Toward Efficient Aspect Mining for Linux

Danfeng Zhang, **Yao Guo**, Xiangqun Chen

Institute of Software, Peking University, Bejing, PR China

Talk Outline

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

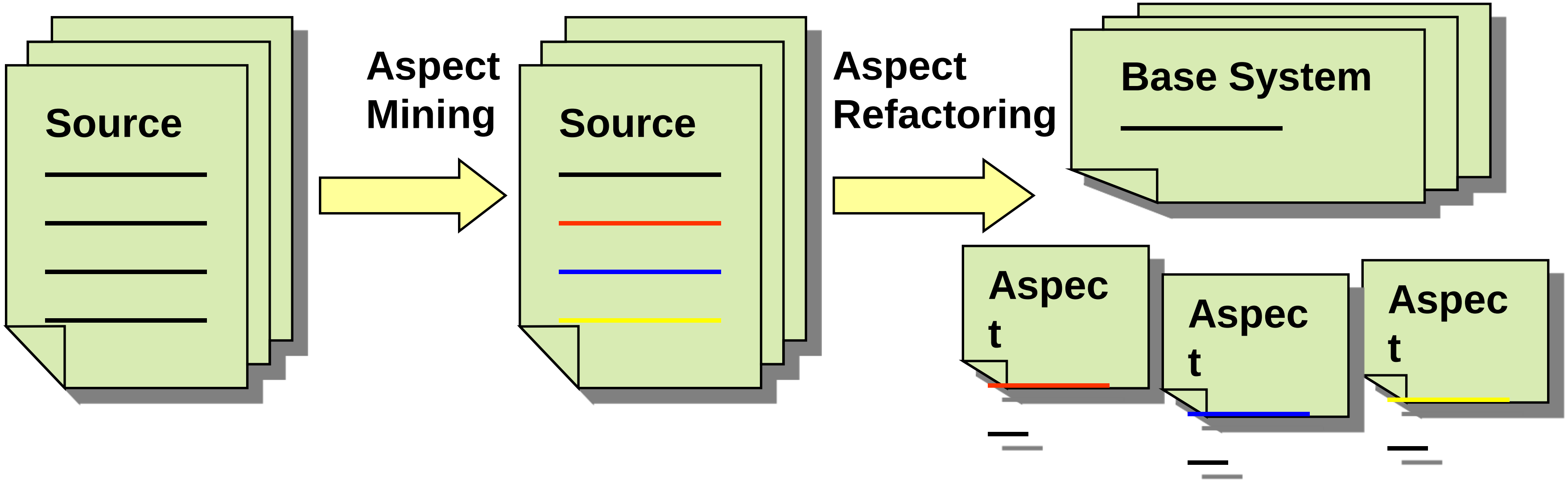
Evolution of AOP

- 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 ➔ 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

Aspect	LOC	Fraction
Parameter Check	3943	4.71%
Error Handling	12310	14.69%
Synchronization	1162	1.39%
Tracing	203	0.24%
Total	17618	21.03%

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 Handing 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 dection tool
  - It can only find about 44% of them with about 40% fake candidates

Mining Parameter Check and Error Handing 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

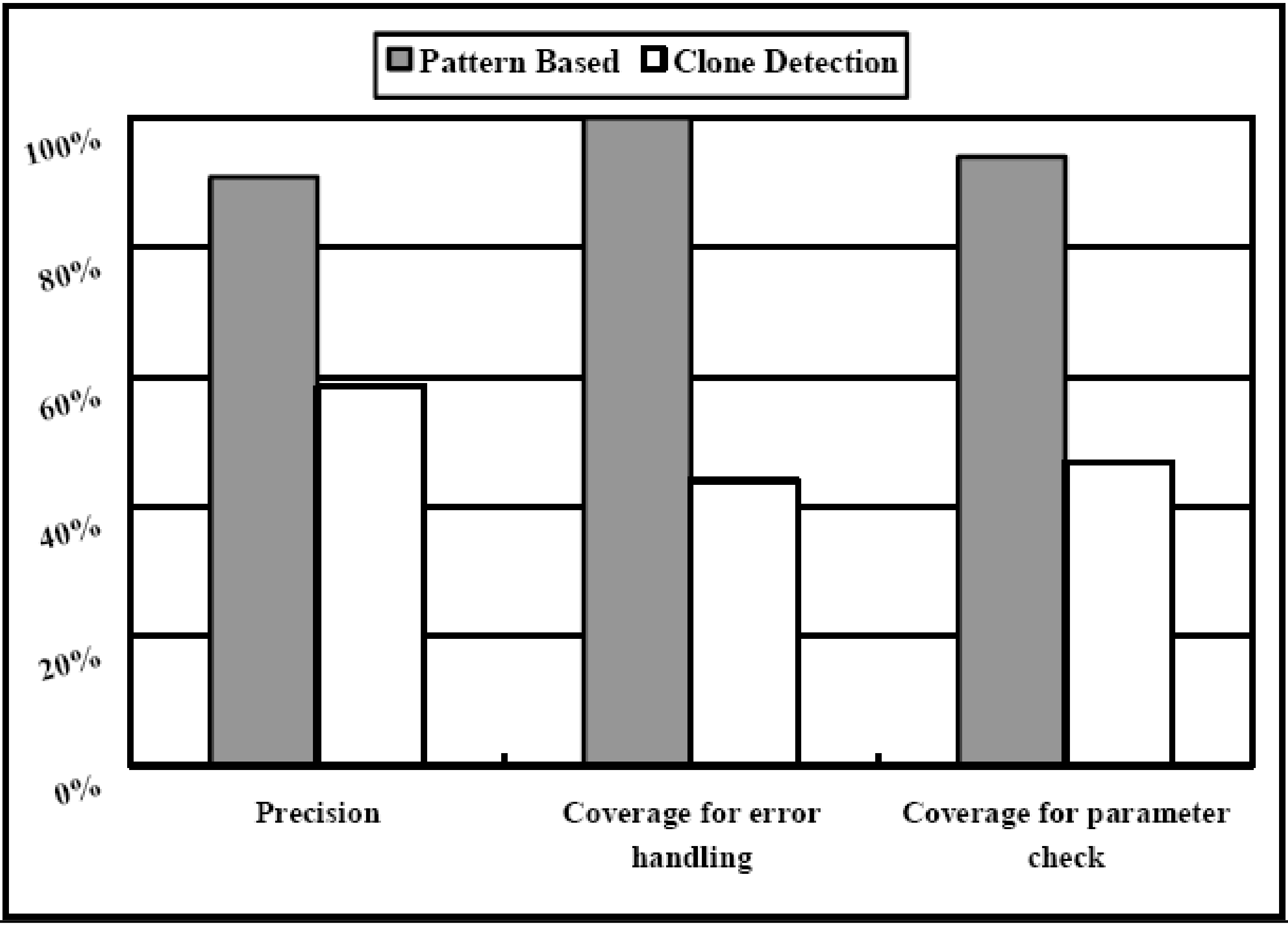
error_handing → **if lpar exp_of_funcall rpar**
code_segment else_segment |
switch lpar exp_of_funcall rpar lbpar
case_statements default_statement **rbpar** |
assign_statement statement branch_statement
assign_statement → id **EQ** function_call **semicolon**
else_segment → **else** code_segment | **null**

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 Handing Concern

Results



Mining Synchronization

Mechanisms	Related functions
Atomic operation	**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, atomic_inc_and_test*
mutex	**DECLARE_MUTEX**, **DECLARE_MUTEX_LOCKED**, *down_interruptible, init_MUTEX, init_MUTEX_LOCKED down_trylock,up,down,*
read/write semaphore	**DECLARE_RWSEM**, *init_rwsem, down_write, up_read, up_write down_read, rwsem_atomic_update, rwsem_atomic_add,*
spin lock	*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,**
read/write lock	**read_lock, write_lock, read_unlock, write_unlock, rwlock_init**
big kernel lock	*lock_kernel, unlock_kernel,* **kernel_locked, release_kernel_lock, reacquire_kernel_lock**

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

Apply Current Technique

- Synchronization is called fr
  - Fan-in analysis seems to be

Step 1. Automatically compute the metric for all the methods in the targeted sour

Step 2. Filter the result of the firs

- Restrict the set of metho 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.

Threshold affects the mining precision & coverage

“set_xxxx”, “get_xxx” in Linux are filtered

Results for Fan-in Analysis

- 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

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 a an aspect candidate

Proposed Technique

- 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.

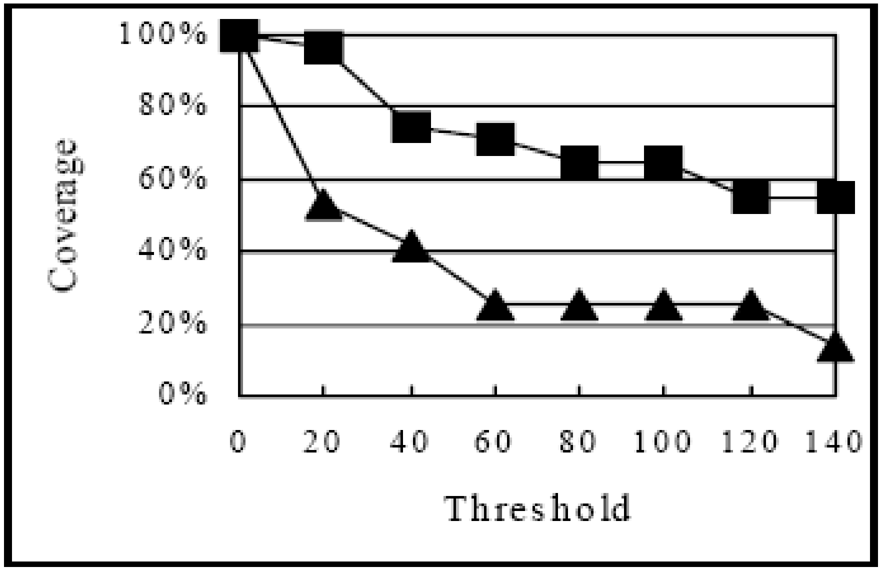
Step 3. Filtering of the result of step 2:

- 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_*.

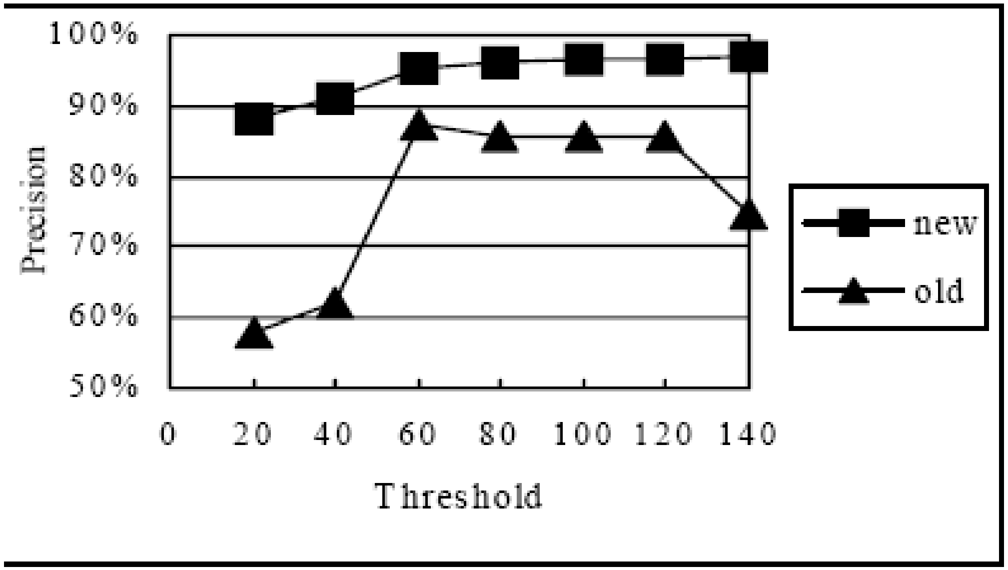
Step 4. (Mainly manual) Analyze object-like macros in the remaining set of classes.

Mining Synchronization

Results



A. Coverage at different threshold.



B. Precision at different threshold.

Figure 2. Performance comparison for classified fan-in analysis.
New: Classified fan-in analysis. Old: Original fan-in analysis.

Mining Tracing

Tracing - example

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

- 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

Proposed Technique

\linux\include\linux\Sched.h

```
#define PT_PTRACED 0x00000001
#define PT_TRACESYS 0x00000002
#define PT_DTRACE 0x00000004
#define PT_TRACESYSGOOD 0x00000008
#define PT_PTRACE_CAP 0x00000010
```

- 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.

Results

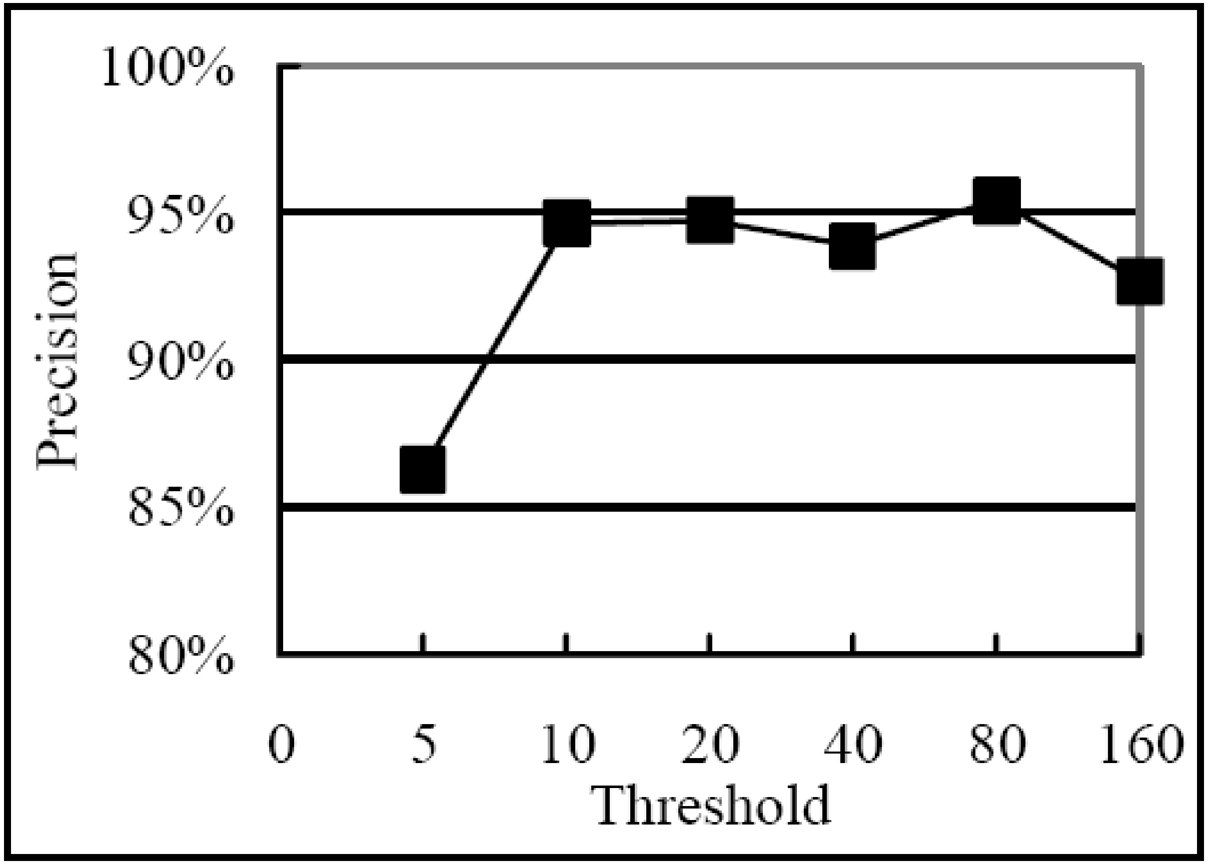


Figure 3. Precision at different threshold for extended classified fan-in analysis

Coverage is always 100%.

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

Identifier Analysis

1

Based on Good Naming Conventions

Fan-in Analysis

2

Implementation of crosscutting concerns by means of a single method in the system

Clone Detection

3

Implementation of crosscutting concerns by code duplication