ecosystems (if any) and their related data are immediately visualized.

From here the user can navigate through the other two main views (“Create Ecosystem”, “Settings”) of the application by interacting with the tab bar placed at the bottom of the page (see Figure 5).

### 3.4 Create Ecosystem

This is one of the three views contained inside the tab bar at the bottom of the page. Its purpose is to let the user creating its own ecosystem, with a unique name of his choice, and to create a list of repositories to load in that specific ecosystem. When the button “Create ecosystem” is pressed the ecosystem and its list of repositories are added to the database.



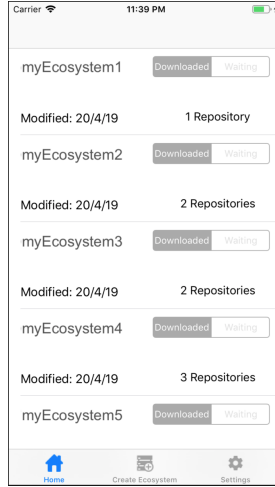


Figure 5: Home with User Ecosystems list

Then, the downloading of all the selected repositories starts, thus involving the GitHub Miner component (see Figure 6).

### 3.5 Ecosystem's Repository List View

This view shows all the repositories contained in a specific ecosystem. From here the user can see details about every single repository, and has access to a set of buttons fixed at the bottom of the page. Each of these buttons will bring the user to another view containing a graph visualizing specific types of information.

On the top left of the page it is present a plus button which allows the user to add a new repository. In this case, the download of the repository starts immediately (see Figure 7).

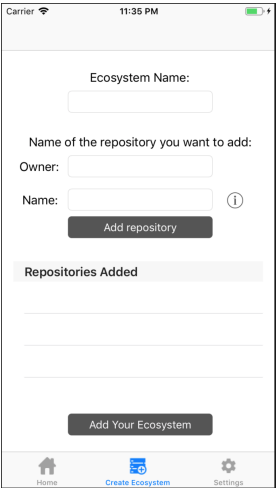


Figure 6: Ecosystem creation

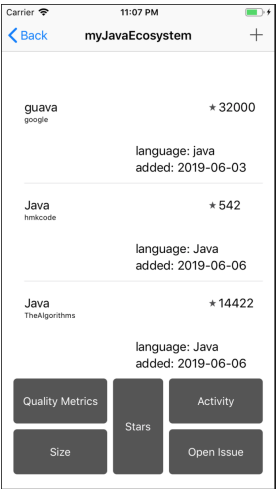


Figure 7: Repositories list for an Ecosystem

### 3.6 Ecosystem's Repository Stats View

The main objective of these five views is to allow the user to visualize in a quick way information about all the repositories contained in a specific ecosystem.

The only difference in these views is the data they respectively visualize.

In particular, we provide charts for: quality metrics, size (loc: lines of code), number of stars, number of open issues, and activity (number of: closed issue, pull requests and commits).

Since the main goal of this app is to allow the comparison of the code quality for different projects in the ecosystem, we show in Figure 8 the view visualizing the code quality graph.

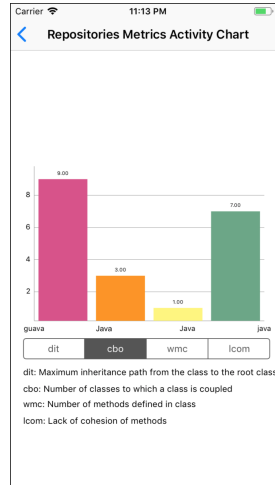


Figure 8: Quality Metrics Graph Chart for an Ecosystem

## 4 Conclusion & Future Work

We presented an iOS app to compare software projects belonging to a given ecosystem of interest from different perspectives.

The difficulties we faced during the development of the thesis were concerning mostly the web application, due to the many dependencies among the components in it. For this reason, with more time, its design could be substantially improved.

Also, the GUI can be reworked to gain more appeal. Summarizing, the main directions for future work are:

- Improve the web service implementation.
- Gain more insights regarding repository activities.
- The GUI design objectives are still to be fully achieved.

## References

- [1] “GitHub - <http://www.github.com>.”
- [2] M. Lanza and R. Marinescu, *Object-Oriented Metrics in Practice: Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems*, 1st ed. Springer Publishing Company, Incorporated, 2010.
- [3] “GitHub - <http://tomcat.apache.org>.”
- [4] “Mauricio Aniches CK at - <https://github.com/mauricioaniche/ck>.”
- [5] S. R. Chidamber and C. F. Kemerer, “A metrics suite for object oriented design,” *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476–493, June 1994.