Product Vision

by

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. Who is going to buy the product? Who is the target customer?

This product will be mainly serving as a research tool. It will be useful for researchers who want to gain more insight into the differences between different automated static analysis tools (ASATs). These tools will run over (source) code and give feedback without executing the code itself (Ayewah et al., 2008).

Researchers can use this product to get a quick overview of the outcomes of different ASATs. They could be using this tool for their research.

Another possible group for whom this product is eventually useful are software developers. They can run this product to analyse their own code and compare the different results of ASATs. But this group is not our main focus.

Main customers:
Andy Zaidman (Associate Professor in the Software Engineering Research Group at TU Delft)
Moritz Beller (PhD Student in the Software Engineering Research Group at TU Delft)

2 Which customer needs will the product address?

The main need we try to address is a comparison between different ASATs. We will do this by running different kind of tools on multiple different Java projects. Then we will try to visualize these outcomes in such a way that you get a clear overview of the differences of the tools.

For large projects it is often not feasible to make everything perfect and efficient. Often code and testing do not co-evolve gracefully (Beller et al., 2015). There is always some kind of technical debt (Kruchten et al., 2012).

The consequence is that already existing bug finding tools will produce an incredibly long list of warnings (Ayewah et al., 2010). This makes it hard for big projects to use these tools later on. Using this product you can see which tool fits your project most and why some tools fits better than others. Sometimes a combination of different tools give a better overview. Maybe we will even find room for improvements within an ASAT.

Questions we could be asking:
Are there tools that generally work better than others?
Why do some tools work better for specific projects than others?
Which combination of tools often have better results? Which worse?
Is there room for improvement in specific ASATs?

3 Which product attributes are crucial to satisfy the selected needs, and therefore to the success of the product?

The product should give a clear visualization of different results taken from different ASATs. The visualization should not only be clear, but it should also be meaningful. There are multiple specific attributes needed, depending on the different use cases of the users.

For the best usage for the customer the visualization should generate within a reasonable amount of time and should be easy to interpret. Low data-to-ink ratios and high visual densities tend to work better than minimal “clean” visualizations (Borkin et al., 2013).

Researchers want to be able to see or measure the different qualities of tools. Software developers want constant feedback of the quality and progress they are making. Using this tool they can see which warnings are meaningful, so that they know where they have to pay more attention next time. We want our product to produce visualizations using the seven tasks that Shneiderman mentioned. (Shneiderman, 1996)

Example questions we want to answer:
Does our visualization give a clear overview of the different analysis tools?
To which degree are the outcomes of different tools comparable?
What is the performance for each ASAT that we tested?

4 How does the product compare against existing products, both from competitors and the same company? What are the product's unique selling points?

One unique selling point is the global overview of warnings found by a specific tool. Another unique selling point is the capability to zoom in to different heights of a Java project and get a high overview level.

There are already existing products that are similar. But most products address only one problem. Sometimes the product only focus on visualizing the structure of the code. They show cohesion and coupling in the project. On the other side we can find the products that show a long list of bugs/weak quality code. We want to combine these aspects into one product where you can compare all these different tools.

Existing comparable products on bug finding:

- PMD, Checkstyle, FindBugs and Cobertura: Automated static analysis tools that we will be using to gather information about bugs/bad code. They have their own mini visualization, but these are constrained to the IDE in which they are used. Their way of error showing should be more intuitive and use detailed explanation of defects with automatic fixes where appropriate (Johnson et al., 2013). We want to map the results of these tools individually and combine this in a good clear visualization.

Existing comparable products on visualization of source code:

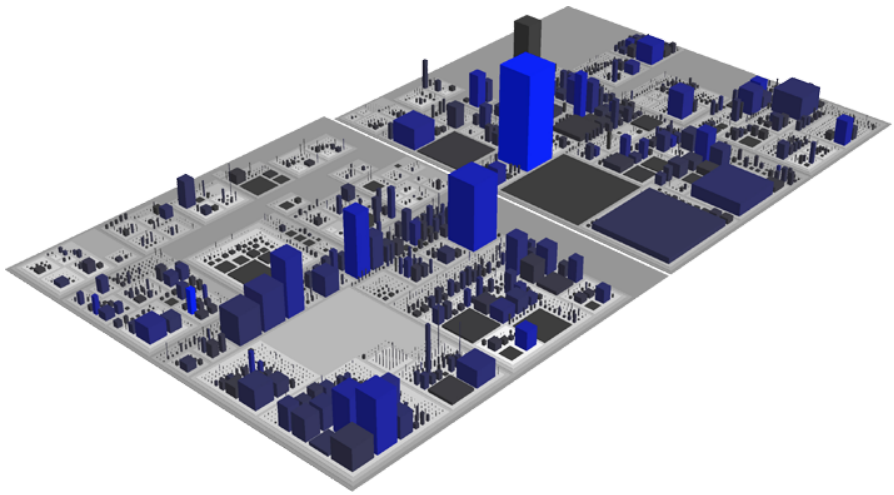
- *CodeCity*: Software that will visualize your java code (Wettel, 2008). It gives an overview of long classes, cohesive classes and other features. Its selling point is their visualization that forms a city where you can 'walk' through. It is not that useful but super trendy.
- *CodeFlower*: Visualizing repository structures and sizes (Zaninotto, 2013). It makes an interactive tree of your files and of how they are grouped. Files with lots of lines of code are given a bigger circle than smaller files. It works for every Github repository.

Our goal is to combine the features of both types. To achieve this we are going to use already existing bug finding tools and combine this with visualization frameworks. Here we try to visualize the different outcomes as clear and meaningful as possible. We will try to achieve this by including easy and useful configuration options for the tool. This should be also considered when implementing a tool that will lead to higher usage of static analysis

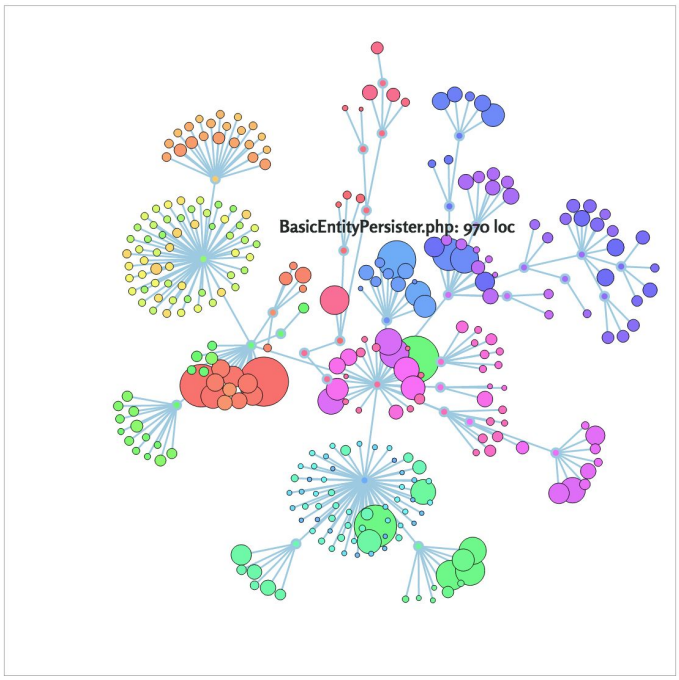
tools for improving software code quality and maintaining coding standards (Johnson et al., 2013). This should all be integrated into one final product.

5 What is the target timeframe and budget to develop and launch the product?

We have ten full weeks after the meeting to develop and present the product. The budget is not discussed, but we don't expect to have any additional expenses.



Example of CodeCity visualization (Wettel, 2008)



Example of CodeFlower visualization (Zaninotto, 2013)

6 References

Ayewah, N., & Hovemeyer, D., & Morgenthaler, J.D., & Penix, J., & Pugh, W. "Using Static Analysis to Find Bugs", IEEE Software, vol.25, no. 5, pp. 22-29, Sept.-Oct. 2008, doi:10.1109/MS.2008.130. Retrieved from http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4602670&tag=1

Ayewah, N. and Pugh, W. “The Google FindBugs Fixit,” in Proc. ISSTA, 2010, pp. 241–252. Retrieved from https://pdfs.semanticscholar.org/a483/be124411ef98e51ca3aa393935fc85bf0a39.pdf

Beller, M., & Gousios, G., & Panichella, A., & Zaidman, A. (2015). “When, How, and Why Developers (Do Not) Test in Their IDEs”. *ESEC/FSE’15*, 179-190. Retrieved from http://www.testroots.org/assets/papers/2015_beller_gousios_panichella_zaidman_when_how_and_why_developers_do_not_test_in_their_ides.pdf

Borkin, M.A., & Vo, A.A., & Bylinskii, Z., & Isola, P., & Sunkavalli, S., & Oliva, A., & Pfister, H. "What Makes a Visualization Memorable?", IEEE Transactions on Visualization & Computer Graphics, vol.19, no. 12, pp. 2306-2315, Dec. 2013, doi:10.1109/TVCG.2013.234. Retrieved from http://cvcl.mit.edu/papers/Borkin_etal_MemorableVisualization_TVCG2013.pdf

Johnson, B., & Song, Y., & Murphy-Hill, E., & Bowdidge, R. (2013). “Why Don’t Software Developers Use Static Analysis Tools to Find Bugs?” 1-10. Retrieved from http://people.engr.ncsu.edu/ermurph3/papers/icse13b.pdf

Kruchten, P., & Nord, R.L., & Ozkaya, I. "Technical Debt: From Metaphor to Theory and Practice", IEEE Software, vol.29, no. 6, pp. 18-21, Nov.-Dec. 2012, doi:10.1109/MS.2012.167. Retrieved from https://www.computer.org/csdl/mags/so/2012/06/mso2012060018.pdf

Shneiderman, B. "The Eye Have It: A Task by Data Type Taxonomy for Information Visualizations," Visual Languages, 1996. Retrieved from http://www.interactiondesign.us/courses/2011_AD690/PDFs/Shneiderman_1996.pdf

Wettel, R., & Lanza, M. (2008). “CodeCity, 3D Visualization of Large-Scale Software”. In *ICSE’08.* May 10–18, 2008, Leipzig, Germany. Retrieved from http://www.inf.usi.ch/faculty/lanza/Downloads/Wett2008a.pdf

Wettel, R. (2008). CodyCity [Image]. Retrieved May 4, 2016, from http://wettel.github.io/codecity.html

Zaninotto, F. (2013). CodeFlower [Image]. Retrieved May 3, 2016, from http://www.redotheweb.com/CodeFlower/