Automation in Modern Code Review

Our research group at [redacted] is conducting a research project aimed at understanding code-changes inside code review and possible ways to automate the code review process/activities.

In today's software engineering workflow code review plays an integral part. It helps us to find defects, ensures maintainability, serves as a tool to transfer knowledge, and is a means to communicate progress. The dominant practice in industry is still the manual inspection of the artefact undergoing a change. This comes with a few downsides:

- it is time consuming (developers spend on average six hours per week reviewing changes of others);
- developers need to switch context away from their current work;
- blocking issues (code defects) are often not found and most changes concern maintainability;
- usefullness is dependent on the experience of the reviewer and the quality of the reviews;
- people's (hierarchical) roles inside teams may influence the outcome of the review;

In our research, we investigate ways to automate the code review process. Our goal is to define approaches able to automatically fix issues that emerge during code review by analyzing information available in static and historical data.

To this extend, we have analyzed over 500 code review changes of eight open source projects (available on Gerrit) and categorized them accordingly into a fine-grained taxonomy of code-review changes. As results of this analysis, we observed that there exist many recurring issues which could potentially be fixed automatically by research tools and prototypes.

To perform more steps toward our research direction we are interested to investigate how code review is performed by industrial and open source developers as well as by research organizations.

Given your solid experience in software engineering and code review practices, we kindly ask you to fill in the following (brief) questionnaire.

Thank you for your effort.
Best regards,

[redacted]

* Erforderlich

Pre-Questionnaire

1. **What is your current job?** *
   Markieren Sie nur ein Oval.
   - ◯ Open Source Developer
   - ◯ Industrial Developer
   - ◯ Senior Researcher
   - ◯ CS Student
   - ◯ Other Occupation

2. **Approximatively, what is the size (in terms of lines of code) of the system you are contributing in most?** *
   _____

3. **What is the approximate size of the development team of the system you are contributing to most?** *
   _____

4. **How many years of programming experience do you have?** *
   Markieren Sie nur ein Oval.
   - ◯ < 2
   - ◯ Between 2 and 5
   - ◯ Between 5 and 8
   - ◯ > 8

5. **How do you rate your programming experience?** *
   Markieren Sie nur ein Oval.

	1	2	3	4	5	
Poor	◯	◯	◯	◯	◯	High

Taxonomy of changes in Code Review

When answering the questions below, please focus on the overall process of code review regardless of the specific code review tool you are using at your work.

Please open the Taxonomy under the following link:

[redacted]

6. **What is code review?** *

Taxonomy of Changes in Code Review (1/2)

Artifact	Activity	Category	Topic	Detailed Change	
Production & Test Code	Maintainability / Perfective Maintenance (Modification of a software product after delivery to improve performance or maintainability)	Documentation	Textual Documentation Issues concerning the documentation through textual representation, such as naming of classes, method, variables. This also includes license headers, typos in either line comments or javadoc	**Naming** Problems relating to software element (=methods, classes, variables, etc) names that do not conform to the naming policy of the project	
				Comments Explanations of complex code fragments, classes, methods. Issues include wrongly placed comments, missing comments, missing or wrong Javadoc etc.	Line Javadoc
				License Header Issues regarding missing or wrong license headers inside source-files	
				Other	
			Language Supported Documentation Documentation through statements/elements that the programming language offers (e.g., java public modifier to document that it is accessible from the outside)	**Immutability** Not declaring variable to be immutable when it should have been or declaring it immutable when it should have not been	
				Visibility (Modifiers) Software element (e.g. method, variable, class) has too much or too restricted visibility	
		Style		**Brackets & Braces** e.g., single statement after a conditional branch	
				Indentation consistent indentation of the code	
				Blank Lines excess of blank lines or too few blank lines or wrong split of lines	
				Long Lines code statement too long, over a specific amount of characters	
				Whitespace Usage usages of blank spaces in the code	
				Grouping grouping of methods with related functionality or adding class variables at the beginning of the class	
				Commented out code remove code that is commented out (also TODO and FIXME)	
		Structure	Solution Approach Solution approach defects require an alternative implementation method. For example, replacing the program's array data structure with a vector and knowing the existence of prebuilt functionality that could be used instead of a self-programmed implementation would be considered a solution approach defect. Therefore, solution approach defects are not about reorganizing existing code but rethinking the current solution and implementing it in a different way.	**Semantic Duplication** Code structures that have a similar intention but are implemented syntactically different	
				Semantic Dead Code Code fragments that are executed, but they do not serve any meaningful purpose and/or have no effect on the result	
				Change Function Change function call to another function because it uses old or deprecated functions	
				Standard Coding Conventions Use exceptions for error messaging instead of return values, use predefined constants instead of magic numbers etc.	
				New Functionality new functionality to ensure evolvability, e.g., create new classes, methods to make code more maintainable	
				Testing Issues regarding test coverage, wrong tests, additional tests etc.	
				Other	
			Organization Defects that can be fixed by applying structural modifications to the software. Moving a piece of functionality from module A to module B is a good example of this.	**Imports** Issues with wrong or missing or unused import statements	
				Move Functionality move functions, part of functions, or other functional elements to a different class, file, or module	
				Long Sub Routine split long and complex functions into multiple functions	
				Dead Code remove code that is never reached and executed	
				Duplication / Redundant Code remove duplicate code or code that is not used	
				Complex Code / Simplification restructure or rewrite implementation to make it more understandable	
				Statement Issue splitting, combining or otherwise reorganizing a statement inside a function	
				Consistency Means the need to keep code consistent in a sense that similar code elements operate in a similar fashion and are more or less symmetrical. For example, similar tasks in similar classes should have similar implementations	
				Other	

Taxonomy of Changes in Code Review (2/2)

	Functionality / Corrective Maintenance (Reactive modification of a software product performed after delivery to correct discovered problems.)	Interface Communication with a different part of the system	**Function Call** call to another part of system or libary is incorrect or missing
			Parameter function call or other interaction has incorrect or missing parameters
		Logic	**Compare** mistake in a comparison statement
			Compute computations produce incorrect results
			Wrong Location correct operation is performed, but it is done too soon or too late
			Algorithm/Performance inefficient algorithm is used
			Other
		Resource	**Variable Initialization** Variables are left uninitialized prior to use. Uninitalized variables may contain any value and using such variable for comparison or calculation produces arbitrary results.
			Memory Management Mistake is made in handling the system memory.
			Data & Resource Manipulation Defects related to manipulating or releasing data or other resources.
		Check	**Check Function** when a function is called there is also a need to check that the value returned is valid and that no error occurred
			Check Variable there is a need to check variable
			Check User Input the need to validate user input
		Larger Defects	**Completeness** partially implemented feature
			GUI Defects in the user interface code relating to the consistency of the user-interface, and to the options made possible to the user in each situation.
			Check outside code Defects that required that part of the application code that was not under review to be checked, as it was likely to contain incorrect code based on the current review.

Other Changes (Config files,Scripts, CI/CD, README etc.)	**Commit Message**
	CI / CD configurations
	ASAT configurations
	Language or Framework specific
	Scripts
	README
	HTML
	VCS
	External Software Documentation
	Runtime Configurations docker-configs, ansible playbooks, deployment configs etc.

7. **Do you think the taxonomy covers all changes that occur in code review?** *

Markieren Sie nur ein Oval.

- ◯ Yes
- ◯ No

8. **If your previous answer was "No", please explain why you think the taxonomy is not exhaustive? Otherwise, you can also send us a modified version of the taxonomy per e-mail to** [redacted]

9. **In your opinion, issues regarding which Categories/Topics (above taxonomy) occur the most inside code review? Please create a ranking listing the 5 most frequent ones.** *

10. **What kind of feedback do you expect from other developers during code review?** *

11. **What kind of feedback do you usually receive from other developers during code review?**

12. **What kind of feedback would you expect from recommender-tools during code review?** *

13. **What kind of automation do you envision for automating code review practices? Please while answering, keep in mind that the envisioned automated tools should be related to the automation of detection/fixing the aforementioned code review issues.** *

14. **In regards to Documentation issues, what kind of automation do you envision for the fixing and detection of these issues?** *

15. **In regards to Style issues, what kind of automation do you envision for the fixing and detection of these issues?** *

16. **In regards to Structure issues, what kind of automation do you envision for the fixing and detection of these issues?** *

17. **In your opinion which Topics of code changes and their Detailed Changes could be automatically detected and fixed by tools?** *

18. **Briefly describe how you would approach the detection and fixing of the issues described in your last answer.** *

Thank you for participating!

The results of this survey will help us greatly in building a first prototype towards automating parts of the code review process.

19. **If you have any additional comments or questions, please feel free to write them here or send us your feedback per e-mail.**