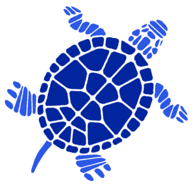
Product Planning



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

Most software developers have heard of static analysis tools (or at least we hope so). These are tools that software developers can use to do the following things:

- Monitor code quality.
- Check whether there are flaws in the source code.
- Check whether the code follows certain standards.

These tools are automated and do not require you to run / execute the code, hence the name static analysis tools. After the tools have finished the analysis, each tool will write its result of the analysis to a file. The format of these files varies between different tools e.g. XML or HTML.

These tools sound like something that every software developer should be using for the development of their software product, but do software developers really pay attention to the results of the analysis? Do they actually try to fix the warnings that they received from the analysis?

Having only a few warnings shouldn't take a lot of time to fix them, but what if the list of warnings is very long? Where do you start? What if each tool has a different name for the same kind of warnings?

Having different kinds of output files from the tools makes it harder for software developers to visualize where the warnings are from. What if most of these warnings are from a very important component of the system? Not being able to visualize these kinds of things, the software developers might spend their time fixing warnings that are not that important. These are the things that discourage software developers to try to fix the warnings, certainly if they are almost done with their project.

The goal of this project is therefore for us, team BlueTurtle, to come up with a solution that improves the way how software developers can visualize the results from the SATs. Our product therefore needs to be able to let software developers analyze their source code and output a visualization of the results from the analysis.

The following topics will be discussed in the rest of this document: In chapter 2 we will be discussing about our product; we will give a high-level product backlog and also a roadmap that will illustrate what are the goals we want to achieve at certain point of time in this project. In chapter 3 we will discussing about the product backlog using user stories and our initial release plan. In the last chapter (4) we will be discussing on how we decide whether we are done with this project or not.

2. Product

The product of this project will be a piece of software that software developers can use to analyze their source code and based on the results of the analysis, our product will output a visualization of the results. In section 2.1 we will use a set of epics to represent a high-level product backlog. In section 2.2 we will use a table to illustrate what are the goals that we want to achieve at a certain point of time in this project.

2.1 High-level product backlog

Below is a set of epics that will represent the high-level product backlog [1]. We've also used MoSCoW method [2] to categorise our high-level product backlog. This is in the appendix.

- As a software developer, I want the system to analyze my source code and get a visualization from the results of the analysis, so I know how I can schedule my time to fix the flaws in my source code.
- As a software developer, I want the system to have a user interface, where the visualization of the results of the analysis, will be shown. This way I don't have to look at all the (different kinds of) output of each SAT, in order to know where the warnings are from.
- As a software developer, I want the system to use different shadings of a color and different shape sizes to illustrate severity of the warnings and the amount of warnings from a component of my system. This way I can visualize which component has a lot of warnings and the severity of the warnings, so that I know where I need to focus on.
- As a software developer, I want the system to group the same kind of warnings together. This way I won't be confused on what the actual problem is, when I am reading the output of each SAT.

2.2 Roadmap

The following table is a sketch of what we want to achieve after each sprint.

Sprint	**Descriptions and goals**
Sprint #1	This is where the research and design phase takes place. In this sprint we want to achieve the following thing(s): ● First sketch / concept of our system ● Draft of documents that are required for this project.
Sprint #2	This is where we will start with the actual implementation of our system. In this sprint we want to achieve the following thing(s): ● Our system is able to read the output of results produced by multiple SATs of our choice.

	• Basic visualization (setup and testing). • Final version of documents from previous sprint (Not the architecture)
Sprint #3	This is where we will add more must-have features to our system. In this sprint we want to achieve the following thing(s): • Our system should be able to create a visualization of the results from different ASATs. • An user interface of our system. • The system should group warnings from the same component (e.g. from same class) together.
Sprint #4	This is where we will add some should-have or could-have features. In this sprint we want to achieve the following thing(s): • Different kinds of visualizations. • The system should also group same kind of warnings together.
Sprint #5	This is where we will add more should-have or could have features and also make sure that must-have features are finished. In this sprint we want to achieve the following things: • More sophisticated visualizations.
Sprint #6	This is where we add more features to the system. In this sprint we would like to achieve the following thing(s): • Being able to click in the user interface in order to see where the warnings are from. • History-based visualization.
Sprint #7	This is where we will implement features that are not finished and also add the following feature: • Support for projects that are written in other programming languages.
Sprint #8	This is the last sprint. In this sprint we want to achieve the following thing(s): • All the must-have features must be implemented in our system. Also as many should-have and could-haves features that we can implement.

3. Product backlog

The product backlog is a list of items that needs to be done. These items are represented with user stories [2].

3.1 User stories

The table on the next page shows the product backlog, each with item with its user story, its estimation (story points) and its priority. The items are sorted by their priorities.

User story	**Estimation (story points)** (1 - 5)	**Priority** (1 - 5)
As a software developer, I want that the system can produce a visualization of the results, produced from the multiple SATs and be able to compare them.	5	1
As a software developer, I want that the system can run an analysis on my source code, using SATs.	5	1
As a software developer, I want that the system can group the warnings, that are from the same component, together.	2	2
As a software developer, I want that the system can group the same kind of warnings together.	2	2
As a software developer, I want the system to have the ability to navigate to location of the warning, by clicking in the user interface	4	3
As a software developer, I want that the system to have different kinds of visualizations.	3	3
As a software developer, I would like the system to show me a history-based visualization e.g. show amount of warnings in my system for each time I used the system.	2	4
As a software developer, I want that the system can support projects that are written in other programming languages	5	5
Total points	28	

3.2 Initial release plan

Sprint	**Milestone**
Sprint 1	Concept of our system.
Sprint 2	Ability to read output of SATs and basic visualization (setup and testing).
Sprint 3	Grouping of warnings from same component (e.g. class), user interface and visualization of the results produced by the SATs.

Sprint 4	Grouping same kind of warnings and different kinds of visualization
Sprint 5	More sophisticated visualization.
Sprint 6	Ability to click on the UI, to navigate to the location of a warning and history-based visualizations.
Sprint 7	Support for other programming languages.
Sprint 8	Final release version.

4. Definition of Done

A key principle of agile software development is “done means DONE!” Therefore, it is important for everyone involved in the project knows and understands what Done means. So DoD should creates a shared understanding of what it means to be finished with a feature, so it can be closed and everyone can agree on that [3].

Our DoD should be split into following three levels: Features, Sprints, Releases.

4.1 Features Level

- Code produced.
- Code commented, checked in and run against current version in source control.
- Peer reviewed (through reviewing pull requests in GitHub).
- Build without errors.
- Unit tests written and passing .
- Passed UAT and signed off as meeting requirements [4].

4.2 Sprints Level

- Unit tests and UAT passed without failures.
- Deployed to system test environment and passed system tests.
- Remaining hours for task during the sprint is set to zero and tasks with highest priority are closed(priority A).
- Product documentation: for product vision, product planning, architecture design and the final report, spelling and grammar need to be checked, and reviewed by at least three group members. All criterias on the guideline must be met.
- Sprint documentation: for sprint backlog and sprint retrospectives, all team members must take part in making them, including spelling and grammar check, and finally reviewed by the entire group.
- Diagrams updated/uploaded, retrospective for the previous sprint is handed in, and sprint backlog for the new sprint is uploaded.

4.3 Releases Level

- Demo and presentation given to the client.
- Final working version of the code.
- Relevant documentation is handed to the client.

5. Glossary

UI	-	User Interface.
DoD	-	Definition of Done.
UAT	-	User Acceptance Testing. A test conducted to determine if the requirements of a specification or contract are met.
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.

6. References

[1] K.S. Rubin. (2013). *Essential Scrum: A Practical Guide to the Most Popular Agile Process*. Pearson Education, Inc.

[2] Information on MoSCoW method: https://en.wikipedia.org/wiki/MoSCoW_method

[3] Information regarding product backlog
http://www.scrum-institute.org/The_Scrum_Product_Backlog.php

[4] All about Agile http://www.allaboutagile.com/definition-of-done-10-point-checklist/

[5] What is DoD?
https://www.scrumalliance.org/community/articles/2008/september/what-is-definition-of-done-(dod)

Appendix A: Using MoSCoW to categorise high-level product backlog.

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.

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 the ability to let the user visually compare the results between static analysis tools.
- The tool should have an user interface which enable users to navigate through the system.
- The tool should have the ability to group warnings, that are from the same components, together.
- 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[1].
- 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.

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.

Won’t Haves

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

[1] http://www.st.ewi.tudelft.nl/~zaidman/publications/bellerSANER2016.pdf