Visualizing Code and Coverage Changes for Code Review

Sebastiaan Oosterwaal,* Arie van Deursen,* Roberta Coelho**
Anand Ashok Sawant,* Alberto Bacchelli*
*Delft University of Technology, The Netherlands
**Federal University of Rio Grande do Norte, Brazil
sebastiaan.oosterwaal@gmail.com, Arie.vanDeursen@tudelft.nl, souzacoelho@gmail.com,
A.A.Sawant@tudelft.nl, A.Bacchelli@tudelft.nl

ABSTRACT

One of the tasks of reviewers is to verify that code modifications are well tested. However, current tools offer little support in understanding precisely how changes to the code relate to changes to the tests. In particular, it is hard to see whether (modified) test code covers the changed code. To mitigate this problem, we developed OPERIAS, a tool that provides a combined visualization of fine-grained source code differences and coverage impact. OPERIAS works both as a stand-alone tool on specific project versions and as a service hooked to GitHub. In the latter case, it provides automated reports for each new pull request, which reviewers can use to assess the code contribution. OPERIAS works for any Java project that works with maven and its standard Cobertura coverage plugin. We present how OPERIAS could be used to identify test-related problems in real-world pull requests. OPERIAS is open source and available on GitHub with a demo video: https://github.com/SERG-Delft/operias

CCS Concepts

•**Software and its engineering** → **Software maintenance tools; Software configuration management and version control systems;** *Integrated and visual development environments;*

Keywords

code review, software testing, software evolution

1. INTRODUCTION

Code review consists in the manual assessment of source code changes by developers other than the author and is mainly intended to identify defects and quality problems before the deployment in a live environment [9]. Several studies provided evidence that code review supports software quality and reliability crucially [8, 19].

Modern code reviews (MCR) [9], as currently used in most large close- and open-source software (OSS) projects, are informal, asynchronous, and supported by tools. Popular examples of code review tools are Microsoft's CodeFlow [9], Google's Gerrit [5], and GitHub's pull-request (PR) mechanism [4].

Most code review tools offer limited support to help reviewers in evaluating the quality of a change, other than basic highlighted textual differencing. Particularly, review tools offer little information on how a code change affects test coverage, even though developers reported the lack of testing contributing to faulty changes being committed to the repository [13] and coverage as one of the most importance pieces of information they use when assessing a code change [21]. Changes to existing code requires a retest, since they could potentially invalidate the test suite [15].

In this paper we present OPERIAS, a tool we devised to try to mitigate this problem. OPERIAS enriches code review tools with *fine-grained test coverage change information*. It comprises two parts: (1) The core part, which accepts two versions of a software project, computes the differences in both source code and statement coverage, and outputs a report in XML and HTML format; and (2) the code review extension part, which runs the core as a service and connects it to GitHub, generating a report for every opened PRs to provide fine-grained test coverage information at review time.

As a preliminary assessment, we use OPERIAS to analyze PRs from three OSS projects. Results show that OPERIAS provides reviewers with new information for 27% to 71% of the PRs and that it could be useful in different scenarios, e.g., showing that a code change affects the coverage of a class not modified in the PR.

2. OPERIAS IN A NUTSHELL

OPERIAS is a tool to collect, analyze, and visualize code change and related test coverage information to support code review.

2.1 Implementation Details

OPERIAS works for Java, builds upon the Maven [1] setup (tests are executed with the Surefire plugin), and obtains *statement* and *condition* coverage information from the Cobertura plugin.

Given two versions of such a maven project, OPERIAS produces an XML and a HTML report that provides the combined visualization of the changes in the code as well as in the test coverage. The two versions can be in two separate directories or can be identified as two commits (or tags) in a git repository. To get the changes between the two folders, we use Myer's diff algorithm [16] and annotate them with test coverage information [17].

OPERIAS can be used in two ways. The first way is as a standalone tool, whereby a report is generated on a local machine for two different versions of a project; this standalone version can easily be converted into a maven plugin to make it part of the standard build cycle. The second way is as a service hooked to git or GitHub: With this, when a PR is opened on GitHub, OPERIAS is run to visualize the changes in code and coverage introduced by the PR; the service notifies GitHub users by automatically adding a comment to the PR and providing a link to the visualization (Figure 1).

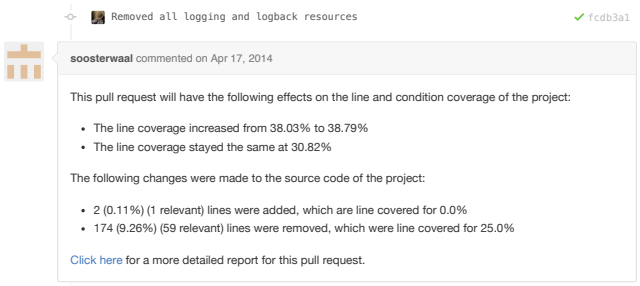

Figure 1: An OPERIAS generated comment on GitHub

2.2 Reporting Changes And Test Coverage

OPERIAS generates browsable reports to visualize code changes together with the corresponding test coverage information. We detail them from the least to the most fine-grained.

Project Overview. The 'project overview' is the report (Figure 2) in which all packages are displayed. By clicking on a package, all changed classes within this package appear. For every class and package, two bars visualize the status of condition and statements coverage. These bars use four colors (also used in the 'class view' with the same semantic): *light green* indicates parts covered in both the original and the version, *dark green* indicates an increase in coverage in the new version, *light red* indicates parts covered neither in the original nor in the new version, and *dark red* indicates parts that are no longer covered in the new version. Coverage percentage points are visualized in the report.

The 'project overview' reports also provides an indication for deleted and newly created classes. A shaded row means that package or class was deleted. In that case, the coverage bars indicate the coverage of the original file by only using the light colors. If a class is new, the bars consists of only dark red and dark green parts, which indicate the revised coverage percentage of the class.

Test View. The 'test view' report (Figure 3) contains information on source changes in test classes (coverage does not apply). To support reviewing tests, we show the outcome of the execution of the test cases. We show, for both the original and revised versions, a list of failed or errored test cases. When clicking on a test case in the list, it shows whether it failed or errored and see the complete stacktrace generated by the test suite.

Class View. The finest-grained visualization is offered by the 'class view' report, which can be accessed by clicking on any class in the 'project overview'. In the report, up to four code views are shown: *original file*, where the original file is shown with the coverage information for that version of the code (as expected, green means covered, red means not covered); *revised file*, which corresponds to the previous view, but showing the new version of the file; *source changes*, where only source changes between the versions (since red and green are used for conveying coverage information, we use the shaded background to mean that the line was deleted and a box around a line or a group of lines means that these lines were inserted in the new version); and *combined view*, the most characteristic view of OPERIAS, where it shows both source changes (similarly to the previous view) and coverage information for both versions (Figure 4) using the four colors that are used to indicate a change in coverage in the same way as described above, but now for specific lines of code.

These four views are available for changed files. For added files, only the revised file view is shown including the coverage information, for deleted files, only the original file view is shown. When opening a changed test file, only the source diff view is viewable since there is no information about coverage available.

Table 1: Distribution of test coverage change across pull requests

Project	Test coverage across pull requests		
	Decreased	Stable	Increased
Bukkit	38%	29%	33%
JUnit	20%	73%	07%
Wire	25%	35%	40%

3. REAL-WORLD USAGE SCENARIOS

We present real-world usage scenarios to provide initial anecdotal evidence on how the support that OPERIAS offers could be potentially beneficial. We explore three OSS projects (JUnit [6], Wire [7], and Bukkit [2]) from different application domains and size, and hosted on the GitHub platform.

Overall applicability. As a first step, we get an indication of the general *applicability* of OPERIAS. We check the distribution of changes in test coverage across all the PRs of the selected project. We do so by running OPERIAS core on the entire code history and computing the effects of each single PR on the test coverage of the overall project. Table 1 summarizes the results, showing the proportion of PRs in which test coverage decreases, is stable, or increases. Results show that for JUnit (a well-tested system) only few PRs increase the coverage, while for Bukkit and Wire (with less coverage to start with) at least a third of PR increase it. More extensive metrics and underlying causes are discussed in the accompanying thesis [17]; here we note that OPERIAS would provide previously unavailable test coverage change information on those PRs for which the coverage has changed, for a minimum of 27% PRs for JUnit, up to a maximum of 71% PRs for Bukkit.

Potential usefulness. As a second step, we investigate the potential usefulness of OPERIAS for reviewers. To do so, from the three projects we manually inspect several PRs in which test coverage is either increased or decreased. The complete analysis can be found in the accompanying thesis [17], here we limit ourselves to interesting PRs from JUnit.

PR/#767: In this PR, a new 'plugin' package is added. OPERIAS' 'project overview' shows the reviewer that all the newly created classes are dark green and fully (100%) tested (figure omitted for space reasons, available in [17]). Furthermore, the PR changed another class and the reviewer can see a small dark red bar, indicating new code that is not tested. The reviewer is able to click on that class and, with the combined 'Class View' (Figure 5), see exactly which lines were added and where testing is lacking.

PR/#896: In this PR, the contributor makes a 1-line change to one class and adds 117 test lines for this class. While this sounds like a good PR, using OPERIAS the reviewer can see (Figure 6) that the change affects the statement coverage of a completely different class ('EachTestNotifier') reducing its coverage by 10%. Even though this class is not part of the original PR, OPERIAS shows it because its coverage is affected by the changes under review. Industrial reviewers reported that knowing which parts of the code are indirectly affected by a change is crucial to asses its quality [21]; using OPERIAS indirect changes in coverage are easy to detect.

PR/#646: In this PR, five new test cases added to the project, next to a few changes in the code. Even if the test cases would properly test new or existing code, they are not executed because they are not added to the 'AllTests' class; in fact, for a test case to be successfully executed within the JUnit project,

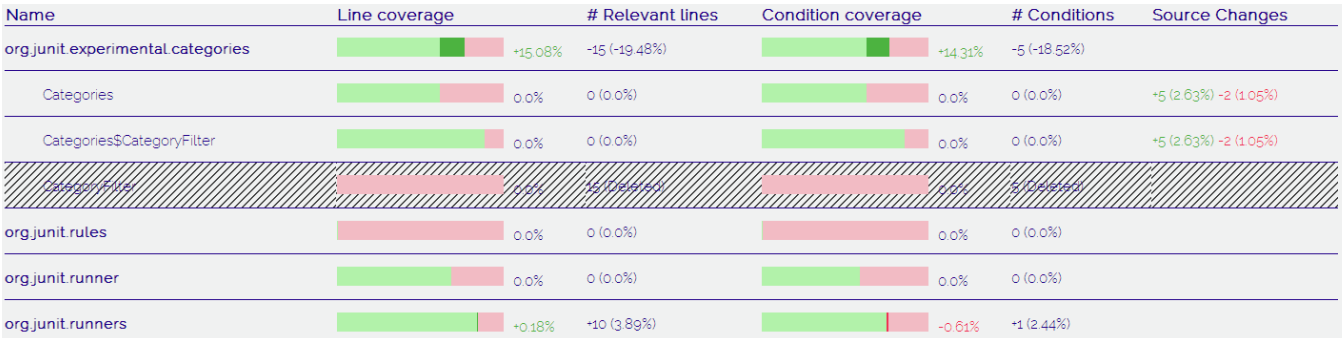

Figure 2: The 'project overview' with package- and class-level information

Figure 3: The 'test view' with data on added/removed test lines

Figure 4: The combined view in the 'class view' report

it must be added to this class. Using OPERIAS, the reviewer can quickly see that the added test code affects neither line coverage nor condition coverage (Figure 7), thus indicating that the new tests are not executed and the absence of changes to the class 'AllTests' from the view.

Although anecdotal, these examples of PRs provide initial evidence on the potential of OPERIAS in supporting the code review process. As a future evaluation, we plan to design and conduct a controlled experiment to measure the causal effects of OPERIAS on the code review process, in particular with respect to the reviewing speed and number of changes suggested by reviewers. Moreover, an observational study can be conducted to see whether the usage of OPERIAS has a relation with a reduced number of further changes needed in code already accepted through PRs. Finally, further work should be conducted to investigate the (potentially distracting) effects that visible code coverage information can have on the effectiveness of reviewers and their behavior.

4. RELATED WORK

Previous research on the pull-based development model has highlighted the importance of tests in pull requests. First, pull requests are merged faster in a well-tested system [10]; then integrators, responsible for merging, indicate that adequate testing is a key quality factor taken into account when deciding whether or not to accept

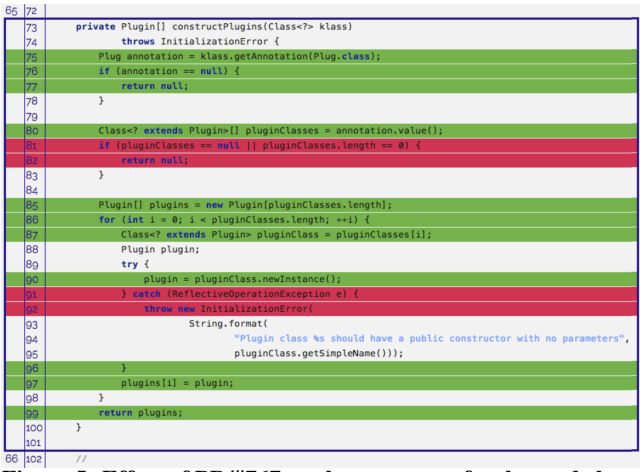

Figure 5: Effect of PR/#767 on the coverage of a changed class.

a change [12] and contributors behave accordingly [11]. Pham et al. discuss the testing culture on GitHub projects, and observe that projects indeed insist on tests in PRs [18].

Although many tools exist to either show differences between two versions of a piece of code or compute test code coverage (e.g., [3]), only a few combine both pieces of information in one view. A promising (yet in early development phase) plugin for Gerrit shows aggregated coverage information [22], but the most popular are: Coveralls.io [14] and SonerQube [20].

Coveralls.io [14] analyzes the report created by Cobertura [3] by comparing the test coverage metrics to a previous report. It shows an overview with detailed coverage information also showing whether test coverage increased or decreased, at the file level. Test coverage information is not integrated in the review process and Coveralls.io does not provide fine-grained information on lines.

SonarQube [20] is an extensive tool to evaluate the quality of a codebase and its changes; it visualizes information on code duplication, coverage, code complexity and more. Particularly, it shows current coverage information of a class and one can filter on selected changes or timeframes, showing lines to cover, branches to cover, uncovered lines and uncovered branches. Nevertheless, SonarQube does not provide any comparison view of test coverage between changes, but only reports on review specific statuses.

5. CONCLUSIONS

We created OPERIAS, a code review support tool that lets reviewers visualize fine-grained test coverage information while reviewing a code contribution. Through real-world examples we gave initial evidence of its potential in different review scenarios.

Name	Line coverage	# Relevant lines	Condition coverage	# Conditions	Source Changes
org.junit.internal.runners.model	-7.41%	0 (0.0%)	0.0%	0 (0.0%)	
EachTestNotifier	-9.52%	0 (0.0%)	0.0%	0 (0.0%)	
org.junit.runners	0.0%	0 (0.0%)	0.0%	0 (0.0%)	
ParentRunner	0.0%	0 (0.0%)	0.0%	0 (0.0%)	+1 (0.22%) -1 (0.22%)
ParentRunner$4	0.0%	0 (0.0%)	0.0%	0 (0.0%)	+1 (0.22%) -1 (0.22%)

Test Classes

Name	Amount of lines changed
/src/test/java/org/junit/tests/running/classes/ParentRunnerTest.java	+117 (78.0%)

Figure 6: Effect of PR/#896 on the coverage of classes, including the *not* changed 'EachTestNotifier' class.

Name	Line coverage	# Relevant lines	Condition coverage	# Conditions	Source Changes
org.junit	-5.3%	+16 (6.56%)	-7.36%	+5 (10.64%)	
Assert	-7.41%	+16 (8.99%)	-13.57%	+5 (16.67%)	+35 (3.72%)

Test Classes

Name	Amount of lines changed
/src/test/java/org/junit/tests/utilityclass/MultipleConstructorUtil.java	11 (New)
/src/test/java/org/junit/tests/utilityclass/NonFinalUtil.java	7 (New)
/src/test/java/org/junit/tests/utilityclass/ProperUtil.java	11 (New)
/src/test/java/org/junit/tests/utilityclass/PublicConstructorUtil.java	12 (New)
/src/test/java/org/junit/tests/utilityclass/UtilityClassTest.java	39 (New)

Figure 7: Effect of PR/#646 on the coverage; since 'AllTests' does not include the new tests, there is no positive change in coverage.

6. REFERENCES

[1] Apache Maven. https://maven.apache.org/.

[2] Bukkit. http://bukkit.org/.

[3] Cobertura. http://cobertura.github.io/cobertura/.

[4] Collaborative code review. https://github.com/features.

[5] Gerrit code review. https://www.gerritcodereview.com.

[6] Junit. http://junit.org/.

[7] Wire. https://github.com/square/wire.

[8] A. Ackerman, L. Buchwald, and F. Lewski. Software inspections: an effective verification process. *Software, IEEE*, 6(3):31–36, 1989.

[9] A. Bacchelli and C. 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.

[10] G. Gousios, M. Pinzger, and A. van Deursen. An exploratory study of the pull-based software development model. In *ICSE*, pages 345–355, 2014.

[11] G. Gousios, M.-A. Storey, and A. Bacchelli. Work practices and challenges in pull-based development: The contributor's perspective. In *Proceedings of the 38th International Conference on Software Engineering*, ICSE '16, pages 285–296. ACM, 2016.

[12] G. Gousios, A. Zaidman, M. Storey, and A. Van Deursen. Work practices and challenges in pull-based development: the integrator's perspective. In *Proceedings International Conference on Software Engineering (ICSE)*, 2015.

[13] A. Guzzi, A. bacchelli, Y. Riche, and A. van Deursen. Supporting developers' coordination in the IDE. In *18th ACM conference on Computer-Supported Cooperative Work and Social Computing*, CSCW 2015, pages 518–532, 2015.

[14] L. H. Industries. Coveralls. https://coveralls.io/.

[15] L. Moonen, A. van Deursen, A. Zaidman, and M. Bruntink. On the interplay between software testing and evolution and its effect on program comprehension. In *Software evolution*, pages 173–202. Springer, 2008.

[16] E. Myers. Ano (nd) difference algorithm and its variations. *Algorithmica*, 1(1-4):251–266, 1986.

[17] S. Oosterwaal. Combining source code and test coverage changes in pull requests. Master's thesis, Delft University of Technology, 2015. http://repository.tudelft.nl/.

[18] R. Pham, L. Singer, O. Liskin, K. Schneider, et al. Creating a shared understanding of testing culture on a social coding site. In *Software Engineering (ICSE), 2013 35th International Conference on*, pages 112–121. IEEE, 2013.

[19] P. Rigby, B. Cleary, F. Painchaud, M. Storey, and D. German. Open source peer review – lessons and recommendations for closed source. *To appear in IEEE Software*, 2012.

[20] S. SA. Sonarqube. http://www.sonarqube.org/.

[21] Y. Tao, Y. Dang, T. Xie, D. Zhang, and S. Kim. How do software engineers understand code changes?: An exploratory study in industry. In *Proceedings of FSE 2012*.

[22] Ullink. Gerrit coverage plugin. https://github.com/Ullink/gerrit-coverage-plugin.