Which Configuration Option Should I Change?

Sai Zhang, Michael D. Ernst
University of Washington

Presented by: Kıvanç Muşlu
I have released a new software version ...

I cannot get used to the UI

I do not know how to configure it ...

Developers

Users
Diagnosis of User-Fixable Software Errors

• **Goal:**
  – enable users to fix software errors

• **Challenges:**
  – Errors can be crashing or non-crashing
  – Users much less understand source code
  – Developer tools are of little use
Our previous work [ISSTA’13]

*Help users adapt to the new UI*

I cannot get used to the UI
A new software version

I do not know how to configure it

This paper: How to help users configure the new software version (i.e., diagnosis of configuration errors)
Software system often requires **configuration**

**Configuration errors:**
- Users use wrong values for options
- The software exhibits unintended behaviors

*Example:*

```
--port_num = 100.0
```

Should be a valid integer

Options include:

```
--help           Display this help text
--v or --verbose Print messages about ISendfile actions
-s               Silent, opposite of verbose
-h               No banner for this job
-F=format        Format is one of the following:
                 f - formatted, l - leave control characters, o - Postscript
                 p - use 'pr' format, r - FORTRAN, c - GIF, d - dvi, g - plot
                 n - ditroff, t - troff, v - raster
-C=class         Class is used on banner page; up to 31 characters
-T=job           Job title
```
Configuration errors are common and severe

Root causes of high-severity issues in a major storage company [Yin et al, SOSP’11]

Configuration errors can have disastrous impacts (downtime costs 3.6% of revenue)
Configuration errors are difficult to diagnose

• Error messages are absent or ambiguous
  – e.g., Server Error in '/' Application.
    Description: An error occurred during the parsing of a resource
    (after setting --port_num = 100.0 in web server)

• Infeasible to automatically search for a good configuration
  – Need to know the spec of a valid configuration option value
    (e.g., regex, date time, integer value range)
  – Huge search space
  – Need to specify a testing oracle for automation

• Cannot directly use existing debugging techniques
  [Zhang et al., ICSE’13]
**Goal:** diagnosing configuration errors for evolving software

To maintain the desired behavior on the new version

*Which configuration option should I change?*
Diagnosing configuration errors with ConfSuggester

Key idea:
The execution trace on the old version as the “intended behavior”

Our technique: ConfSuggester

Old version

New version

Suspicious configuration options
Design constraints for ConfSuggester

- **Accessible**: no assumption about user background
  (e.g., users cannot read or write code annotations)

- **Easy-to-use**: fully automated

- **Portable**: no changes to OS or runtime environment

- **Accurate**: few false positives
Outline

• Example
• A Study of Configuration Evolution
• The ConfSuggester Technique
• Evaluation
• Related Work
• Contributions
Outline

• Example
• A Study of Configuration Evolution
• The ConfSuggester Technique
• Evaluation
• Related Work
• Contributions
A popular performance testing tool

Use Jmeter to monitor a website’s performance
Use JMeter to monitor a website’s performance
No regression bugs.

The new version behaves as designed, but differently from what a user expects.

Causes XML parsing error.
All regression tests passed

Version 2.8

ConfSuggester

... Suspicious configuration options

output_format

Resolve the problem: set output_format = XML
Version 2.8

All regression tests passed

Version 2.9
Outline

• Example

• A Study of Configuration Evolution
• The ConfSuggester Technique
• Evaluation
• Related Work
• Contributions
Do configuration changes arise in software evolution?

• 8 open-source programs
  
  Apache  MySQL  Firefox  WEKA
  
  JMeter  randoop  jchord  synoptic
  
• 40 versions released in the past 6 years
• Searched for “configuration changes”-related messages in 7022 commits and 28 change logs
  – Count the number of changes made to configuration options
Results

- Configuration changes arise in every version of all software systems (394 configuration changes in total)

- Configuration change can lead to unexpected behaviors (details later)
Outline

• Example
• A Study of Configuration Evolution
• The ConfSuggester Technique
• Evaluation
• Related Work
• Contributions
**Key insights of ConfSuggester**

- **Control flow** propagates most configuration options’ effects

- The execution traces on the old version can serve as the “intended behavior”
  - The control flow difference and their impacts provides diagnosis clues

```java
/* a configuration option in JMeter */
String output_format = readFromCommandLine();
...
if (output_format == "XML") {
    saveAsXML();
} else {
    saveAsCSV();
}
```

The evaluation result of this predicate affects the next 1000+ instructions
Workflow of ConfSuggester

1.2.3. Melipsis Report

- Old version
- New version

An old trace

A new trace

Trace Comparison

Deviated execution parts (at the predicate-level)
Workflow of ConfSuggester

User demonstration: show the error

Dynamic analysis: understand the behavior

Static analysis: compute the solution
Workflow of ConfSuggester
User demonstration

Old version

New version

An old trace

A new trace

Code instrumentation, monitoring:
1. predicate execution frequency and result
2. execution of each other instruction
Execution trace comparison

An old trace

A new trace

Matching predicates

Identifying deviated predicates

Ranking deviated predicates

: a predicate

: a deviated predicate
Matching predicate across traces

- JDiff algorithm [Apiwattanapong’07]
  - Tolerate small changes between versions

```java
if (output_format == "XML") {
    saveAsXML();
} else {
    saveAsCSV();
}
...  
```

Old version

```java
if (isValidFormat(output_format)) {
    //check validity
}
if (output_format == "XML") {
    checkXMLParser();
    saveAsXML();
} else {
    saveAsCSV();
}
...  
```

New version
**Identifying deviated predicates**

**Goal:**

A predicate $p$’s behavior in an execution trace $t$: 

$$\phi(p, t) = \frac{2}{\text{exec frequency}} + \frac{1}{\text{true ratio}}$$

A predicate $p$’s behavior difference across executions:

$$\text{deviation}(p, t_{\text{old}}, t_{\text{new}}) = |\phi(p, t_{\text{old}}) - \phi(p, t_{\text{new}})|$$

$p$ is a **deviated predicate**, if $\text{deviation}(p, t_{\text{old}}, t_{\text{new}}) > \delta$
Ranking deviated predicates

Rank predicates by their impacts

A predicate p’s deviation impact
= deviation(p, t_{old}, t_{new})
×\{controlled_instructions(p, t_{old})+controlled_instructions(p, t_{new})\}

Defined in the previous slide

Predicate p:

... if (output_format == “XML”) {
  saveAsXML();
} else {
  saveAsCSV();
}
...

Old trace

# of instructions executed

Save As XML

Save As CSV
Ranking deviated predicates

Rank predicates by their impacts

A predicate $p$’s deviation impact

$$= \text{deviation}(p, t_{\text{old}}, t_{\text{new}}) \times (\text{controlled_instructions}(p, t_{\text{old}}) + \text{controlled_instructions}(p, t_{\text{new}}))$$
Ranking deviated predicates

Rank predicates by their impacts

A predicate $p$’s deviation impact

\[
= \text{deviation}(p, t_{old}, t_{new}) \\
\times (\text{controlled_instructions}(p, t_{old}) + \text{controlled_instructions}(p, t_{new}))
\]

Approximate the impact of a predicate’s behavior change to the subsequent program execution.
Find **configuration options** affecting the deviated predicate
- Using static thin slicing [Sridharan ’07]

```java
// a configuration option in JMeter
String output_format = ...;
...
if (output_format == "XML") {
    saveAsXML();
} else {
    saveAsCSV();
}
```

The behavior of this predicate deviates
Compute a backward thin slice from here
Find the affecting predicate

Report
1. output_format
2.
3. ....
Outline

• Example
• A Study of Configuration Evolution
• The ConfSuggester Technique
• Evaluation
• Related Work
• Contributions
8 configuration errors from 6 subjects

<table>
<thead>
<tr>
<th>Subject</th>
<th>LOC</th>
<th>#Options</th>
<th>ΔLOC</th>
<th>#Config errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Randoop</td>
<td>18587</td>
<td>57</td>
<td>1893</td>
<td>1</td>
</tr>
<tr>
<td>Weka</td>
<td>275035</td>
<td>14</td>
<td>1458</td>
<td>1</td>
</tr>
<tr>
<td>Synoptic</td>
<td>19153</td>
<td>37</td>
<td>1658</td>
<td>2</td>
</tr>
<tr>
<td>JChord</td>
<td>26617</td>
<td>79</td>
<td>3085</td>
<td>2</td>
</tr>
<tr>
<td>JMeter</td>
<td>91797</td>
<td>55</td>
<td>3264</td>
<td>1</td>
</tr>
<tr>
<td>Javalanche</td>
<td>25144</td>
<td>35</td>
<td>9261</td>
<td>1</td>
</tr>
</tbody>
</table>

Non-trivial code changes

Reproduced from change logs and user reports.
ConfSuggester’s accuracy

- Measure accuracy by the rank of the actual root cause in ConfSuggester’s output
ConfSuggester’s accuracy

• Measure accuracy by the rank of the actual root cause in ConfSuggester’s output

<table>
<thead>
<tr>
<th>Technique</th>
<th>Average Root Cause Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>23.3</td>
</tr>
</tbody>
</table>

• **Baseline:**
  – Users select options in an arbitrary order
  – Half of the total number of available options
ConfSuggester’s accuracy

- Measure accuracy by the rank of the actual root cause in ConfSuggester’s output

<table>
<thead>
<tr>
<th>Technique</th>
<th>Average Root Cause Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>23.3</td>
</tr>
<tr>
<td>ConfAnalyzer [Rabkin’11]</td>
<td>22</td>
</tr>
</tbody>
</table>

- **ConfAnalyzer:**
  - Use program slicing for error diagnosis
ConfSuggester’s accuracy

• Measure accuracy by the rank of the actual root cause in ConfSuggester’s output

<table>
<thead>
<tr>
<th>Technique</th>
<th>Average Root Cause Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>23.3</td>
</tr>
<tr>
<td>ConfAnalyzer [Rabkin’11]</td>
<td>22</td>
</tr>
<tr>
<td>ConfDiagnoser [Zhang’13]</td>
<td>15.3</td>
</tr>
</tbody>
</table>

• ConfDiagnoser:
  – Use trace comparison (on the same version) for error diagnosis
ConfSuggester’s accuracy

- Measure accuracy by the rank of the actual root cause in ConfSuggester’s output

<table>
<thead>
<tr>
<th>Technique</th>
<th>Average Root Cause Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>23.3</td>
</tr>
<tr>
<td>ConfAnalyzer [Rabkin’11]</td>
<td>22</td>
</tr>
<tr>
<td>ConfDiagnoser [Zhang’13]</td>
<td>15.3</td>
</tr>
<tr>
<td>ConfSuggester (this paper)</td>
<td>1.9</td>
</tr>
</tbody>
</table>

- ConfSuggester:
  - 6 errors: root cause ranks 1st
  - 1 error: root cause ranks 3rd
  - 1 error: root cause ranks 6th
ConfSuggester’s efficiency

• **User demonstration**
  – 6 minutes per error, on average

• **Error diagnosis**
  – 4 minutes per error, on average
Outline

• Example
• A Study of Configuration Evolution
• The ConfSuggester Technique
• Evaluation
• Related Work
• Contributions
Related work on configuration error diagnosis

• Tainting-based techniques
  – Dynamic tainting [Attariyan’08], static tainting [Rabkin’11]
    *Focuses exclusively on crashing errors*

• Search-based techniques
  – Delta debugging [Zeller’02], Chronus [Whitaker’04]
    *Requires a correct state for comparison, or OS-level support*

• Domain-specific techniques
  – PeerPressure [Wang’04], RangeFixer [Xiong’12]
    *Targets a specific kind of configuration errors, and does not support a general language like Java*

**A common limitation:** do not support configuration error diagnosis in software evolution.
Outline

• Example
• A Study of Configuration Evolution
• The ConfSuggester Technique
• Evaluation
• Related Work
• Contributions
Contributions

• A technique to diagnose configuration errors for evolving software

  *Compare relevant predicate behaviors between executions from two versions*

✓ Accessible: no assumption about user background
✓ Easy-to-use: fully automated
✓ Portable: no changes to OS or runtime environment
✓ Accurate: few false positives

• The ConfSuggester tool implementation

  [http://config-errors.googlecode.com](http://config-errors.googlecode.com)