Design of Software Systems

Project Assignment

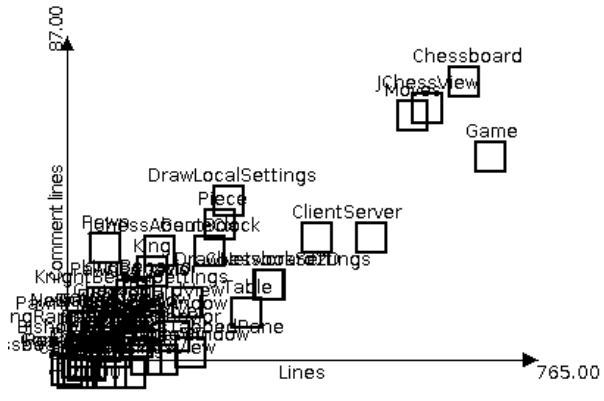
Iteration 1

1 Introduction

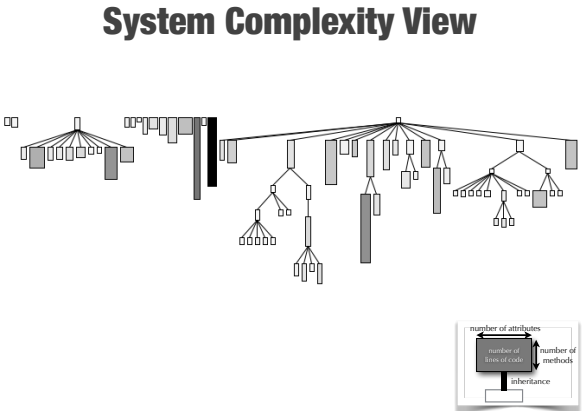
The project for the Design of Software Systems course consists of designing and implementing an extension to SonarQube[1]. SonarQube (previously known as "Sonar") is an open source software project for managing the code quality of your software projects. It can analyze the source code of your project (supporting more than 20 different programming languages, including Java) and create reports and other visualizations about various code-quality aspects. For instance, SonarQube can detect whether source code adheres to certain user-specified coding standards, it can analyze the dependencies between different components, can detect duplicated code and can even automatically find some types of source code bugs. In order to perform such analyses, SonarQube calculates various *metrics* on the source code, such as for instance the cyclomatic complexity and number of lines of code for each method.

One of the interesting aspects of SonarQube is that it is extensible. You can write a plugin to support new programming languages, new metrics or new kinds of code quality visualizations. The goal of this project is for you to design and implement a flexible new visualization plugin named *Polymorphic Views*. A polymorphic view is a visualization of some collection of resources according to one or more user-configurable metrics.

A concrete example of a polymorphic view is a scatter plot of all classes of a certain project, where the X-axis represents the number of lines of code of the classes and the Y-axis represents the number of lines of comments of the classes. Another example is a system complexity view of all classes in a certain Java package, where each class is represented by a box of which the width is relative to the number of attributes of the class, the height is relative to the number of methods and the color depends on the number of lines of code. These two examples are shown in Figure 1 below. Also see the slides of lecture 4 at `http://roelwuyts.be/OSS-1415/OSS-1415-3.Metrics_SoftwareVisualization.pdf`.



(a) Scatter plot



(b) System Complexity View

Figure 1: Example polymorphic views

[1] `http://www.sonarqube.org/`

2 Project Organization and Timeline

The project consists of three iterations. In Iteration 1, you will have to perform an analysis of SonarQube, in order to get acquainted with its design and architecture. This is an essential process to undertake, in order to start the next iteration. In Iteration 2 you will design and implement the polymorphic view extension, and in Iteration 3 you will refactor and/or further extend the extension. Table 1 shows the start and end dates for each iteration.

We expect you to team up into groups of 4 or 5 students. The group composition is up to you, but you must let us know which group you are in **before October 17**, by sending an e-mail to **oss@cs.kuleuven.be**. You will remain in the same group for all three iterations. If you have trouble finding a group, let us know as soon as possible by mailing to the same address.

After you have sent your group composition, your group will be assigned a supervisor. This supervisor is part of the course staff and is there to help you with the project. You are allowed to schedule a 1 hour meeting with your supervisor at most **once per week**. When you schedule a meeting, make sure you prepare **concrete questions** and have **appropriate design artifacts** available to support your questions (e.g. you can show class diagrams for two alternative designs that you propose, and the supervisor can help you decide which design is more appropriate). Note that the supervisor will not do the work for you: if you give insufficient input, the supervisor will not give you an answer.

After each iteration there will be a short evaluation session where you must present and defend your work to the team of supervisors. For the first and last iterations, this defense will be private, i.e., only your group members and the team of supervisors will attend. For the second iteration, the defense will be public, i.e., the other groups will also attend.

Date	Event
10 Oct 2014	Start of iteration 1
24 Oct 2014	Defense of Iteration 1 (private) & start of Iteration 2
21 Nov 2014	Defense of Iteration 2 (public) & start of Iteration 3
19 Dec 2014	Defense of Iteration 3 (private)

Table 1: Project timeline

3 Iteration 1

3.1 Assignment

The goal of this iteration is to analyze the architecture and design of SonarQube. A thorough analysis is essential in order to start the next iteration. Following the steps below will get you started on the analysis.

Running SonarQube:

1. Go to the SonarQube web site at `http://www.sonarqube.com` and download SonarQube 3.7.4 (LTS) and SonarQube Runner (version 2.4).
2. Follow the installation guide at `http://docs.codehaus.org/display/SONAR/Installing` to set up a database server on your machine (we recommend MySQL), create a new database schema and user for SonarQube and configure the SonarQube server and Runner to access this new schema.
3. Once you have the SonarQube server up and running, choose a Java open-source project at will and analyze its source code using the Runner. You can easily find open source projects in a programming language of choice on `http://www.sourceforge.net`. See `http://docs.codehaus.org/display/SONAR/Analyzing+with+SonarQube+Runner` for instructions on how to analyze projects with SonarQube Runner. Note that you may have to add the line `sonar.java.source=1.7` to the `sonar-project.properties` file, if the project uses Java 6 or 7.
4. Experiment with the SonarQube Runner and Server. See what metrics are available by default and check out the available widgets by logging in on the SonarQube web server (the default username/password are `admin`/`admin`) and clicking 'Configure widgets'.
5. Some metrics for Java projects are based on the Java source code, while others are based on the Java bytecode. Compile the Java source code of the open-source project you selected using whatever build system the project uses and then configure SonarQube Runner to analyze the resulting bytecode by adding the line `sonar.binaries=<path to bytecode>` to the project's `sonar-project.properties` file. Run the SonarQube Runner again and see what extra metrics now are available on the server.

Getting to know the SonarQube source code:

1. SonarQube is a collection of several subcomponents, some developed by the SonarQube core developers, others developed by the community. All basic components are open source and are hosted on GitHub. The components that we are interested in are the SonarQube core components, the SonarQube Java support component and the SonarQube Runner. Table 2 shows the GitHub repository URL, the right version and the corresponding commit hash for these three components. You should clone these components to a local repository using git[2] and check out the appropriate commit.
2. SonarQube uses Maven as a build system. Although you will not need to build SonarQube in order to analyze it, you will need to build your extension using Maven in the next iteration, so we recommend that you already install Maven 3.0.5 or newer. In order to browse the SonarQube source code, we recommend importing the source code into Eclipse Luna (4.4.1 or newer) using `File -> Import... -> Existing Maven Projects`. It's OK if Eclipse complains about build errors, because we will

[2]See `http://git-scm.com/documentation` for instructions on how to use git.

Component	Repository	Version	Commit
Core	`https://github.com/SonarSource/sonarqube`	3.7.4	d25bc0e
Java	`https://github.com/SonarSource/sonar-java`	2.4	7e7e633
Runner	`https://github.com/SonarSource/sonar-runner`	2.4	578fc62

Table 2: SonarQube components

not be building SonarQube anyway. When you try to import the SonarQube components, you will see that each component consists out of several subprojects, it's OK to import all of them.

3. Try to figure out what happens what happens when you run the SonarQube Runner. What code gets executed? What is the output of the Runner and where is this output stored. Which component performs the code quality analysis? How does the SonarQube server get access to the code quality results? What is the `sonar-ws-client` project and how does it work? You should try to answer all these questions by experimenting with SonarQube and investigating the source code. You should focus on the `sonar-core`, `sonar-batch`, `sonar-plugin-api` and `sonar-ws-client` projects.
4. Analyze the SonarQube source code with some the source code analysis tools you've seen in the class lectures. Focus on the `sonar-core`, `sonar-batch` and `sonar-ws-client` projects. What do you think about the source code quality?

3.2 Deliverables and Evaluation

At the end of the first iteration, your group is expected to present its analysis to the team of supervisors. The presentation should take no more than **10 minutes**. The purpose of this presentation is for you to show us that you understand the architecture and design of SonarQube at an abstract level. You should bring any presentation material that you consider useful to achieve this goal. You do not have to make slides, but if you want to, *you* must bring the necessary hardware (e.g., a laptop) for showing them to us.

The presentation should include at least the following elements:

- A high-level overview of the SonarQube architecture. This should answer questions such as
  - What happens when you execute the SonarQube Runner?
  - What is the output of the SonarQube Runner and where is it stored?
  - When is the code quality analysis performed and which component is responsible for this?
  - How can the SonarQube server access the code quality analysis results?
  - What is the `sonar-ws-client` project and how does it work?

- A domain model illustrating at least the following key SonarQube concepts and the relations between them: Resources, Metrics, Measures and Dependencies.
- A discussion about the design of SonarQube. What are the strong and weak points of the design? What tools did you use to evaluate the design?
- A discussion about the testing approach of SonarQube. Do they use scenario and/or unit tests?
- A high-level plan on how you will develop the Polymorphic Views extension.
  - Do you think SonarQube provides a clean way of developing this extension?
  - When will the polymorphic view output be generated? When running the SonarQube Runner, when visiting the SonarQube web page, or some other time?
  - Where is the necessary data for generating the polymorphic view stored? How will this data be retrieved?