Interactive Code Review for Systematic Changes

Tianyi Zhang,¹ Myoungkyu Song,² Joseph Pinedo,² Miryung Kim¹
¹ University of California, Los Angeles  ² University of Texas at Austin
Code Review

What is code review?
  - inspect changes
  - find mistakes overlooked by developers

State-of-art
  - Eclipse Compare, Gerrit, Phabricator, Code Flow
  - line-level differences
  - manual process
Motivation

- Reviewers have a hard time to inspect *systematic edits* — similar changes scattered across the program

```java
int keyDownEvent (int w) {
    ExpandItem item = items [index];
    switch (w) {
        case OS.SPACE:
            Event event = new Event ();
            event.item = item;
            sendEvent(true, event);
            event.item = focusItem;
            sendEvent(event);
            refreshItem(focusItem);
        
(a) change example
```
Motivation

- Code reviewers cannot easily answer questions like:
  - What other locations are changed similarly to this change?
  - Are there inconsistencies among similar edits?
  - Are there any other locations that are similar to this code but are not updated?
Outline

• Related Work

• Interactive Code Review Approach
  o Phase I: Context-Aware Change Template Generation
  o Phase II: Template Customization
  o Phase III: Change Summarization and Anomaly Detection

• Evaluation
  o Semi-Structured Interviews with Salesforce Engineers
  o A User Study with 12 ECE students at UT Austin

• Conclusion
Related Work

- Modern Code Review and Change Comprehension
  - Decompose large, composite changes into small ones [Rigby et al., Tao et al.]
  - Our work is inspired by these findings.

- Code Clone Analysis
  - Detect duplicated code and find cloning-related bugs [CCFinder, Deckard, CP-Miner, SecureSync]
  - But they are not designed to investigate diff patches.

- Systematic Change Automation
  - Automate similar changes to multiple locations [LASE, Sydit]
  - LASE uses fixed template generation and does not allow interactive customization.
  - LASE is not evaluated with user studies.
Critics: Interactive Code Review Approach for Systematic Changes
int keyDownEvent (int w) {
   ExpandItem item = items [index];
   switch (w) {
      case OS.SPACE:
         Event ev = new Event ();
         ev.item = item;
         sendEvent(true, ev);
         ev.item = focusItem;
         sendEvent(ev);
         refreshItem(focusItem);
   }
}

(a) selected change

(b) abstract change template
Phase II: Template Customization

Event $v1 = \new ..$

.. $v1.item = item$

$exclude$

method_decl

$exclude$

.. $item = items[..]$ $exclude (c) \text{ customized change template}$

sendEvent(true, $v1$)

sendEvent($v1$, focusItem)

refreshItem (focusItem)
### Phase III: Change Summarization and Anomaly Detection

- Critics searches for similar locations in the old revision and the new revision respectively.

<table>
<thead>
<tr>
<th></th>
<th>Match Old</th>
<th>Not Match Old</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Match New</strong></td>
<td>Correct similar change</td>
<td>Similar change to different contexts</td>
</tr>
<tr>
<td><strong>Not Match New</strong></td>
<td>Missing similar change</td>
<td>Irrelevant</td>
</tr>
</tbody>
</table>
Phase III: Change Summarization and Anomaly Detection
Phase III: Change Summarization and Anomaly Detection

- Original RTED algorithm\(^1\) computes node-level alignment between two trees.

- Critics extended RTED in two ways.
  - match a parameterized token with any concrete token.
  - match an excluded node with any node.

Critics Plug-in

Eclipse plug-in are available at https://sites.google.com/a/utexas.edu/critics/. (Zhang et al. FSE 14' Demo)
Research Questions

- RQ1: How critics could be used in practice?
- RQ2: How accurately does a reviewer locate similar edits and mistakes with Critics?
- RQ3: How much time can a reviewer save by using Critics?
## Semi-Structured Interview at Salesforce

<table>
<thead>
<tr>
<th>Subject</th>
<th>Role</th>
<th>Gender</th>
<th>Age</th>
<th>Java Experience</th>
<th>Code Review Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Developer</td>
<td>Male</td>
<td>21-30</td>
<td>4</td>
<td>Weekly</td>
</tr>
<tr>
<td>2</td>
<td>QE</td>
<td>Female</td>
<td>21-30</td>
<td>3</td>
<td>Weekly</td>
</tr>
<tr>
<td>3</td>
<td>Manager</td>
<td>Male</td>
<td>41-50</td>
<td>4</td>
<td>Seldom</td>
</tr>
<tr>
<td>4</td>
<td>QE</td>
<td>Male</td>
<td>31-40</td>
<td>5</td>
<td>Weekly</td>
</tr>
<tr>
<td>5</td>
<td>QE</td>
<td>Female</td>
<td>31-40</td>
<td>10</td>
<td>Weekly</td>
</tr>
<tr>
<td>6</td>
<td>Developer</td>
<td>Male</td>
<td>41-50</td>
<td>14</td>
<td>Daily</td>
</tr>
</tbody>
</table>
Semi-Structured Interview at Salesforce

- 20-minute tutorial about how to use Critics
- A hands-on trial of Critics\(^1\) with one of four diff patches authored by their own team.

<table>
<thead>
<tr>
<th>No.</th>
<th>Patch Description</th>
<th>Changed LOC</th>
<th>Num of Changed Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Refactor test cases by moving bean maps to respective utils classes</td>
<td>743</td>
<td>22</td>
</tr>
<tr>
<td>2</td>
<td>Refactoring the API to get versioned field values</td>
<td>943</td>
<td>34</td>
</tr>
<tr>
<td>3</td>
<td>Refactor test cases to use try-with-resources statements</td>
<td>484</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>Update common search tests by getting versioned test data</td>
<td>2224</td>
<td>12</td>
</tr>
</tbody>
</table>

1. Critics is implemented as an Eclipse plug-in, [http://sites.google.com/utexas/edu/critics/](http://sites.google.com/utexas/edu/critics/)
How could Critics help them with code reviews?

- “... REST APIs across different versions generally share similar code snippets ... It's hard and time-consuming to find mistakes on similar changes on those locations...”

- “The feature in your tool can free us from piling code review tasks on our senior developers...”
How do they like or dislike Critics?

• “Currently COLLABORATOR only highlights the changed location in a very naive way. A feature like extracting and visualizing the change context can help us better understand the change itself as well as find some underlying change patterns between related changes.”

• “It will be helpful if Critics can provide some hints about template customization.”
User Study at UT Austin

- We recruited 12 UT Austin students
  - 4 of them are ECE undergrads, the others are graduate students in Software Engineering
  - All of them have at least one year experience of Eclipse IDE
  - All but one have code review experience using diff tools such as Eclipse Compare and SVN/Git diff.

- We gave them a 20-minute tutorial on how to use Critics plug-in
# Code Review Patches

<table>
<thead>
<tr>
<th>Patch</th>
<th>Version</th>
<th>Change Description</th>
<th>Similar Change</th>
<th>Inconsistent Change</th>
<th>Missing Update</th>
<th>Size(LOC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>JDT 9800 vs JDT 9801</td>
<td>Initiate a variable in a for loop instead of using a hashmap</td>
<td>getTrailingComments(ASTNode) getLeadingComments(ASTNode) getExtendedEnd(ASTNode)</td>
<td>getExtendedStartPosition(ASTNode)</td>
<td>getComments(ANode) getCommentsRange(ASTNode)</td>
<td>190</td>
</tr>
<tr>
<td>2</td>
<td>JDT 16010 vs JDT 10611</td>
<td>extract the logic of unicode traitement to a method</td>
<td>getNextChar() getNextCharAsDigit() getNextToken()</td>
<td>getNextCharAsJavaIdentifierPart(ASTNode)</td>
<td>jumpOverMethodBody() ... 11 locations in total</td>
<td>680</td>
</tr>
</tbody>
</table>
User Study Tasks

• Each participant carried out code review tasks on two different patches, one with Critics and the other with Eclipse Compare
  ○ Q1: Given the change in the method `getTrailingComments`, what other methods containing similar changes can you find?
  ○ Q2: Which of the following methods contains inconsistent changes compared with the change in `getTrailingMethods`?
  ○ Q3: How many methods share context similar to the change in `getTrailingMethods` but have missed updates?

• We measured task completion time and accuracy.
<table>
<thead>
<tr>
<th>Subjects</th>
<th>Critics</th>
<th></th>
<th>Eclipse Compare</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Q1</td>
<td>Q2</td>
<td>Q3</td>
<td>Time</td>
</tr>
<tr>
<td>1</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>13:30</td>
</tr>
<tr>
<td>2</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>13:18</td>
</tr>
<tr>
<td>3</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>18:29</td>
</tr>
<tr>
<td>4</td>
<td>✗</td>
<td>√</td>
<td>✗</td>
<td>15:02</td>
</tr>
<tr>
<td>5</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>19:00</td>
</tr>
<tr>
<td>6</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>23:00</td>
</tr>
<tr>
<td>7</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>11:48</td>
</tr>
<tr>
<td>8</td>
<td>✓</td>
<td>√</td>
<td>✓</td>
<td>20:00</td>
</tr>
<tr>
<td>9</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>29:00</td>
</tr>
<tr>
<td>10</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>16:11</td>
</tr>
<tr>
<td>11</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>14:27</td>
</tr>
<tr>
<td>12</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>35:17</td>
</tr>
</tbody>
</table>

**Average**

83% 100% 92% 19:26 42% 58% 33% 25:39

Human subjects can answer questions about systematic changes 47.3% more correctly with 31.9% saving in time using Critics.
Comparison with LASE

- LASE automates systematic editing by searching for locations and applying edits to individual locations. [Meng et al.]
- It is challenging to directly compare LASE and Critics:
  - fixed vs. interactive template generation
- Simulate observed template customization patterns
- Compare the locations found by the two techniques
Comparison with LASE

• In five out of six cases, Critics achieves the same or higher accuracy than LASE within a few iterations.

<table>
<thead>
<tr>
<th></th>
<th>Critics</th>
<th></th>
<th></th>
<th></th>
<th>LASE</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Precision</td>
<td>Recall</td>
<td>Iterations</td>
<td>Time(sec)</td>
<td>Precision</td>
<td>Recall</td>
<td></td>
</tr>
<tr>
<td>Patch 1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>1.66</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Patch 2</td>
<td>1</td>
<td>0.9</td>
<td>6</td>
<td>8.95</td>
<td>0.92</td>
<td>0.75</td>
<td></td>
</tr>
<tr>
<td>Patch 3</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>13.52</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Patch 4</td>
<td>1</td>
<td>1</td>
<td>7</td>
<td>71.98</td>
<td>1</td>
<td>0.33</td>
<td></td>
</tr>
<tr>
<td>Patch 5</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>6.86</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Patch 6</td>
<td>1</td>
<td>0.33</td>
<td>3</td>
<td>1.47</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td>1</td>
<td>0.87</td>
<td>4</td>
<td>17.41</td>
<td>0.99</td>
<td>0.84</td>
<td></td>
</tr>
</tbody>
</table>
Conclusion

- We present Critics, a novel interactive code review approach for searching systematic changes and detecting potential mistakes.
- A study at Salesforces show that Critics scales to an industry-scale project and can be easily adopted by professional developers.
- Human subjects using Critics can answer questions about systematic changes more correctly within less time, in the comparison of the baseline use of Eclipse Compare.
Q&A
Accuracy variation in Critics’s Simulation

(a) F1 score for finding similar edit locations by excluding statements.

(b) F1 score for finding similar edit locations by parameterizing identifiers.
Subjects and Metrics

• Six patches drawn from Eclipse JDT and SWT [Meng et al.]
  o Patch size ranges from 190 to 680 lines of changed code
  o Consisted of three to ten systematic edits

• Metrics
  o precision
  o recall
  o $F_1$ score