Faster Bug Detection for Software Product Lines with Incomplete Feature Models

Sabrina Souto
Federal University of Pernambuco
Recife, PE, Brazil

Darko Marinov
University of Illinois Urbana, IL, USA

Divya Gopinath
University of Texas Austin, TX, USA

Sarfranz Khurshid
University of Texas Austin, TX, USA

Marcelo d’Amorim
Federal University of Pernambuco
Recife, PE, Brazil

Don Batory
University of Texas Austin, TX, USA

Acknowledgements: US NSF CCF- 0845628, CCF-1012759, CCF-1212683, CCF-1319688, CCF- 1439957

SPLC 2015
Nashville, TN
July 23, 2015

Acknowledgements: Brazil FACEPE BPG-0675-1.03/09 CNPq 457756/2014-4
Context: Software Product Lines

Commonality

Variability

Feature selection
Our Research Background

• **Mostly software testing**
  - Generate new tests to find bugs
  - Run existing tests faster/better

• **Currently dominant approach**
  - Test real code (ideally from open source)
  - May use additional code artifacts (ideally real tests or comments, sometimes academic specs or more)
  - Find real bugs
General Terminology

• **Features**
  • Functionalities of software systems

• **A Software Product Line – SPL**
  • Is a family of programs
  • Each program is defined by a unique combination of features

• **Configurations**
  • Selection of features

• **Feature Model – FM**
  • Defines a set of consistent configurations
  • *Not always documented*
Problem:
Testing SPLs with Incomplete Feature Model

Our Solution:
-- SPLif --
Problem
Testing SPLs with Incomplete Feature Model

Features
M, T, W

Configurations
{} → M → T → W → MT → MW → WT → MTW

Tests
T₁, T₂, T₃, T₄, ..., Tₙ
Problem
Testing SPLs with Incomplete Feature Model
Problem
Testing SPLs with Incomplete Feature Model

Features
- M
- T
- W

Configurations
- {} → T
- M → MT
- T → T
- W → MW
- MT → MT
- MW → MW
- WT → WT
- MTW → MTW

Tests
- T₁ → ✔️
- T₂ → ✗
- T₃ → ✔️
- T₄ → ✔️
- ... → ...
- Tₙ → ✗
The FM is essential to distinguish the causes for test failures, because it can detect (in)consistent configurations.

Possible causes of failures:
1. Inconsistent configurations
2. Test too restrictive
3. Bug in code
Problem

Testing SPLs with Incomplete Feature Models

Possible causes of failures:
1. Inconsistent configurations
2. Test too restrictive
3. Bug in code

The FM is essential to distinguish the causes for test failures, because it can detect (in)consistent configurations.

FMIs are not always available!
Problem Summary

- Feature models play a key role in testing SPLs
  - Constrain the space of configurations to test
  - Enable accurate categorization of failing tests

- Most prior work on testing SPLs assumes the availability of a complete feature model

- In practice, FMs are not always available
  - How to reduce the number of configurations per tests to run?
  - How to discover the causes for test failures?

False positives!
A test can fail due to a configuration that is not in the (absent/incomplete) model.
Related Work

• **SPL Testing**
  [Qu et al. ISSTA’08] [Cabral et al. SPLC’10] [Uzuncaova et al. TSE’10]
  [Garvin et al. ISSRE’11] [Kim et al. AOSD’11][Kastner et al. FOSD’12]
  [Kim et al. ISSRE’12] [Shi et al. FASE’12] [Song et al. ICSE’12]
  [Apel et al. ICSE’13] [Kim et al. FSE’13]

• **FM Extraction and Inference**
  [Czarnecki and Wasowski, SPLC’07] [Alves et al. SPLC’08] [Weston et al. SPLC’09]
  [Rabkin et al. ICSE’11] [She et al. ICSE’11] [Acher et al. VaMos’12]
  [Lopez-Herrejon et al. SSBSE’12] [Haslinger et al. FASE’13]
  [Davril et al. FSE’13] [Xu et al. SOSP’13]

• **Fault Localization**
  [Jones et al. ICSE’02] [Dallmeier et al. ECOOP’05] [Abreu et al. PRDC’06]
  [Abreu et al. TAIC’07] [Qu et al. ISSTA’08] [Renieris et al. ISSTA’08]
  [Abreu et al. ASE’09]

• **Configuration Troubleshooting**
  [Garvin et al. ASAS’12] [Zhang and Ernst et al. ICSE’13]
  [Zhang and Ernst et al. ICSE’14] [Swanson et al. FSE’14]
Related Work

• SPL Testing
  [Qu et al. ISSTA’08] [Cabral et al. SPLC’10] [Uzuncaova et al. TSE’10]
  [Garvin et al. ISSRE’11] [Kim et al. AOSD’11][Kastner et al. FOSD’12]
  [Kim et al. ISSRE’12] [Shi et al. FASE’12] [Song et al. ICSE’12]
  [Apel et al. ICSE’13] [Kim et al. FSE’13]

• FM Extraction and Inference
  [Czarnecki and Wasowski, SPLC’07] [Alves et al. SPLC’08] [Weston et al. SPLC’09]
  [Rabkin et al. ICSE’11] [She et al. ICSE’11] [Acher et al. VaMos’12]
  [Lopez-Herrejon et al. SSBSE’12] [Haslinger et al. FASE’13]
  [Davril et al. FSE’13] [Xu et al. SOSP’13]

• Fault Localization
  [Jones et al. ICSE’02] [Dallmeier et al. ECOOP’05] [Abreu et al. PRDC’06]
  [Abreu et al. TAIC’07] [Qu et al. ISSTA’08] [Renieris et al. ISSTA’08]
  [Abreu et al. ASE’09]

• Configuration Troubleshooting
  [Garvin et al. ASAS’12] [Zhang and Ernst et al. ICSE’13]
  [Zhang and Ernst et al. ICSE’14] [Swanson et al. FSE’14]

No prior work combines FM inference with tests and their executions
Insight

• Tests that fail on consistent configurations indicate real faults

• We need to find fault-revealing consistent configurations soon
  • Enable efficient bug detection

• The FM is not available or is incomplete
  • Do not need to discover the entire FM
  • Discover only the relevant part to check the consistency of the fault-revealing configuration

• Assumption
  • The developer/user will help to check such consistency
  • The developer/user is aware about many feature relationships
Proposal: SPLif

Source code

Tests

Feature Model (optional)

SPLat

Run the tests exploring all reachable configurations

List of passing/failing tests for all configurations

SPLif

Mc contains t.fail

\[ S_t = \frac{F_t}{F_t + P_t} \]

t.fail

Pick the top test

Rank tests

Pick the top configuration

Repair the test or code

Increment the FM

Consistent

Label configuration

Inconsistent

#?(c) t.Fail (Mc & fc).isSAT

Rank failing configurations

Test or code
Specific Terminology

• Each feature can assume 3 values:
  • 0: the feature is disabled (=false)
  • 1: the feature is enabled (=true)
  • ?: the feature has no value yet (=unknown)

• Incomplete vs. Complete Configuration

<table>
<thead>
<tr>
<th>MTW=0?1  (incomplete)</th>
<th>Notepad Features: Menubar, Toolbar, and Wordcount</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTW=010  (complete)</td>
<td></td>
</tr>
</tbody>
</table>

• Consistent vs. Inconsistent Configuration

<table>
<thead>
<tr>
<th>MTW=0?1  (consistent)</th>
<th>Notepad Constraint: M ∨ T (Initially Undocumented )</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTW=00?  (inconsistent)</td>
<td></td>
</tr>
</tbody>
</table>
SPLif on Notepad (1 test)

• Configurations (MTW):

111
011
110
010
10?
00?
SPLif Notepad (1 test)

• Configurations (MTW):

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
<td>✓</td>
</tr>
<tr>
<td>011</td>
<td>✗</td>
</tr>
<tr>
<td>110</td>
<td></td>
</tr>
<tr>
<td>010</td>
<td></td>
</tr>
<tr>
<td>10?</td>
<td>✗</td>
</tr>
<tr>
<td>00?</td>
<td>✗</td>
</tr>
</tbody>
</table>

Execution of some tests fails!
SPLif Notepad (1 test)

- Configurations (MTW):

```
Select failing configurations
```

011 ✗

10? ✗

00? ✗
SPLif Notepad (1 test)

- Configurations (MTW):

<table>
<thead>
<tr>
<th>Rank</th>
<th>configurations for inspection</th>
</tr>
</thead>
<tbody>
<tr>
<td>00?</td>
<td></td>
</tr>
<tr>
<td>10?</td>
<td></td>
</tr>
<tr>
<td>011</td>
<td></td>
</tr>
</tbody>
</table>
SPLif Notepad (1 test)

• Configurations (MTW):

00?  Inconsistent!
10?
011
SPLif Notepad (1 test)

• Configurations (MTW):

Inconsistent!

00?

10?

011

Partial Feature Model (PFM) = !(U c_i), where c_i is an inconsistent configuration

In this case c_i=(!M ∧ !T) and PFM=

!(!(M ∧ !T))
!!M ∨ !!T
M ∨ T
SPLIf Notepad (1 test)

- Configurations (MTW):

00?
10?
011

Partial Feature Model (PFM) = !(U c_i), where c_i is an inconsistent configuration.

In this case, c_i = (!M \land \land \land \land !T) and PFM = !(!M \land \land \land \land !T) !!M \lor \lor \lor !!T
M \lor T

Configurations that violate this constraint will not be inspected!
SPLif Notepad (1 test)

• Configurations (MTW):

Partial Feature Model:

\[ M \lor T \]

00?

10?

011

Consistent

The test failed on configurations where no inconsistency has been observed. User should inspect test and/or code!
Evaluation: Setup

• **Questions**
  • **RQ1**: How well does SPLif rank faulty tests for inspection?
  • **RQ2**: How well does SPLif rank configurations (of selected tests) for inspection?

• **Experiment**
  • 5 SPLs previously used
  • The tests used were created by students
  • 4 techniques:
    • Random
    • Memory
    • Weighted
    • Adaptive
Evaluation: Results

Ranking Tests

**RQ1:** How well does SPLif rank faulty tests for inspection?
**Evaluation: Results**

**Ranking Configurations**

**RQ2:** How well does SPLif rank configurations (of selected tests) for inspection?

<table>
<thead>
<tr>
<th>Mode</th>
<th>Companies</th>
<th>GPL</th>
<th>Notepad</th>
<th>Desktop Searcher</th>
<th>ZipMe</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>146</td>
<td>257</td>
<td>90</td>
<td>44</td>
<td>269</td>
</tr>
<tr>
<td>UpdateFM</td>
<td>69</td>
<td>211</td>
<td>40</td>
<td>30</td>
<td>45</td>
</tr>
<tr>
<td>Weighted and Adaptive</td>
<td>69</td>
<td>223</td>
<td>10</td>
<td>34</td>
<td>49</td>
</tr>
</tbody>
</table>
Case Study: GCC

- **RQ3**: How well does SPLif scale to real code?

  - **Experiment**
    - Applied SPLif against the GNU Compiler Collection
      - 27 years of work from 500+ contributors
      - 7+ Million LOCs
      - 17K+ tests
      - More than 2k configuration variables (not only boolean)
GCC Evaluation: Setup

• Tests
  • 4,108 tests from 3 suites (gcc-dg, dg-torture, tree-ssa)
  • 50 configurations per test
  • Randomized SPLat execution to sample different (reachable) configurations

• Options
  • 40 most frequently cited options in the GCC bug reports
  • Initial model (incomplete) built on the work of [Garvin et al. ASAS’13]

• Failures
  • Inspection of failures on crashes
GCC Evaluation: Results

• Recall
  • We focused only on crash failures
  • We ran each test against 50 reachable configurations

• 4,108 tests analyzed
  • 497 tests failed (due to crash or not)
  • 3,986 pairs of tests and configurations failed (due to crash or not)

• Considering only crashes
  • 43 tests manifested crashes in 268 pairs of test and configurations
GCC Evaluation: Results

RQ3: How well does SPLif scale to real code?

Ranking Tests

<table>
<thead>
<tr>
<th>R</th>
<th>$e_1$</th>
<th>$S$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>2</td>
<td>0.04</td>
<td>0.04</td>
</tr>
<tr>
<td>3</td>
<td>0.51</td>
<td>0.51</td>
</tr>
<tr>
<td>4</td>
<td>0.50</td>
<td>0.50</td>
</tr>
<tr>
<td>5</td>
<td>0.46</td>
<td>0.46</td>
</tr>
<tr>
<td>6</td>
<td>0.44</td>
<td>0.44</td>
</tr>
<tr>
<td>7</td>
<td>0.43</td>
<td>0.43</td>
</tr>
<tr>
<td>8</td>
<td>0.42</td>
<td>0.42</td>
</tr>
<tr>
<td>9</td>
<td>0.40</td>
<td>0.40</td>
</tr>
<tr>
<td>10</td>
<td>0.36</td>
<td>0.36</td>
</tr>
<tr>
<td>11</td>
<td>0.36</td>
<td>0.36</td>
</tr>
<tr>
<td>12</td>
<td>0.34</td>
<td>0.34</td>
</tr>
<tr>
<td>13</td>
<td>0.32</td>
<td>0.32</td>
</tr>
<tr>
<td>14</td>
<td>0.32</td>
<td>0.32</td>
</tr>
<tr>
<td>15</td>
<td>0.32</td>
<td>0.32</td>
</tr>
<tr>
<td>16</td>
<td>0.32</td>
<td>0.32</td>
</tr>
<tr>
<td>17</td>
<td>0.31</td>
<td>0.31</td>
</tr>
<tr>
<td>18</td>
<td>0.29</td>
<td>0.29</td>
</tr>
<tr>
<td>19</td>
<td>0.28</td>
<td>0.28</td>
</tr>
<tr>
<td>20</td>
<td>0.25</td>
<td>0.25</td>
</tr>
<tr>
<td>21</td>
<td>0.25</td>
<td>0.25</td>
</tr>
<tr>
<td>22</td>
<td>0.28</td>
<td>0.28</td>
</tr>
<tr>
<td>23</td>
<td>0.27</td>
<td>0.27</td>
</tr>
<tr>
<td>24</td>
<td>0.26</td>
<td>0.26</td>
</tr>
<tr>
<td>25</td>
<td>0.26</td>
<td>0.26</td>
</tr>
<tr>
<td>26</td>
<td>0.26</td>
<td>0.26</td>
</tr>
<tr>
<td>27</td>
<td>0.25</td>
<td>0.25</td>
</tr>
<tr>
<td>28</td>
<td>0.24</td>
<td>0.24</td>
</tr>
<tr>
<td>29</td>
<td>0.24</td>
<td>0.24</td>
</tr>
<tr>
<td>30</td>
<td>0.24</td>
<td>0.24</td>
</tr>
<tr>
<td>31</td>
<td>0.23</td>
<td>0.23</td>
</tr>
<tr>
<td>32</td>
<td>0.20</td>
<td>0.20</td>
</tr>
<tr>
<td>33</td>
<td>0.20</td>
<td>0.20</td>
</tr>
<tr>
<td>34</td>
<td>0.20</td>
<td>0.20</td>
</tr>
<tr>
<td>35</td>
<td>0.19</td>
<td>0.19</td>
</tr>
<tr>
<td>36</td>
<td>0.18</td>
<td>0.18</td>
</tr>
<tr>
<td>37</td>
<td>0.18</td>
<td>0.18</td>
</tr>
<tr>
<td>38</td>
<td>0.14</td>
<td>0.14</td>
</tr>
<tr>
<td>39</td>
<td>0.14</td>
<td>0.14</td>
</tr>
<tr>
<td>40</td>
<td>0.13</td>
<td>0.13</td>
</tr>
<tr>
<td>41</td>
<td>0.12</td>
<td>0.12</td>
</tr>
<tr>
<td>42</td>
<td>0.04</td>
<td>0.04</td>
</tr>
<tr>
<td>43</td>
<td>0.02</td>
<td>0.02</td>
</tr>
</tbody>
</table>

Ranking Configurations

- Random
- UpdateFM
- Weighted
- Adaptive
GCC Evaluation: Results

RQ3: How well does SPLif scale to real code?

New bugs found

<table>
<thead>
<tr>
<th>Cluster data</th>
<th>#Tests</th>
<th>#Pairs</th>
<th>Id</th>
<th>Confirmed</th>
<th>Fixed</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>compute_affine_dependence, tree-data-ref.c: 4233</td>
<td>34</td>
<td>223</td>
<td>61980</td>
<td>Aug.1,2014</td>
<td>-</td>
<td>NEW</td>
</tr>
<tr>
<td>int_cst_value, tree.c: 10625</td>
<td>4</td>
<td>34</td>
<td>62069</td>
<td>Aug.8,2014</td>
<td>-</td>
<td>NEW</td>
</tr>
<tr>
<td>verify_ssa failed, tree-ssa.c: 1056</td>
<td>1</td>
<td>6</td>
<td>62070</td>
<td>Aug.8,2014</td>
<td>Aug.11,2014</td>
<td>RESOLVED FIXED</td>
</tr>
<tr>
<td>Segmentation fault: 11</td>
<td>1</td>
<td>1</td>
<td>62141</td>
<td>Aug.14,2014</td>
<td>Nov.19,2014</td>
<td>RESOLVED FIXED</td>
</tr>
</tbody>
</table>

Recently the first reported bug has been also fixed.
Conclusions

• The FM can detect (in)consistent configurations
  • It is essential to distinguish the causes for test failures

• Prior research assumes that SPLs come equipped with complete, formally specified FMs
  • This assumption does not always hold in practice

• We proposed SPLif
  • A new approach for testing SPLs with incomplete/absent FM

• Experiments show that SPLif
  • Helps the user prioritize failing tests and configurations for inspection
  • Is promising and can scale to large systems, such as GCC