Multi-Perspective Visualization to Assist Code Change Review

Wang Chen
Wuhan University
Contents

1 Motivation
2 Framework
3 Experiment
4 Conclusion
Motivation
Introduction

- Code review becomes a very practical approach in software quality assurance
- How to perform code review in timely manner is considered as a big challenge
Related Work

- Some existing tools help code review. Well-known tools include CodeReviewHub, Gerrit, Codecov, CRITICS and etc.
- These tools focus on extracting the code differences, analyzing code coverage, checking code style, facilitating comments input.

|   |   |@@ -1,5 +1,2 @@
|---|---|---|
| 1 | 1 | def hello():
| 2 | 2 |     return "world"
| 3 |   | -
| 4 |   | -def test_hello():
| 5 |   |     assert hello() == "world"

Code difference
Provided by Codecov
Motivation

• There can be much more information to be presented by a change review tool
• Therefore, we propose a change review tool, namely, MultiViewer, to serve the purposes.
Framework
Overview

MultiViewer will present three useful information, namely effort, risk and impact about change of one commit and visualize these information.

Comprehensive View provided by MultiViewer
Three Metrics

**Effort**
Costs of making the changes

**Risk**
Change’s closeness to bug fixing

**Impact**
Correlation with other components and influence on the entire system
Effort Calculation

Current commit → Contains → Certain code file → Involves → (LOCC) Lines of Changed Code
Risk Calculation

Current commit

Contains

Certain code file

Involved in

History commits with Bug fixing

Severity of Bug fixing

Dev

LOCC

IS
Impact Calculation

Current commit

Contains

Certain code file

Code files of whole system

coop-change happen in

co-change strengthen

History commit

IS
Visualization

• **MultiViewer** provide *Spider Chart* and *Coupling Chart* to help
• The *Spider Chart* provides a preliminary overview of the commit
• *Coupling Chart* focuses on a changeset, visualizing its correlation with other relevant files.
Research Question

• **RQ1**: How changes committed by different groups of developers (authors) distinguish from each other with respect to Effort, Risk and Impact.

• **RQ2**: How different types of commits distinguish from each other with respect to Effort, Risk and Impact.
Experiment Setup

- Project selection: 10 Java projects on GitHub
- Project popularity: Popular and Unpopular

<table>
<thead>
<tr>
<th>Project</th>
<th>Star</th>
<th>Fork</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buck</td>
<td>4696</td>
<td>722</td>
</tr>
<tr>
<td>Druid</td>
<td>6405</td>
<td>3092</td>
</tr>
<tr>
<td>Hadoop</td>
<td>3302</td>
<td>3144</td>
</tr>
<tr>
<td>Realm-Java</td>
<td>7736</td>
<td>1215</td>
</tr>
<tr>
<td>Spring-Framework</td>
<td>14152</td>
<td>10431</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Project</th>
<th>Star</th>
<th>Fork</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jackson-databind</td>
<td>1274</td>
<td>552</td>
</tr>
<tr>
<td>Commons-lang</td>
<td>960</td>
<td>596</td>
</tr>
<tr>
<td>Graphhopper</td>
<td>1198</td>
<td>518</td>
</tr>
<tr>
<td>BIMServer</td>
<td>268</td>
<td>203</td>
</tr>
<tr>
<td>Nutch</td>
<td>1188</td>
<td>850</td>
</tr>
</tbody>
</table>
RQ1

How changes committed by different groups of developers (authors) distinguish from each other with respect to Effort, Risk and Impact.
Author Group

- **Author group.** We divided authors into two groups, namely “Internal” and “External” authors.

<table>
<thead>
<tr>
<th>Project Type</th>
<th>Author Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$G_1$</td>
<td>Popular</td>
</tr>
<tr>
<td>$G_2$</td>
<td>Popular</td>
</tr>
<tr>
<td>$G_3$</td>
<td>Niche</td>
</tr>
<tr>
<td>$G_4$</td>
<td>Niche</td>
</tr>
</tbody>
</table>

All Commit groups ▶
Result of Risk

- Internal authors make commits with lower Risk values than External authors
- In Unpopular projects, the conclusion is opposite

Risk comparison among all Groups of commits
RQ2

How different types of commits distinguish from each other with respect to Effort, Risk and Impact
Commit Type

- **Commit type**: We considered seven major types of commits and used keyword to recognize the type.

<table>
<thead>
<tr>
<th>Name for short</th>
<th>Type name</th>
<th>content</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLN</td>
<td>Cleanup</td>
<td>deleting code</td>
</tr>
<tr>
<td>IMP</td>
<td>Improvement</td>
<td>improving performance</td>
</tr>
<tr>
<td>DOC</td>
<td>JavaDoc</td>
<td>inputing documents</td>
</tr>
<tr>
<td>CONFIG</td>
<td>Configuration</td>
<td>configuring the project</td>
</tr>
<tr>
<td>DEF</td>
<td>Defect</td>
<td>fixing bugs</td>
</tr>
<tr>
<td>FEATURE</td>
<td>Feature</td>
<td>implementing features</td>
</tr>
<tr>
<td>TEST</td>
<td>Test</td>
<td>Testing code</td>
</tr>
</tbody>
</table>
Result of Risk

“Defect” is distinguishable with its high Risk

“JavaDoc” and “Test” involve low risk.
Result of Effort

“Feature” and “Improvement” have high Effort

“Config” and “JavaDoc” commits generally require very low Effort
Conclusion
Conclusion

• We defined metrics for code changes: the change effort, risk and impact.
• We also provided a change review assistance tool for GitHub, namely, MultiViewer to visualize such information
• We demonstrated the helpfulness of MultiViewer by showing its ability as indicators to some important project features
Thanks!
Any question?