Requirements
Software Visualization Tool

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)

TU Delft

Table of contents

1. Functional Requirements

This chapter will be about the requirements regarding the functionalities and services of our system. We will be using the MoSCoW method[1] to prioritize the requirements.

1.1 Must Haves

- The tool must be able to analyze the project of the user; the tool will be using static analysis tools to produce an analysis.
- The tool must be able to produce a visualization of the result that was produced from the analysis with the static analysis tools.
- The tool must have the ability to let the user select his / her project.
- The tool should have the ability to let the user visually compare the results between static analysis tools.
- The tool should have the ability to group warnings, that are from the same components, together.

1.2 Should Haves

- The tool should have the ability to produce different kinds of visualization e.g. a visualization where the user can see how many checkstyle warnings there is in a component.
- The tool should have an user interface which enable users to navigate through the system.
- The tool should have the ability to group same kinds of warnings together in the same category. This will be done using the General Defect Classification[2].
- The tool should have the ability to let the user navigate to the location of the warning that is shown in the visualization e.g. the user can click in the user interface in order to see where the warning is from.

1.3 Could Haves

- The tool could have the ability to support multiple programming languages. This means that the tool can be used for projects that are written in different programming languages.
- The tool could have the ability to produce a history-based visualization e.g. the tool can produce a visualization where the user can see a timeline, and how much he / she has improved within this timeline; he / she can see how many warnings there are from a specific static analysis tool in his / her project, in specific period in this timeline.

[1] https://en.wikipedia.org/wiki/MoSCoW_method
[2] http://www.st.ewi.tudelft.nl/~zaidman/publications/bellerSANER2016.pdf

1.4 Won't Haves

- The tool won't have the ability to automatically fix the warnings that are shown in the visualization.

2. Non-Functional Requirements

This chapter is about the non-functional requirements of our product. These requirements does not indicate what our product should do, but what instead indicate the constraints that apply to the system or the development process of the system.

- The system shall be developed using the following languages:
    - Javascript: This is for the visualizer (the component that will be producing the visualizations).
    - Java: This is for the analyzer (the component that will be analyzing the project that is given as input, and producing the right output files for the visualizer).
- GitHub shall be used for code versioning.
- The repository on GitHub shall be open and public.
- The source code shall be analyzed using the following three static analysis tools: Checkstyle, PMD and FindBugs.
- The development shall follow the pull-based development model[3]
- Scrum[4] methodology will be used for the development of the system.
- The analyzer component must have at least 70% of test coverage.

[3] https://docs.google.com/document/d/1Xb4vrF9V78ge_vq4VLrl3YZ2Qq0daCLW2oid2IrGRfA/pub
[4] https://en.wikipedia.org/wiki/Scrum_(software_development)