Improving Code Reviews using the Envision IDE

Master Thesis Project Description

Manuel Galbier

Supervisor: Dimitar Asenov
ETH Zürich
24.02.2016

1 Introduction

Whether in large scale software projects, where hundreds of developers work on the same system or in smaller software projects: code reviews are an universally applied, essential practice for achieving a successful evolution of the software system. Not only can defects be found that way, but also improvement of code and knowledge transfer between the developers can be achieved by employing code reviews [2]. The process itself is often time-consuming. In today's code review tools the visualization of changes is typically done in a marked up textual representation of the source code, which uses information supplied by the version control system. Many developers use the aid of modern tools like GitHub's code review feature [1]. This enables the developer to create e.g. pull requests, which have to be reviewed by other team members before they are committed. But even with the aid of these tools, which help in managing code reviews, the developer has only a textual, marked up representation of the changes at hand. The tools do not provide additional information like for example an automatic summary of the changed code, which could help in understanding the changes and therefore support the code review process.

The goal of this project is to help the developer to perform code reviews. To improve the code review process we will design extensions for the Envision project [1]. Envision is a next generation IDE which uses a visual programming interface. By using Envision we are not constrained to only textual representations of e.g. changes between two versions of a software system. We can try to guide the developer through the review process by leveraging the visual capabilities of Envision in an attempt to improve on current tools.

2 Related Work

Bacchelli and Bird [2] investigated the motivations, challenges and outcomes of tool-based code reviews. They observed and interviewed developers and managers and found that the key aspect in code reviews is code understanding. This crucial part also takes most of the time of a review. They came to the conclusion that modern code review tools do not help the developers enough in the process of code understanding. While many modern IDEs integrate tools which aid context and understanding, all the current code review tools they knew of only showed a highlighted diff of the changed files and would not support any of these tools. Their suggestion is that code review tools should help more in the area of code comprehension. Combined with the possibility for the developer to provide the reviewer with context and direction regarding changes they expect it to lead to faster and better code reviews.

Czerwonka et al.[3] state that code reviewing is often the longest part of the code integration activities. They found that code reviews often do not find functional defects which should block submission of the code. Further it is important to assign a code reviewer who has the right set of skills and knowledge to inspect the code for better results. They suggest that the quality of code reviews seems to be higher if the number of changes i.e. the number of included files, for a single review is not too big. In general they state that due its cost code review is a topic which should be better understood and integrated in the software engineering workflow.

[1]`https://github.com/features`

McIntosh et al.[5] studied the relationship between software quality, code review coverage and code review participation. By examining the Qt[2], VTK[3] and ITK[4] projects they found that code review coverage and participation share a significant link with software quality. They showed that poorly reviewed code has a negative impact on software quality in large software systems using modern code review tools. They suggest that the amount of discussion generated during reviews should be considered when making integration decisions.

3 Core goals

This section outlines the core goals of the project.

3.1 Exploration of related work

To get an idea of what already exists in the domain of code review tools related work has to be considered. This also helps to get an idea what Envision can try to improve using its visual capabilities and what pitfalls to avoid. We will explore mainly these areas:

- Visualization of differences
- Code review
- Code maintenance & understanding

3.2 Visualization of two different versions

Envision has the back-end of an AST-based fine-grained version control system [4, 6]. At the moment there is no visualization of differences between two versions of a software project available. In order to supply the developer with this information an appropriate visualization is needed.
At a minimum it should be possible for the developer to see which parts of the code were added, deleted, changed or moved:

- **Added:** Highlight newly added code with color.
- **Deleted:** Mark where something was deleted. Show the deleted code if the developer moves the cursor to the deletion mark.
- **Changed:** Highlight changed code with color. Show next to the changed code how it looked originally if the cursor is moved to the changed block.
- **Moved:** Highlight moved code blocks. Indicate also visually (e.g. with arrows) if a code block was moved and let the developer know where it originates from.

Generally, the tool should not show all possible information at once, but let the developer decide which parts of the information on the changes should be visible at any given moment. With this approach we can avoid an information overflow and provide the developer with an overview.

[2]`http://www.qt.io/`
[3]`http://www.vtk.org/`
[4]`http://www.itk.org/`

3.3 Designing ways to improve code reviews

The main part of the project is to design different efficient ways to support and improve code reviews. There are many possibilities to leverage the visual capabilities of Envision to reduce the time needed for code reviews. A goal is to provide the developer with help for changed code (e.g. what exactly was changed, where are the changes located in the system, etc) and new code (where was the new code added, what is the new code trying to accomplish, etc). We also aim to implement the following features:

- **Summary of changes:** Instead of listing all the changes we want to provide the developer a summary of the changes made. This could be achieved for example by using the zoom feature of Envision. If zoomed out, the changes could be abstracted, e.g. instead of showing the detailed change information it is highlighted that a certain module contains changes. If the developer zooms into the changed module the changed classes become highlighted and finally the changes themselves become visible.

- **Comment system:** The reviewer should be able to leave comments on the changed parts for the developer. The developer on the other hand should easily be able to see which parts of his changes were commented on and mark parts which were changed by the reviewer. This could be achieved by having textboxes with the comments next to the changes which they refer to.

Nearing the end of the thesis it would be interesting to evaluate the implemented features and compare them to the features of current modern code review tools.

4 Possible extensions

This section outlines possible additional goals of the project.

4.1 Provide guidance to the reviewer

Before the code is reviewed it could be possible to annotate parts of the changed code with additional information. In some cases the developer may want to communicate some parts of his reasoning directly to the reviewer instead of adding comments which would then be part of the software system. The visual capabilities of Envision can benefit the presentation of such additional information.

- **author-defined guidance:** the author of the change could for example define an exploration path which helps the reviewer to get an overview of the changes and where to start. This could be realised by using the 2D layout of Envision where the reviewer could step-by-step traverse the defined review path. Further it could be possible to show additional notes from the author which correspond to the current review step. Using the enhanced comments of Envision the author could not only add textual comments but also show visual aids like schematics or diagrams which aid comprehension of the change. This way the author himself can also point to critical parts of the change, where feedback would be especially important.

- **general guidance:** It could be helpful to the reviewers if there are some general checklists integrated in the code review system which help to make sure the reviewer does not forget to check certain parts of the changes which should always be checked independent of the concrete change. This could cover topics like coding convention, checking for existence of commented out code, software architecture diagrams etc.

4.2 Detect Refactorings

A way to further improve code understanding could be to detect refactorings of the code, e.g. name changes and show them as one semantic change instead of showing multiple changes to the code. In the case of name changes, a possibility could be to color code a changed name and show a list which contains the colored names and what its original name was.

4.3 Check for Completeness & Consequences

Provide the developer with information regarding the completeness of the changes, e.g. if all events which could be received at a certain interface of the software are handled. Other completeness checks could be realised by highlighting the software components which contain similar code to the change and need to be inspected by the developer if the change must be applied to these components as well. Using Envision the context of the similar code components can easily be identified due to the visual representation of their parent components. This helps in deciding whether the change should apply for these components.
Another interesting topic could be to show the developer what consequences the changes have on the system. For example highlight affected modules in Envision which directly use the changed code. Alternatively it could be indicated by using arrows which parts of the system use the changed code and which parts are used by it. This way the developer gets an overview over which parts of the software system are directly impacted by the change.

5 Schedule

Task	Date	Duration (months)
Explore related Work	01.03.2016	0.5
Design and implement diff visualization	15.03.2016	1
Design and implement code review basics	15.04.2016	1
Work on extensions	15.05.2016	2
Evaluation	15.07.2016	0.5
Write report & finish implementation	01.08.2016	1
End of project	01.09.2016	

References

[1] Dimitar Asenov and Philipp Muller. Envision: A fast and flexible visual code editor with fluid interactions (overview). In *Visual Languages and Human-Centric Computing (VL/HCC), 2014 IEEE Symposium on*, pages 9–12. IEEE, 2014.

[2] Alberto Bacchelli and Christian Bird. Expectations, outcomes, and challenges of modern code review. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 712–721. IEEE Press, 2013.

[3] Jacek Czerwonka, Michaela Greiler, and Jack Tilford. Code reviews do not find bugs: how the current code review best practice slows us down. In *Proceedings of the 37th International Conference on Software Engineering-Volume 2*, pages 27–28. IEEE Press, 2015.

[4] Balz Guenat. Tree-based version control in envision. ETH Zürich, 2015.

[5] Shane McIntosh, Yasutaka Kamei, Bram Adams, and Ahmed E Hassan. The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 192–201. ACM, 2014.

[6] Martin Otth. Fine-grained software version control based on a program's abstract syntax tree. ETH Zürich, 2014.