# SUPPORT SYSTEMS FOR MODERN CODE REVIEWS (II)

**This briefing reports scientific evidence of 20 studies that investigate support systems related to the understanding of code changes and managing code reviews.**

---

# FINDINGS

**Understanding the code changes that need to be reviewed.**

Refactoring changes code structure to improve testability, maintainability, and quality without changing its behavior. Supporting the review of such changes has been the focus of refactoring-aware tools.

Refdistiller aims at detecting behavior-changing edits in manual refactorings [D3]. The tool uses two techniques: (a) a template-based checker that finds missing edits; and (b) a refactoring separator that finds extra edits that may change a program's behavior. In a survey of 35 developers of the Gerrit project, researchers found that it would be useful to differentiate between refactored and behavior-changing code, making reviews more efficient and correct.

ReviewFactor is a tool able to detect both manual and automated refactorings (made in an IDE) [D9]. The evaluation of the tool showed that it can detect behavior-changing refactorings with high precision (92%) and recall (94%). When it does not detect them, it fails because the interleaving of refactoring and non-refactoring changes, and the composite of multiple refactorings.

CRITICS is an interactive approach to review systematic code changes [D10]. It allows developers to find changes similar to a specified template, detecting potential mistakes. The evaluation indicates that: (a) six engineers at Salesforce, who used the tool, would like to have it integrated in their review environment; and (b) the tool can improve reviewer productivity, compared to a regular diffing tool.

An inspection of 453 code changes in open source projects revealed that up to 29% of the changes are composite, i.e., address different concerns [D16]. The researchers propose automatically separating unrelated code changes to create cohesive and self-contained sub-changes that are easier to understand. A preliminary user study suggests that the understanding of code did indeed improve when the changes were partitioned.

Other research looked into the order in which changed files should be presented to the reviewer to achieve an effective review process [D1]. The study used logged review navigation data, interviews and an online survey to determine the following main principle for the ordering: group related change parts as closely as possible.

Another contribution to improve the understanding of changed code suggests identifying the so-called "salient" class, i.e., the class in which the main change was made and which affects changes in other dependent classes [D11]. The researchers hypothesize that reviews could be more efficient if the salient class would be known, making the logic of the commit easier to understand. A preliminary evaluation (questionnaire-based) with 14 participants showed that the knowledge about the salient class improves the understanding of a commit.

A similar idea is implemented in BLIMP tracer, which inspects the impact of changes on a file level, rather than on a class level [D14]. The tool was evaluated with 45 developers at Dell EMC and the researchers found that it improved speed and accuracy of identifying the artifacts that are impacted by a code change. Furthermore, the researchers observed that the tool helped to better understand the system architecture.

MultiViewer is a code change review assistance tool that calculates metrics to better understand the change effort, risk, and impact of a change request [D20].

A step further goes the approach implemented in the tool GETTY, which aims at providing meaningful change summaries by identifying changed invariants through analyzing code differences and test run results [D12]. With GETTY, reviewers can more easily determine if a set of code changes have produced the desired effect. The approach was evaluated with the participation of 18 practitioners. The main finding was that GETTY substantially modified the review process to a hypothesis-driven. This process change led to better review comments.

Another direction of research for improving code understanding for reviews uses visualization of information. For example, ViDI supports visual design inspection and code quality assessment [D15]. The tool uses static code analysis reports to identify and visualize critical areas in code, display the evolution of the amount of issues found in a review session, and allow the reviewer to inspect the impact of code changes.

Another tool called Git Thermite focuses on structural changes made to source code [D13]. The tool analyzes and visualizes data (metadata gathered from GitHub, code metrics for the modified files, and static source code analysis of the changes) from pull requests.

OPERIAS, yet another tool, focuses on the particular problem of understanding how particular changes in code relate to changes to test cases [D18]. The tool visualizes source code differences and a change's coverage impact.

Finally, a tool was developed to improve the review process of visual programming languages (such as Petri nets) [D7]. It supports the code review of visual programming languages, similar to what is already possible with textual programming languages (diffs, discussion threats, bug tracking integration, file lists, participant list, and notifications).

*What we think: There has been a wide range of research on improving the understanding of changed code, patches and pull requests, spanning from rearranging information, showing the impact to visualize changes. Many of the approaches are, however, prototypes and have not been shown to be effective, beyond the proof of concept. While some evaluation results are impressive, investigations on the practical benefit of the approaches, in particular on the efficiency and quality of reviews, are needed.*

**Managing code reviews.**

Before code hosting platforms, such as Github, became popular and supported code reviews, researchers investigated how to provide support for reviews in IDEs. SeeCode integrates with Eclipse and provides a distributed review environment with review meetings and comments [D26].

Similarly, ReviewClipse supports a continuous post-commit review process [D28].

Scrub combines regular peer reviews with reports from static source code analyzers in a standalone application [D5].

Java Sniper is a web-based, collaborative code reviewing tool [D25].

All these early tools have been outlived by modern code hosting and reviewing infrastructure services such as GitHub, GitLab, BitBucket, Review Board, and Gerrit. However, while these platforms provide basic code reviewing functionalities, research has also looked at improving the reviewing process in different ways [D2].

For example, researchers suggested continuous code reviews that allow anyone to comment code they are reading or reusing, e.g., from libraries [D27]. Developers can then push questions and comments to upstream authors from within their IDE, without context switching.

Fistbump is a collaborative review platform built on top of GitHub, providing an iteration-oriented review process that makes it easier to follow rationale and code changes during the review [D29]. Furthermore, the tool integrates issue management directly into the displayed source code, supports real time updates, and can show entire files (not only changed code) during reviews.

*What we think: Code review management (creating a review, assigning reviewers, enabling discussions) is now supported by many mainstream software development platforms. Current and future research seems to focus on integrating code review closer into the development process to reduce the negative effect of context switching and to streamline the review information so that the decisions made in the process contribute to knowledge and rationale of the performed changes.*

*Who is this briefing for?*

Software engineering practitioners who want to make decisions about code reviews based on scientific evidence.

*Where the findings come from?*

All findings of this briefing were extracted from the research studies identified through a literature review.

*What is included in this briefing?*

Summaries of research papers in the form of takeaways for the practitioners.

*To access other evidence briefings on code reviews:*

https://rethought.se/modern-code-reviews/

*For additional information about code review research contact -*

deepika.badampudi@bth.se
michael.unterkalmsteiner@bth.se
ricardo.britto@bth.se

**References**

| ID | Title | Link |
|---|---|---|
| D1 | On The Optimal Order Of Reading Source Code Changes For Review | https://www.sback.it/publications/icsme2017.pdf |
| D2 | On The Need For A New Generation Of Code Review Tools | http://tobias-baum.de/rp/reviewtools.pdf |
| D3 | Refdistiller: A Refactoring Aware Code Review Tool For Inspecting Manual Refactoring Edits | http://web.cs.ucla.edu/~miryung/Publications/fse2014demo-refdistiller.pdf |
| D5 | Scrub: A Tool For Code Reviews | https://spinroot.com/gerard/pdf/ScrubPaper_rev.pdf |
| D7 | Code Review Tool For Visual Programming Languages | Please contact one of the authors of this evidence briefing to receive a copy of this paper. |
| D9 | Refactoring-Aware Code Review | https://people.engr.ncsu.edu/ermurph3/papers/vlhcc17-refactoring.pdf |
| D10 | Interactive Code Review For Systematic Changes | http://web.cs.ucla.edu/~tianyi.zhang/critics.pdf |
| D11 | Salient-Class Location: Help Developers Understand Code Change In Code Review | Please contact one of the authors of this evidence briefing to receive a copy of this paper. |
| D12 | Semantics-Assisted Code Review: An Efficient Tool Chain And A User Study | https://par.nsf.gov/servlets/purl/10061392 |
| D13 | Pharo git thermite a visual tool for deciding to weld a pull request | http://bergel.eu/MyPapers/Salg17-GitThermite.pdf |
| D14 | Blimp Tracer: Integrating Build Impact Analysis With Code Review | http://rebels.ece.mcgill.ca/papers/icsme2018_wen.pdf |
| D15 | Code Review: Veni, Vidi, Vici | http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.726.2634&rep=rep1&type=pdf |
| D16 | Partitioning Composite Code Changes To Facilitate Code Review | http://home.cse.ust.hk/~hunkim/papers/tao-msr2015.pdf |
| D18 | Visualizing Code And Coverage Changes For Code Review | http://www.gtii.sback.it/publications/fse2016td.pdf |
| D20 | Multi-Perspective Visualization To Assist Code Change Review | https://www.researchgate.net/profile/Peng_Liang4/publication/322987202_Multi-Perspective_Visualization_to_Assist_Code_Change_Review/links/5a7b05060f7e9b41dbd74df3/Multi-Perspective-Visualization-to-Assist-Code-Change-Review.pdf |
| D25 | Design And Implementation Of Java Sniper - A Community-Based Software Code Review Web Solution | Please contact one of the authors of this evidence briefing to receive a copy of this paper. |
| D26 | Seecode - A Code Review Plug-In For Eclipse | Please contact one of the authors of this evidence briefing to receive a copy of this paper. |
| D27 | Continuous Code Reviews: A Social Coding Tool For Code Reviews Inside The Ide | http://hpi.uni-potsdam.de/hirschfeld/publications/media/Duerschmid_2017_ContinuousCodeReviews_AcmDL.pdf |
| D28 | Adopting Code Reviews for Agile Software Development | http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.467.7668&rep=rep1&type=pdf |
| D29 | A Collaborative Code Review Platform For Github | Please contact one of the authors of this evidence briefing to receive a copy of this paper. |