Proactive Detection of Inadequate Diagnostic Messages for Software Configuration Errors

Sai Zhang
Google Research

Michael D. Ernst
University of Washington
**Goal:** helping developers improve software error diagnostic messages

Users → Input data → Configuration → Software → Errors

- Crashing
- Silent failures

---

---

**--port_num = 100.0** (should be an integer)

**Our technique:** detecting such inadequate diagnostic messages caused by configuration errors

A bad diagnostic message: “… unexpected system failure …”
**Goal**: helping developers improve software error diagnostic messages
**Goal**: helping developers improve software error diagnostic messages

Users

```
--port_num = 100.0
(should be an integer)
```

Configuration

Software (with improved diagnostic message)

A good diagnostic message:

“… wrong value in –port_num…”
Why configuration errors?

• Software systems often require configuration
Why configuration errors?

- Software systems often require configuration
- Software 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)
Why diagnostic messages?

• Often the **sole data source** available to understand an error

• Many diagnostic messages in practice are **inadequate**
  – Missing
  – Ambiguous
Why diagnostic messages?

• Often the sole data source available to understand an error

• Many diagnostic messages in practice are inadequate
  - Missing
  - Ambiguous

A misconfiguration in Apache JMeter
  output_format = XYZ (an unsupported format)

No diagnostic message, but JMeter saves output in the default “XML” format
Why diagnostic messages?

- Often the sole data source available to understand an error
- Many diagnostic messages in practice are inadequate
  - Missing
  - Ambiguous

A misconfiguration in Apache Derby
derby.stream.error.method = hello

Diagnostic message:
IJ ERROR: Unable to establish connection
Why diagnostic messages?

• Often the **sole data source** available to understand an error

• Many diagnostic messages in practice are **inadequate**
  – Missing
  – Ambiguous

**Our technique**: detecting those **inadequate messages before they arise in the field.**
Outline

• Motivation

• The ConfDiagDetector technique

• Evaluation

• Related work

• Contributions
Challenges of **proactive** detection of **inadequate** diagnostic messages

- How to *trigger* a configuration error?

- How to *determine the inadequacy* of a diagnostic message?
ConfDiagDetector’s solutions

• How to *trigger a configuration error*?
  – Configuration mutation + checking system tests’ results

• How to *determine the inadequacy* of a diagnostic message?
  – Use a NLP technique to check its semantic meaning

Similar semantic meanings?

Failed tests ≈ triggered errors

Diagnostic messages output by failed tests

Use manual
ConfDiagDetector workflow

System tests

All tests pass!
**ConfDiagDetector workflow**

- **An example configuration**
- **Configuration mutation**
- **System tests**
- **Diagnostic messages issued by failed tests**
- **Run tests under each Mutated configuration**
- **Message analysis**
- **Inadequate Diagnostic messages**

**Use manual**

Software (binary)
Configuration mutation

• Randomly mutates option values
  – One mutated option in each mutated configuration

A configuration → Mutated configurations
Configuration mutation

• Randomly mutates option values
  – One mutated option in each mutated configuration

• Mutation rules for one configuration option
  – Delete existing value
    \[ \text{format=xml} \rightarrow \text{format=} \]
  – Using a random value
    \[ \text{format=xml} \rightarrow \text{format= xyz} \]
  – Injecting spelling mistakes
    \[ \text{format=xml} \rightarrow \text{format= xmk} \]
  – Change the case of text
    \[ \text{format=xml} \rightarrow \text{format= XML} \]
Running tests

• Run the *all* tests under *each* mutated configuration

• Parse each failed test’s log file or console to get the diagnostic message
Running tests

• Run the *all* tests under *each* mutated configuration

• Parse *each failed test's* log file or console to get the diagnostic message
Message analysis

- A message is adequate, if it
  - contains the mutated option name or value
  OR
  - has a similar semantic meaning with the manual description
Message analysis

- A message is adequate, if it
  - contains the mutated option name or value
  - has a similar semantic meaning with the manual description

**Example:**

Mutated option:

```
--percentage-split
```

Diagnostic message:

“the value of `percentage-split` should be > 0”
Message analysis

• A message is adequate, if it
  – contains the mutated option name or value
  – has a similar semantic meaning with the manual description

OR

Example:
  Mutated option:  
    --fnum
  Diagnostic message:  
    “Number of folds must be greater than 1”  
  User manual description of --fnum:  
    “Sets number of folds for cross-validation”
Message analysis

• A message is adequate, if it
  – contains the mutated option name or value
  – has a similar semantic meaning with the manual description

A NLP technique [Mihalcea’06]
Key idea of the employed NLP technique

A message

Has similar semantic meanings, if many words in them have similar meanings

Example:

The program goes wrong

The software fails

Manual description

• Remove all stop words
• For each word in the diagnostic message, tries to find the similar words in the manual
• Two sentences are similar, if “many” words are similar between them.
Outline

• Motivation
• The ConfDiagDetector technique
• Evaluation
• Related work
• Contributions
Research questions

• ConfDiagDetector’s effectiveness
  – The detected inadequate messages
  – Time cost in inadequate message detection
  – Comparison with two existing techniques
### 4 mature configurable software systems

<table>
<thead>
<tr>
<th>Subject</th>
<th>LOC</th>
<th>#Options</th>
<th>#System Tests</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weka</td>
<td>274,448</td>
<td>125</td>
<td>16</td>
</tr>
<tr>
<td>JMeter</td>
<td>91,979</td>
<td>212</td>
<td>5</td>
</tr>
<tr>
<td>Jetty</td>
<td>123,028</td>
<td>23</td>
<td>7</td>
</tr>
<tr>
<td>Derby</td>
<td>645,017</td>
<td>56</td>
<td>7</td>
</tr>
</tbody>
</table>

Converted from usage examples in the user manual.
Detected inadequate diagnostic messages

50 distinct diagnostic messages
Detected inadequate diagnostic messages

- 25 missing messages
- 18 ambiguous messages
- 7 adequate messages
Detected inadequate diagnostic messages

- 25 missing messages
- 18 ambiguous messages
- 7 adequate messages

Validating each message’s adequacy by user study
User study

User manual

Diagnostic message

3 grad students

Each with 10 years coding experience

Adequate or not?
User study results

ConfDiagDetector’s results

- 7 adequate messages
- 18 ambiguous messages
- 25 missing messages

User’s judgment

- 8 adequate messages
- 17 ambiguous messages

Differs only in 1 message

Zero false negative, and 2% false positive rate
Time cost

• **Manual effort**  
  – 3.5 hours in total (4.2 minutes per message)  
  • Converting usage examples into tests  
  • Extract configuration option description from the user manual

• **ConfDiagDetector’s efficiency**  
  – 3 minutes per message, on average
Comparison with two existing techniques

• No Text Analysis
  – Implemented in ConfErr [Keller’08] and Spex-INJ [Yin’11]
  – A message is adequate if the misconfiguration option name or value appears in it
  – False positive rate: 16% (ConfDiagDetector’ rate: 2%)

• Internet search
  – Search the diagnostic message in Google
  – A message is adequate if the misconfiguration option appears in the top 10 entries
  – False positive rate: 12% (ConfDiagDetector’ rate: 2%)
Outline

• Motivation

• The ConfDiagDetector technique

• Evaluation

• Related work

• Contributions
Related work

• Configuration error diagnosis techniques
  – Dynamic tainting [Attariyan’08], static tainting [Rabkin’11],
    Chronus [Whitaker’04]

  Troubleshooting an exhibited error rather than detecting
  inadequate diagnostic messages

• Software diagnosability improvement techniques
  – PeerPressure [Wang’04], RangeFixer [Xiong’12], ConfErr
    [Keller’08] and Spex-INJ [Yin’11], EnCore [Zhang’14]

  Requires source code, usage history, or OS-level support
Outline

• Motivation
• The ConfDiagDetector technique
• Evaluation
• Related work
• Contributions
Contributions

• A technique to detect inadequate diagnostic messages
  
  *Combine configuration mutation and NLP techniques*
  
  – Requires no source code and prior knowledge
  – Analyzes diagnostic messages in natural language
  – Requires no OS-level support
  – Accurate and fast

• An evaluation on 4 mature, configurable systems
  – Identify 25 missing and 18 inadequate messages
  – No false negative, 2% false positive rate