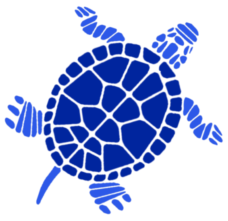
Architecture Design [Draft]



Delft University of Technology

Faculty of Electrical Engineering, Mathematics and Computer Science.

Mekelweg 4,

Delft

Team Members:

Name:	StudentID:	E-mail:
Boning Gong	4367308	boninggong@yahoo.com
Clinton Cao	4349024	C.S.Cao@student.tudelft.nl
Michiel Doesburg	4343875	M.S.Doesburg@student.tudelft.nl
Sunwei Wang	4345967	S.Wang-11@student.tudelft.nl
Tim Buckers	4369459	TimBuckers@gmail.com

SE TA:

Bastiaan Reijm

Course:

Context Project (TI - 2806)

Table of contents

1. Introduction

This document provides a sketch of the architecture of the system that is going to be built during the context project tools for software engineering. It provides a high level overview of the components and interaction thereof of the system.

1.1 Design goals

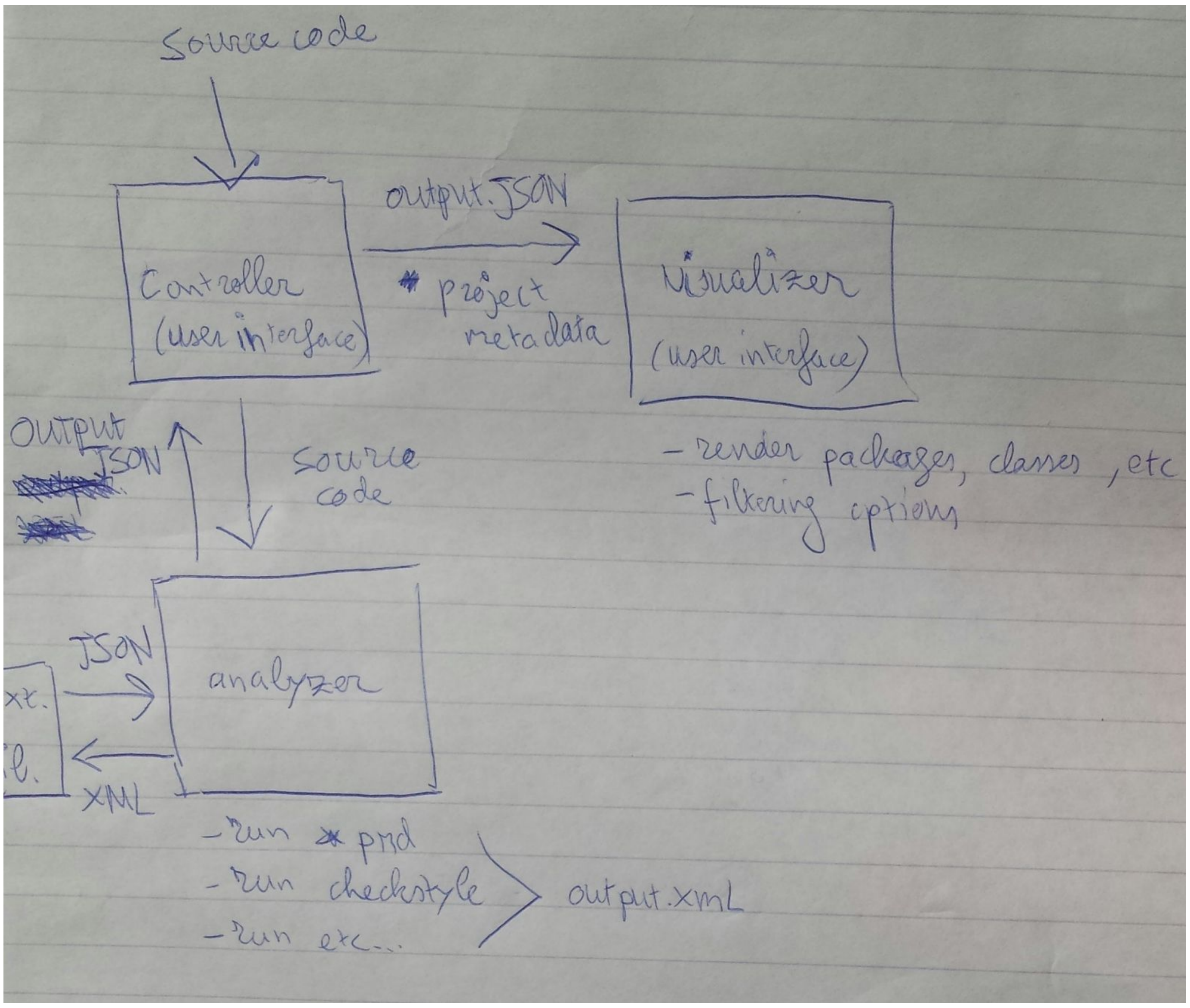
The following design goals will be maintained throughout the project:

- Modularity: our project lends itself extremely well to a modular oriented design. First of all, we want to split the code analysis and visualization part into two different engines; the analyzer and visualizer respectively. These components will run independently of each other, and as such that each component will be easy to replace or adjust without needing to touch the other component. Modularity will also be maintained internally in these two components; adding another SAT for example should be easy to do. Each SAT should be handled independently by the analyzer, and in the end producing one unified output. If another SAT is added, it should simply latch on to the queue and append its output to the unified output in some way. The same holds for the visualizer. Adding a new visualization technique will not interfere with any visualization technique that is already present.
- Code quality: the aspect of code quality is an obvious necessity. Code will be structured in a unified style. Many good coding practices will be applied, like making classes carry just the right amount of responsibility (not too long, not too short), methods shouldn't be too long, using easy to understand variable names, etc… Checkstyle, PMD and code reviews will, among other things, be used to guarantee code quality.

2. Software architecture views

2.1 Subsystem decomposition

The system consists of three main components. The analyzer and the visualizer, and the controller which controls the first two components. The controller is the main program which controls the analyzer and the visualizer. The user provides the source code to the controller and indicates which SATs they want to run. The controller will hand the source code to the analyzer. The analyzer runs the designated SATs and combines their output into XML format. This XML output is translated to JSON with an external library. The JSON output is handed back to the controller. The controller then hands the JSON output and the project metadata to the visualizer which will handle the rendering and visualization of the project. A rough sketch is provided below:



2.2 Persistent data management

The system will have to deal with all kinds of output data from SATs like html, xml among others. This data will be unified into one common type. We have chosen this common type to be xml because many SATs (checkstyle, PMD, findbugs, Emma) already provide xml output. For converting other types of output to xml and for translating the xml into JSON external libraries will be used.

2.3 Choice of programming language

The analyzer and controller will be written in Java. Mostly because the team has the most experience in Java and are most familiar with its frameworks for basic user interfaces.

The visualizer will be written in Javascript. Javascript was chosen for the visualizer because it has very good frameworks to build upon. The treemap visualization framework for example is a solid groundwork to build our visualizer upon. Java does not have these kinds of visualization frameworks.

3. Glossary

SAT
Static analysis tool. A program that takes source code as input and produces as output some measurement of quality of code. These might be metrics like average lines per method, or these might be warnings of possible bugs.

Checkstyle
A static analysis tool mainly used for ensuring a good and homogenous coding style between programmers.

PMD
A static analysis tool which mainly checks for redundant, wrong or suboptimal code. It for example highlights code where bugs may possibly exist or code which will never be reached.

HTML
HyperText Markup Language. It is mainly used as a markup language for webpages.

XML
Extensible Markup Language. It is a markup language for encoding documents to make them both easily readable by humans and machines.

Java
A general purpose programming language that is designed to be easily runnable on different machines through the use of Java virtual machines.

Javascript
A programming language generally used for scripting and enabling interactivity on web pages and web applications. Features many different kinds of frameworks for these purposes.