Toward Efficient Aspect Mining for Linux

Danfeng Zhang, Yao Guo, Xiangqun Chen

Institute of Software, Peking University,
Beijing, PR China
Talk Outline

- Motivation & Background
- Crosscutting Concerns in Linux
- Case Study on Current Mining Approaches
- Proposed Mining Approaches
- Experimental Results
- Conclusion
Evolution of AOP

- AOP has been successful during the last decade
  - Aspect-Oriented Languages
  - Aspect-Oriented Implementations
  - Aspect Mining
  - ……

- Many systems have been *aspectized*.
AOP for Legacy Software

- Aspect Mining -> Refactoring
Aspect Mining

- Current Approaches mainly focus on Object-Oriented Programs
  - Identify Analysis
    - Based on good naming conventions
      - E.g., using Natural Language Processing (AOSD’07)
  - Clone Detection
    - Code clones are likely aspects!
    - Many implementations, such as CCFinder.
  - Fan-in Analysis
    - Calculate the fan-in value of a method
    - High fan-in \(\Rightarrow\) more likely an aspect
Aspect Mining for Linux

- **Background**
  - Many researchers have explored AOP in operating systems
    - Coady’s work on FreeBSD, PURE, Bossa(Linux), etc.
  - Little work on how to identify crosscutting concerns in Linux

- **Our Motivation**
  - To evaluate how existing mining approaches work on Linux
  - Explore new aspect mining approaches for Linux
    - Concerns could be found more effectively by mining approaches targeting at their characteristics
How to Identify Meaningful Crosscutting Concerns?

Identifying Crosscutting Concerns

At what granularity of aspect should we mine?
- Coarse granularity
  - Memory management, interrupt handling, system calls……
- Finer granularity
  - How about page allocation, page swapping in MM?

A crosscutting concern should possess the following desired properties [Marion AOSD’06]
- A general intent
- An implementation idiom in a non-AOP language
- An aspect mechanism to refactor
Studied Concerns in Linux

- Four Crosscutting concerns are chosen for mining
  - **Parameter Check**: code to validate a parameter or handle different parameters
  - **Error Handling**: code to check whether a function succeeds, and handle the error accordingly in the case of an error
  - **Synchronization**: code to handle synchronization in Linux
  - **Tracing**: the trace point in the Linux code implementing the system call “ptrace”
Concerns Distribution

- Manual identification of all occurrences of these concerns in (a subset of) Linux
  - Work done by students exploring Linux source code

<table>
<thead>
<tr>
<th>Aspect</th>
<th>LOC</th>
<th>Fraction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter Check</td>
<td>3943</td>
<td>4.71%</td>
</tr>
<tr>
<td>Error Handling</td>
<td>12310</td>
<td>14.69%</td>
</tr>
<tr>
<td>Synchronization</td>
<td>1162</td>
<td>1.39%</td>
</tr>
<tr>
<td>Tracing</td>
<td>203</td>
<td>0.24%</td>
</tr>
<tr>
<td>Total</td>
<td>17618</td>
<td>21.03%</td>
</tr>
</tbody>
</table>
Experimental Framework

- Implemented as a plug-in based on Eclipse
- Used CDT (C/C++ Development Tools) as the indexer and parser
  - Due to the limitation of CDT, we analyzed a subset of the entire Linux 2.4.18
  - Over 1000 .c files
  - Over 83,000 lines of code
- Clone Detection implementation
  - CCFinder (10.1.12.4)
- Fan-in analysis implementation
  - Using CDT
Evaluation Criteria

- Mining Coverage
  - Percentage of identified concerns among all crosscutting concerns in the code

- Mining Precision
  - Percentage of “true” aspect candidates among all the candidates identified

- Coverage vs. Precision
  - which one is more important?
Mining Parameter Check and Error Handling Concern

- Examples

**Parameter Check**
```
if (table == NULL) {
    unlock_kernel();
    return i;
}
```

**Error Handling**
```
p = alloc_task_struct();
if (!p)
    return p;
```

- Clone detection is applied to identify these concerns
  - We use CCFinder as the clone detection tool
  - It can only find about 44% of them with about 40% fake candidates
Mining Parameter Check and Error Handling Concern

Proposed Technique

- Pattern-based approach

Parameter Check

```
parament_check → if lpar exp_of_para rpar
code_segment else_segment |
switch lpar exp_of_para rpar lbpar
case_statements default_statement rbpar |
else_segment → else code_segment | null
```

Error Handling

```
error_handling → if lpar exp_of_funcall rpar
code_segment else_segment |
switch lpar exp_of_funcall rpar lbpar
case_statements default_statement rbpar |
assign_statement statement branch_statement
assign_statement → id EQ function_call semicolon
else_segment → else code_segment | null
```
Mining Parameter Check and Error Handling Concern

Implementation of New Technique

- Pattern-based approach
  - DOM (Document Object Model) is used
    - DOM tree is generated by CDT
    - Pattern matching is accomplished by walking through the DOM tree

- The approach needs some help
  - An expert who is familiar with the source code is needed to specify the patterns
Mining Parameter Check and Error Handling Concern

Results

![Bar Chart]

- Precision
- Coverage for error handling
- Coverage for parameter check

Pattern Based vs Clone Detection
Mining Synchronization

- Similar concerns on synchronization have been studied in PURE
- Synchronization in Linux is very important for maintainability and evolution.

<table>
<thead>
<tr>
<th>Mechanisms</th>
<th>Related functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atomic operation</td>
<td>ATOMIC_INIT, atomic_read, atomic_set, atomic_add, atomic_sub, atomic_dec, atomic_add_negative, atomic_sub_and_test, atomic_inc, atomic_dec_and_test</td>
</tr>
<tr>
<td>mutex</td>
<td>DECLARE_MUTEX, DECLARE_MUTEX_LOCKED, down_interruptible, init_MUTEX, init_MUTEX_LOCKED, down_trylock, up, down,</td>
</tr>
<tr>
<td>read/write semaphore</td>
<td>DECLARE_RWSEM, init_rwlock, down_write, up_read, up_write, down_read, rwsem_atomic_update, rwsem_atomic_add,</td>
</tr>
<tr>
<td>spin lock</td>
<td>spin_lock, spin_trylock, spin_unlock, spin_lock_init, spin_is_locked, spin_unlock_wait, spin_lock_bh, spin_lock_string, spin_unlock_string, spin_lock_irqsave, spin_lock_irq, spin_unlock_irqrestore, spin_unlock_irq, spin_is_locked, spin_unlock_bh, spin_trylock_bh,</td>
</tr>
<tr>
<td>read/write lock</td>
<td>read_lock, write_lock, read_unlock, write_unlock, rwlock_init,</td>
</tr>
<tr>
<td>big kernel lock</td>
<td>lock_kernel, unlock_kernel, kernel_locked, release_kernel_lock, reacquire_kernel_lock</td>
</tr>
</tbody>
</table>
Mining Synchronization

Apply Current Technique

- Synchronization is called from many places
  - Fan-in analysis seems to be a good fit

Threshold affects the mining precision & coverage

“set_xxxx”, “get_xxxx” in Linux are filtered

---

**Step 1.** Automatically compute the fan-in metric for all the methods in the targeted source.

**Step 2.** Filter the result of the first step.

- Restrict the set of methods having fan-in above a certain threshold.
- Filter getters and setters from this restricted set.
- Filter utility methods such as `toString()`.

**Step 3.** (Mainly manual) Analysis of the remaining set of methods.
Fan-in analysis applied

- Implemented using CDT
- Function-like macros in C are treated as functions.

Results are not encouraging
- 20-30% coverage with different threshold.
- 50-90% precision with different threshold
Mining Synchronization

Improving the Results?

- Observation
  - Many functions of synchronization concern have low fan-in’s
  - However, lower the threshold would include more “false” candidates
    - Which will affect the precision
  - Many functions follow regular naming conventions
    - With the same or similar prefix

- Solution
  - Group the functions based on their prefixes into classes
  - Calculate fan-in’s for the whole class, instead of for each individual function
  - Identify the whole class as an aspect candidate
Classified fan-in analysis

*Step 1.* Classify all the object-like macros into classes by the prefix of their signature.

*Step 2.* Automatically compute the fan-in metric for all the classes generated in step 1.

*Step 3.* Filtering of the result of step 2:
- Restrict the set of classed to those having a fan-in above a certain threshold.
- Filter meaningless classes, like class with a prefix `MAX_`, `MIN_`.

*Step 4.* (Mainly manual) Analyze object-like macros in the remaining set of classes.
Mining Synchronization

Results

**Figure 2.** Performance comparison for classified fan-in analysis.

A. Coverage at different threshold.

B. Precision at different threshold.
Mining Tracing

- Bruntink [ICSM 2004] has applied clone detection on Dynamic Tracing Mining.
  - In Linux, it’s different
  - Clone detection achieves only about 12% coverage based on our evaluation

Tracing - example

```c
if (p->ptrace & PT_PTRACED)
    send_sig(SIGSTOP, p, 1);
```
Mining Tracing

Proposed Technique

- Specific macros are used for this concern
- Use these macros to find this concern
- Extend the above proposed classified fan-in analysis approach to include macros.

```c
#define PT_PTRACED 0x00000001
#define PT_TRACESYS 0x00000002
#define PT_DTRACE 0x00000004
#define PT_TRACESYSGOOD 0x00000008
#define PT_PTRACE_CAP 0x00000010
```
Coverage is always 100%.

Figure 3. Precision at different threshold for extended classified fan-in analysis.
Conclusion

- A case study of aspect mining in Linux
  - Identified four important aspects in Linux
  - Applied several existing aspect mining approaches to identify them
  - Proposed three new aspect mining approaches
  - Experiments have shown promising results towards efficient aspect mining in Linux.
Thank You!
Motivations behind

1. Identifier Analysis: Based on Good Naming Conventions
2. Fan-in Analysis: Implementation of crosscutting concerns by means of a single method in the system
3. Clone Detection: Implementation of crosscutting concerns by code duplication