FaultTracer: A Change Impact and Regression Fault Analysis Tool for Evolving Java Programs

Lingming Zhang, Miryung Kim, Sarfraz Khurshid
University of Texas at Austin
zhanglm@utexas.edu

FSE Formal Research Demo Track
Nov 14, 2012
Scenario

Old Version P

Test

Regression tests T
Scenario

Old Version P → Evolve → New Version P'

Regression tests T

Test
Scenario

Old Version $P$

Evolve

New Version $P'$

Test

Regression tests $T$

• **Q1:** How to efficiently run the regression tests?
  • Which tests are relevant to program edits?
• **Q2:** How to effectively localize faults when tests fail?
  • Which program edits are relevant to test failures?
Motivation

• **Chianti change impact analysis** is effective at finding suspicious edits but does not rank these edits. [Ren’04, Ren’06]

• **Spectrum-based fault localization** ranks potential faulty code fragments but does not focus on changes. [Jones’02, Abreu’07, Yu’08, Santelices’09, Parnin’11]

• **Our insight** is to combine change-impact analysis and spectrum-based fault localization [ICSM11].
  
  • Identify suspicious edits based on extended call graphs.
  
  • Rank suspicious edits using dynamic program spectrum information.
FaultTracer overview

Old program version: \( P \)

Detect changes and dependences

New program version: \( P' \)

Program edits: \( \Delta \)
FaultTracer: A Change Impact and Regression Fault Analysis Tool for Evolving Java Programs

FaultTracer overview

Old program version: $P$

New program version: $P'$

Detect changes and dependences

Program edits: $\Delta$

Original test suite: $T$

Select affected tests based on ECG analysis

Affected tests: $T'$
FaultTracer overview

Old program version: $P$

New program version: $P'$

Detect changes and dependences

Program edits: $\Delta$

Original test suite: $T$

Select affected tests based on ECG analysis

Failed test: $t$

Affected tests: $T'$

Determine suspicious edits based on ECG analysis

Subset of suspicious edits: $\delta_t$
FaultTracer overview

1. Detect changes and dependences:
   - Old program version: \( P \)
   - New program version: \( P' \)

2. Select affected tests based on ECG analysis:
   - Original test suite: \( T \)
   - Program edits: \( \Delta \)
   - Failed test: \( t \)
   - Affected tests: \( T' \)

3. Determine suspicious edits based on ECG analysis:
   - Subset of suspicious edits: \( \delta_t \)

4. Rank suspicious edits based on program spectrum information:
   - Ranked list of suspicious edits: \( \delta_t' \)
Extended call graph (ECG)

Traditional Call Graphs used by Chianti [Ren’04]

- test1
  - A.Clinit()
  - A.bar()
  - C.C()
  - C.foo()
  - B.B()

- test2
  - <C, C.foo()>

Extended Call Graphs used by FaultTracer

- test1
  - A.Clinit()
  - A.bar()
  - C.C()
  - C.foo()
  - <SFW, A.f2>
  - B.f1

- test2
  - <FR, C.f1>
  - B.f1

FaultTracer: A Change Impact and Regression Fault Analysis Tool for Evolving Java Programs
Step 1: Detect atomic changes & dependences

<table>
<thead>
<tr>
<th>Change types</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CM</td>
<td>Change method</td>
</tr>
<tr>
<td>AM</td>
<td>Add method</td>
</tr>
<tr>
<td>DM</td>
<td>Delete method</td>
</tr>
<tr>
<td>AF</td>
<td>Add field</td>
</tr>
<tr>
<td>DF</td>
<td>Delete field</td>
</tr>
<tr>
<td>CFI</td>
<td>Change instance field</td>
</tr>
<tr>
<td>CSFI</td>
<td>Change static field</td>
</tr>
<tr>
<td>LCM</td>
<td>Method look-up change</td>
</tr>
<tr>
<td>LCF</td>
<td>Field look-up change</td>
</tr>
</tbody>
</table>

- Dependence inference illustration
  - For every AM change, if a method called by the added method is new \textit{and all methods overridden by it are also new}, the caller should be dependent on the added callee.
Step 2: Select tests based on ECG analysis

- FaultTracer directly matches all non-look-up changes with ECGs of the old version to select affected tests.
  - Existing technique needs to transform field changes into constructor change first.
- FaultTracer identifies tests that are influenced by method or field look-up changes as affected tests.
  - Existing technique does not handle field look-up change.
Step 3: Identify suspicious edits based on ECG analysis

- FaultTracer directly identifies all non-look-up changes on ECGs of the new version as suspicious edits.
  - Existing technique needs to select
    - the changes covered by affected tests.
    - the changes that these covered changes transitively depend on.
  - FaultTracer identifies method or field level edits that caused look-up changes on ECGs as suspicious edits.
    - Existing technique cannot find field level edits that caused field look-up changes.
Step 4: Localize failure-inducing program edits using test spectra

- **Relation between suspicious edits and tests**

<table>
<thead>
<tr>
<th>Edits</th>
<th>Test1</th>
<th>Test2</th>
<th>test3</th>
<th>test4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edit1</td>
<td>✗</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Edit2</td>
<td></td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Edit3</td>
<td></td>
<td></td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Edit4</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Result</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
<td>Fail</td>
</tr>
</tbody>
</table>

- **Suspicious score calculation**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Edit1</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Edit2</td>
<td>0.75</td>
<td>0.50</td>
<td>0.50</td>
<td>0.71</td>
</tr>
<tr>
<td>Edit3</td>
<td>0.75</td>
<td>0.50</td>
<td>0.50</td>
<td>0.71</td>
</tr>
<tr>
<td>Edit4</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
</tbody>
</table>
Demo: Step 1.
- Detect changes and dependences

Old program version: $P$

New program version: $P'$

Detect changes and dependences

Program edits: $\Delta$

Failed test: $t$

Original test suite: $T$

Select affected tests based on ECG analysis

Affected tests: $T'$

Determine suspicious edits based on ECG analysis

Subset of suspicious edits: $\delta_t$

Rank suspicious edits based on program spectrum information

Ranked list of suspicious edits: $\delta'_t$
Demo: Step 2.
- Select affected tests

1. Detect changes and dependencies
   - Old program version: $P$
   - New program version: $P'$

2. Select affected tests based on ECG analysis
   - Original test suite: $T$
   - Affected tests: $T'$

3. Determine suspicious edits based on ECG analysis
   - Program edits: $\Delta$
   - Failed test: $t$
   - Subset of suspicious edits: $\delta_t$

4. Rank suspicious edits based on program spectrum information
   - Ranked list of suspicious edits: $\delta_t'$

FaultTracer: A Change Impact and Regression Fault Analysis Tool for Evolving Java Programs
Demo: Step 3.
- Determine suspicious edits

1. Detect changes and dependences
2. Select affected tests based on ECG analysis
3. Determine suspicious edits based on ECG analysis
4. Rank suspicious edits based on program spectrum information

FaultTracer: A Change Impact and Regression Fault Analysis Tool for Evolving Java Programs
Demo: Step 4.
- Rank suspicious edits

1. Detect changes and dependences

2. Select affected tests based on ECG analysis

3. Determine suspicious edits based on ECG analysis

4. Rank suspicious edits based on program spectrum information

Old program version: $P$

New program version: $P'$

Program edits: $\Delta$

Failed test: $t$

Original test suite: $T$

Affected tests: $T'$

Subset of suspicious edits: $\delta_t$

Ranked list of suspicious edits: $\delta_t'$

FaultTracer: A Change Impact and Regression Fault Analysis Tool for Evolving Java Programs
We have successfully applied FaultTracer to real-world Java programs ranging 1.83~80.44 KLoC

- jtopas, xml-security, jmeter, and ant.
- Runtime overhead by Chianti and FaultTracer in collecting call graph information (ms).
Conclusion

• FaultTracer combines a Chianti-style change impact analysis with spectrum-based fault localization.
• FaultTracer improves a Chianti-style change impact analysis based on extended call graph analysis.
• Experimental results show that FaultTracer [ICSM11]
  • outperforms Chianti in determining affecting changes by 20%.
  • outperforms existing technique for localizing failure-inducing program edits by 50%.
• FaultTracer Eclipse plug-in is available for public download:
  https://webspace.utexas.edu/lz3548/www/ftracer.html