OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG

INF FAKULTÄT FÜR INFORMATIK

Code Smells Revisited: A Variability Perspective

Wolfram Fenske,[1] Sandro Schulze[2]

21st January, 2015

[1] University of Magdeburg, Germany
[2] TU Braunschweig, Germany

Code Smells (1)

- *Code smell* term introduced by Fowler, Beck, Brant, Opdyke & Roberts (1999)
- Hint at structural problem / design weakness of the code
- Smelly code $\neq$ buggy code, but
- Smelly code hinders . . .
  - Program comprehension
  - Maintenance (e. g., bug fixing)
  - Evolution (e. g., adding new functionality)
- Indicator that structure should be improved

Code Smells (2)

- So far focus on single systems – variability ignored
- Some work on smells in SPL context, e. g., *variability smells* (Apel et al., 2013):
  - Obscure feature model,
  - #ifdef hell or many extension points,
  - Traceability mess,
  - ...
  - ➥ Rather unsystematic, different kinds of artifacts, not evaluated

Our Approach (1)

- Focus on code smells exclusively
- Use established code smells by Fowler et al. (1999)

"How does smelly code look different if variability is involved?"

- Consider two variability mechanisms:
  1. Composition-based: feature-oriented programming (FOP)
  2. Annotation-based: C preprocessor (cpp)

Our Approach (2)

LONG METHOD in MySQL

```
sig_handler process_alarm(int sig
    __attribute__((unused))) {
  sigset_t old_mask;
  if (thd_lib_detected==THD_LIB_LT &&
      !pthread_equal(pthread_self(),
                     alarm_thread)) {

    printf("thread_alarm in " \
           "process_alarm\n");
    fflush(stdout);


    my_sigset(thr_client_alarm,
              process_alarm);

    return;
  }

  pthread_sigmask(SIG_SETMASK,
    &full_signal_set, &old_mask);
  mysql_mutex_lock(&LOCK_alarm);

  process_alarm_part2(sig);

  /* more code */
}
```

Our Approach (2)

LONG METHOD in MySQL

```
1  sig_handler process_alarm(int sig
2      __attribute__((unused))) {
3    sigset_t old_mask;
4    if (thd_lib_detected==THD_LIB_LT &&
5        !pthread_equal(pthread_self(),
6                       alarm_thread)) {
7
8      printf("thread_alarm in " \
9             "process_alarm\n");
10     fflush(stdout);
11
12
13     my_sigset(thr_client_alarm,
14               process_alarm);
15
16     return;
17   }
18
19   pthread_sigmask(SIG_SETMASK,
20     &full_signal_set, &old_mask);
21   mysql_mutex_lock(&LOCK_alarm);
22
23   process_alarm_part2(sig);
24
25   /* more code */
26 }
```

LONG METHOD with #ifdefs

```
1  sig_handler process_alarm(int sig
2      __attribute__((unused))) {
3    sigset_t old_mask;
4    if (thd_lib_detected==THD_LIB_LT &&
5        !pthread_equal(pthread_self(),
6                       alarm_thread)) {
7  #if defined(MAIN) && !defined(__bsdi__)
8      printf("thread_alarm in " \
9             "process_alarm\n");
10     fflush(stdout);
11 #endif
12 #ifdef SIGNAL_HANDLER_RESET_ON_DELIVERY
13     my_sigset(thr_client_alarm,
14               process_alarm);
15 #endif
16     return;
17   }
18 #ifndef USE_ALARM_THREAD
19   pthread_sigmask(SIG_SETMASK,
20     &full_signal_set, &old_mask);
21   mysql_mutex_lock(&LOCK_alarm);
22 #endif
23   process_alarm_part2(sig);
24 #ifndef USE_ALARM_THREAD
25   /* more code */
26 }
```

Annotation Bundle

Derived from: Long Method

Example: MySQL, `mysys/thr_alarm.c`

```
1  sig_handler process_alarm(int sig __attribute__((unused))) {
2    sigset_t old_mask;
3    if (thd_lib_detected == THD_LIB_LT &&
4        !pthread_equal(pthread_self(),alarm_thread)) {
5  #if defined(MAIN) && !defined(__bsdi__)
6      printf("thread_alarm in process_alarm\n");
7      fflush(stdout);
8  #endif
9  #ifdef SIGNAL_HANDLER_RESET_ON_DELIVERY
10     my_sigset(thr_client_alarm, process_alarm);
11 #endif
12     return;
13   }
14 #ifndef USE_ALARM_THREAD
15   pthread_sigmask(SIG_SETMASK,&full_signal_set, &old_mask);
16   mysql_mutex_lock(&LOCK_alarm);
17 #endif
18   process_alarm_part2(sig);
19 #ifndef USE_ALARM_THREAD
20 #if !defined(USE_ONE_SIGNAL_HAND) && defined(SIGNAL_HANDLER_RESET_ON_DELIVERY)
21   my_sigset(THR_SERVER_ALARM,process_alarm);
22 #endif
23   mysql_mutex_unlock(&LOCK_alarm);
24   pthread_sigmask(SIG_SETMASK,&old_mask,NULL);
25 #endif
26   return;
27 }
```

Long Refinement Chain

Derived from: Long Method

Example: GUIDSL (FOP), method `Main#process(Model)`

```
class Main { // Feature 'dmain'
  public static void process(Model root) throws /*...*/ {
    // layers extend this method for AST processing
  }
}
```

Long Refinement Chain

Derived from: Long Method

Example: GUIDSL (FOP), method `Main#process(Model)`

```
class Main { // Feature 'dmain'
  public static void process(Model root) throws /*...*/ {
    // layers extend this method for AST processing
  }
}
```

```
class Main { // Feature 'fillgs'
  public static void process(Model root) throws /*...*/ {
    original(m);
    // harvest the tree
    m.harvest( new fillFPtable() );
    if (Util.errorCount() != 0)
      throw new SemanticException(
        "Error(s) in specification found");
    m.harvest( new enterGspec() );
    if (Util.errorCount() != 0)
      throw new SemanticException(
        "Error(s) in specification found");
  }
}
```

Long Refinement Chain

Derived from: Long Method

Example: GUIDSL (FOP), method `Main#process(Model)`

```
class Main { // Feature 'dmain'
  public static void process(Model root) throws /*...*/ {
    // layers extend this method for AST processing
  }
}
```

```
class Main { // Feature 'fillgs'
  public static void process(Model root) throws /*...*/ {
    original(m);
  }
}
```

```
class Main { // Feature 'propgs'
  public static void process(Model root) throws /*...*/ {
    original(m);
    grammar.current.visit( new propcons() );
    if (Util.errorCount() !=0)
      throw new SemanticException(
        "Errors in propagating Constraints");
  }
}
```

Long Refinement Chain

Derived from: Long Method

Example: GUIDSL (FOP), method `Main#process(Model)`

```
class Main { // Feature 'dmain'
  public static void process(Model root) throws /*...*/ {
    // layers extend this method for AST processing
  }
}
```

```
class Main { // Feature 'fillgs'
  public static void process(Model root) throws /*...*/ {
    original(m);
  }
}
```

```
class Main { // Feature 'propgs'
  public static void process(Model root) throws /*...*/ {
    original(m);
  }
}
```

```
class Main { // Feature 'formgs'
  public static void process(Model root) throws /*...*/ {
    original(m);
    production.makeFormula();
    pattern.makeFormula();
    if (Util.errorCount() != 0)
      throw new SemanticException(
        "Errors in making propositional formulas");
  }
}
```

Long Refinement Chain

Derived from: Long Method

Example: GUIDSL (FOP), method `Main#process(Model)`

```
class Main { // Feature 'dmain'
  public static void process(Model root) throws /*...*/ {
    // layers extend this method for AST processing
  }
}
```

```
class Main { // Feature 'fillgs'
  public static void process(Model root) throws /*...*/ {
    original(m);
```

```
class Main { // Feature 'propgs'
  public static void process(Model root) throws /*...*/ {
    original(m);
```

```
class Main { // Feature 'formgs'
  public static void process(Model root) throws /*...*/ {
    original(m);
```

```
class Main { // Feature 'clauselist'
  public static void process(Model root) throws /*...*/ {
    original(m);
    production.makeClauses();
    pattern.makeClauses();
    ESList.makeClauses();
    grammar.makeClauses();
    if (Util.errorCount() != 0)
      throw new SemanticException(
        "Errors in making conjunctive normal formulas");
  }
}
```

Long Refinement Chain

Derived from: Long Method

Example: GUIDSL (FOP), method `Main#process(Model)`

```
class Main { // Feature 'dmain'
  public static void process(Model root) throws /*...*/ {
    // layers extend this method for AST processing
  }
}
```

```
class Main { // Feature 'fillgs'
  public static void process(Model root) throws /*...*/ {
    original(m);
  }
}
```

```
class Main { // Feature 'propgs'
  public static void process(Model root) throws /*...*/ {
    original(m);
  }
}
```

```
class Main { // Feature 'formgs'
  public static void process(Model root) throws /*...*/ {
    original(m);
  }
}
```

```
class Main { // Feature 'clauselist'
  public static void process(Model root) throws /*...*/ {
    original(m);
  }
}
```

```
class Main { // Feature 'modelopts'
  public static void process(Model root) throws /*...*/ {
    original(m);
    if (modelMode) {
      try { harvestInfo(); }
      catch (IOException e) {
        JOptionPane.showMessageDialog(null, "Model" +
        " Harvesting Error — see command line for" +
        " details", "Error!", JOptionPane.ERROR_MESSAGE);
        System.err.println(e.getMessage());
}}}}
```

Latently Unused Parameter

Derived from: Long Parameter List & Speculative Generality

Example: Graph Product Line (FOP)

Feature *WeightedOnlyVertices*

```
1  public class Graph {
2    /* More source code ... */
3
4    public void addAnEdge(Vertex start,
5        Vertex end, int weight)
6    {
7      addEdge(start, end, weight);
8    }
9
10   public void addEdge(Vertex start,
11                       Vertex end,
12                       int weight)
13   {
14     addEdge(start, end);
15     start.addWeight(weight);
16     /* More source code ... */
17   }
18
19   /* More source code ... */
20 }
```

Feature *DirectedOnlyVertices*

```
1  public class Graph {
2    /* More source code ... */
3
4    public void addAnEdge(Vertex start,
5        Vertex end, int weight)
6    {
7      addEdge(start, end);
8    }
9
10   public EdgeIfc addEdge(Vertex start,
11                          Vertex end)
12   {
13     start.addAdjacent(end);
14     return (EdgeIfc) start;
15   }
16
17   /* More source code ... */
18 }
```

Inter-Feature Code Clones

Derived from: Duplicated Code

Example: Graph Product Line (FOP)

Feature *BFS*

```
1  public class Graph
2  {
3    public void GraphSearch(Workspace w)
4    {
5      VertexIter itr = getVertices();
6      /* more source code... */
7      for (itr=getVertices();
8           itr.hasNext();) {
9        Vertex v = itr.next();
10       if (!v.visited) {
11         w.nextRegionAction(v);
12         v.nodeSearch(w);
13       }
14     }
15   }
16   /* more methods... */
17 }
```

Feature *DFS*

```
1  public class Graph
2  {
3    public void GraphSearch(Workspace w)
4    {
5      VertexIter itr = getVertices();
6      /* more duplication... */
7      for (itr=getVertices();
8           itr.hasNext();) {
9        Vertex v = itr.next();
10       if (!v.visited) {
11         w.nextRegionAction(v);
12         v.nodeSearch(w);
13       }
14     }
15   }
16   /* other methods... */
17 }
```

Interlude: Refactoring Inter-Feature Code Clones

1. Find target for common code

```
1  public class Graph {
2    void search(Workspace w) {
3      VertexIter itr = getVertices();
4      if (!itr.hasNext())
5        return;
6      while (itr.hasNext()) {
7        Vertex v = itr.next();
8        v.init_vertex(w);
9      }
10     /* More common code... */
11   }
12 }
```

➥ New feature *Search*

Interlude: Refactoring Inter-Feature Code Clones

1. Find target for common code

```
1  public class Graph {
2    void search(Workspace w) {
3      VertexIter itr = getVertices();
4      if (!itr.hasNext())
5        return;
6      while (itr.hasNext()) {
7        Vertex v = itr.next();
8        v.init_vertex(w);
9      }
10     /* More common code... */
11   }
12 }
```

➥ New feature *Search*

2. Delete code duplication in *BFS*, *DFS*

```
1 public class Graph {
2   /* BFS code... */
3 }
```

➥ Feature *BFS'*

```
1 public class Graph {
2   /* DFS code... */
3 }
```

➥ Feature *DFS'*

Interlude: Refactoring Inter-Feature Code Clones

1. Find target for common code

```
1  public class Graph {
2    void search(Workspace w) {
3      VertexIter itr = getVertices();
4      if (!itr.hasNext())
5        return;
6      while (itr.hasNext()) {
7        Vertex v = itr.next();
8        v.init_vertex(w);
9      }
10     /* More common code... */
11   }
12 }
```

➥ New feature *Search*

2. Delete code duplication in *BFS*, *DFS*

```
1 public class Graph {
2   /* BFS code... */
3 }
```

➥ Feature *BFS'*

```
1 public class Graph {
2   /* DFS code... */
3 }
```

➥ Feature *DFS'*

3. Modify FM

$$FM' := FM\{BFS \mapsto BFS', DFS \mapsto DFS'\} \wedge ((BFS' \vee DFS') \rightarrow Search)$$

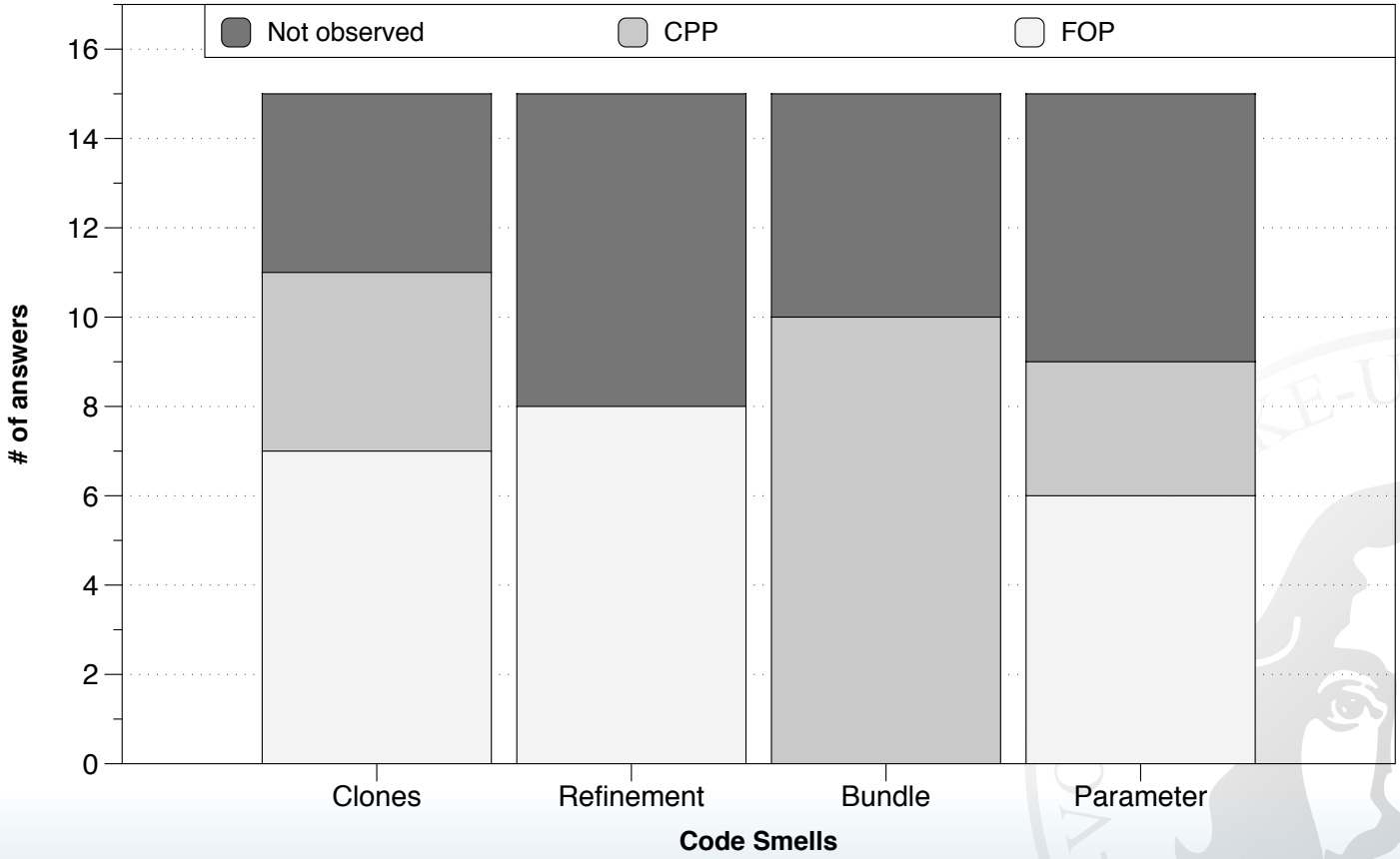
➥ ***Variant-preserving refactoring***

Evaluation: Setup

- Questionnaire
- Objective:
  - ***Q1:*** *Do our proposed smells exist in the design and implementation of SPLs?*
  - ***Q2:*** *Are our smells problematic with respect to different aspects of SPL development?*
- Participants:
  - *International meeting on feature oriented software development*, held at Schloß Dagstuhl in May 2014 (`http://www.fosd.de/meeting2014`)
  - Mostly academic audience (from PhD student to full professor)
  - Experience with SPL implementation techniques
- Received 15 complete response sets

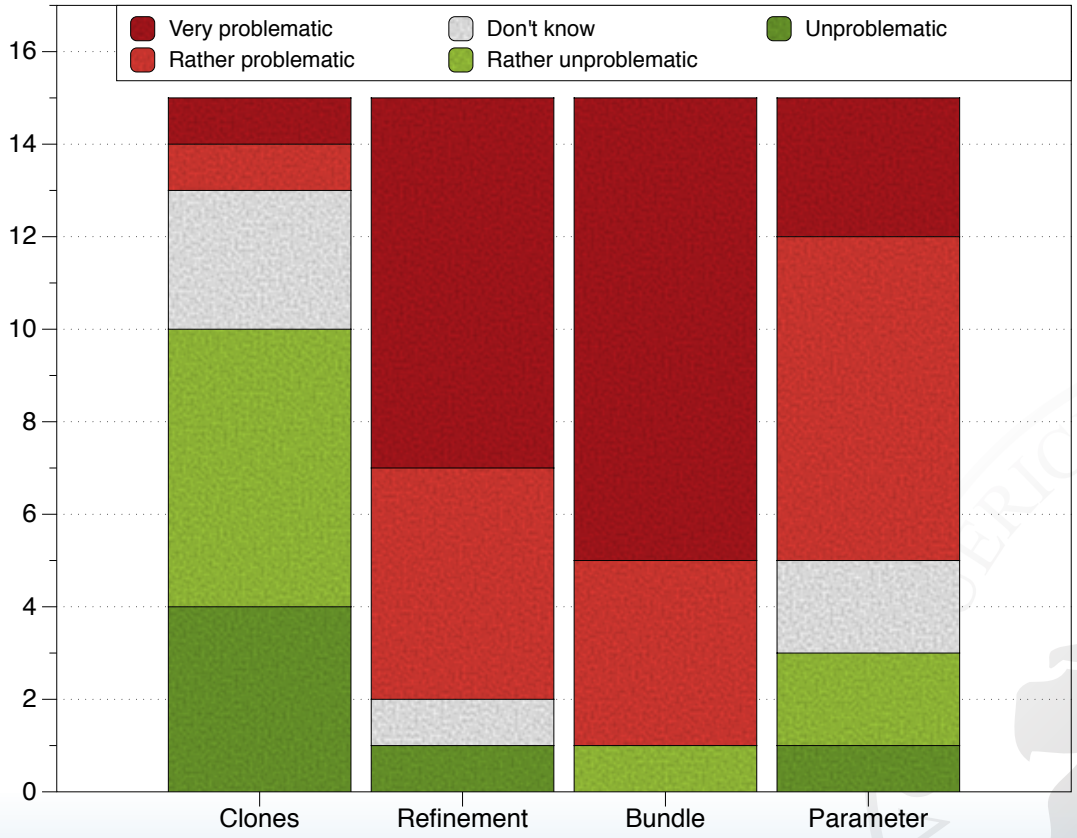
Evaluation: Questionnaire Results (1)

Q1: *Do our proposed smells exist in the design and implementation of SPLs?*



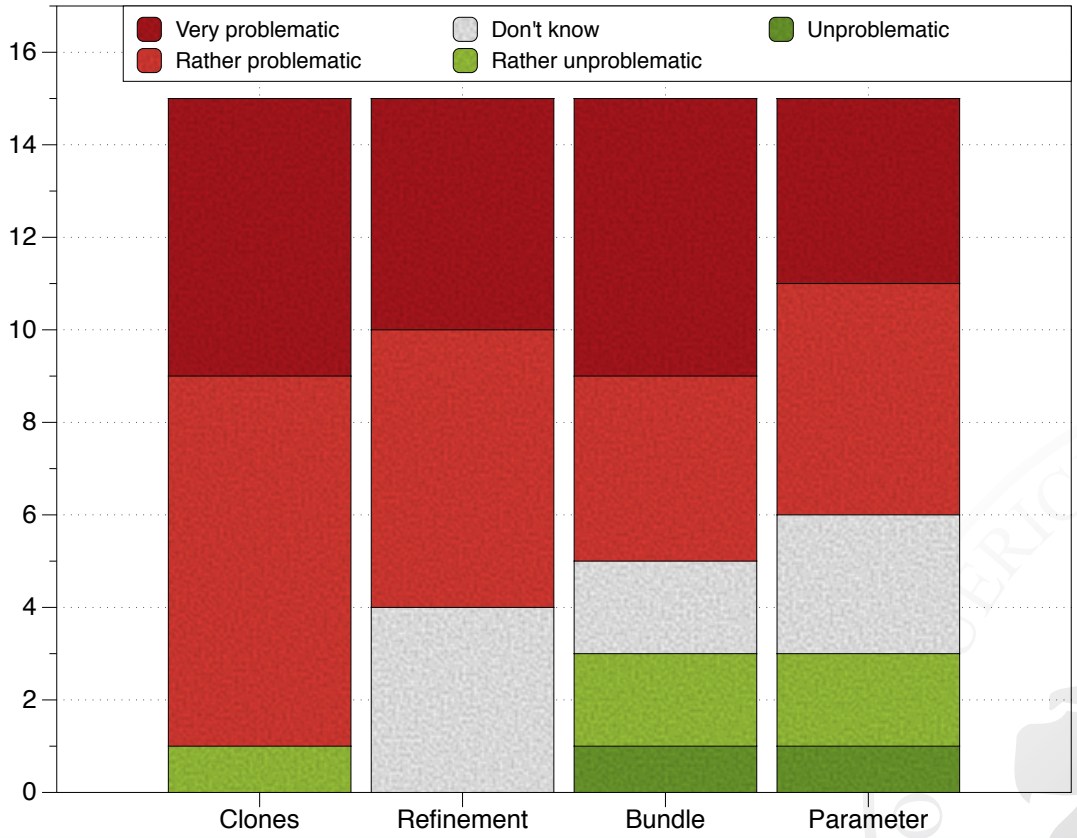
Evaluation: Questionnaire Results (2)

Q2: *Are our smells problematic with respect to program comprehension?*



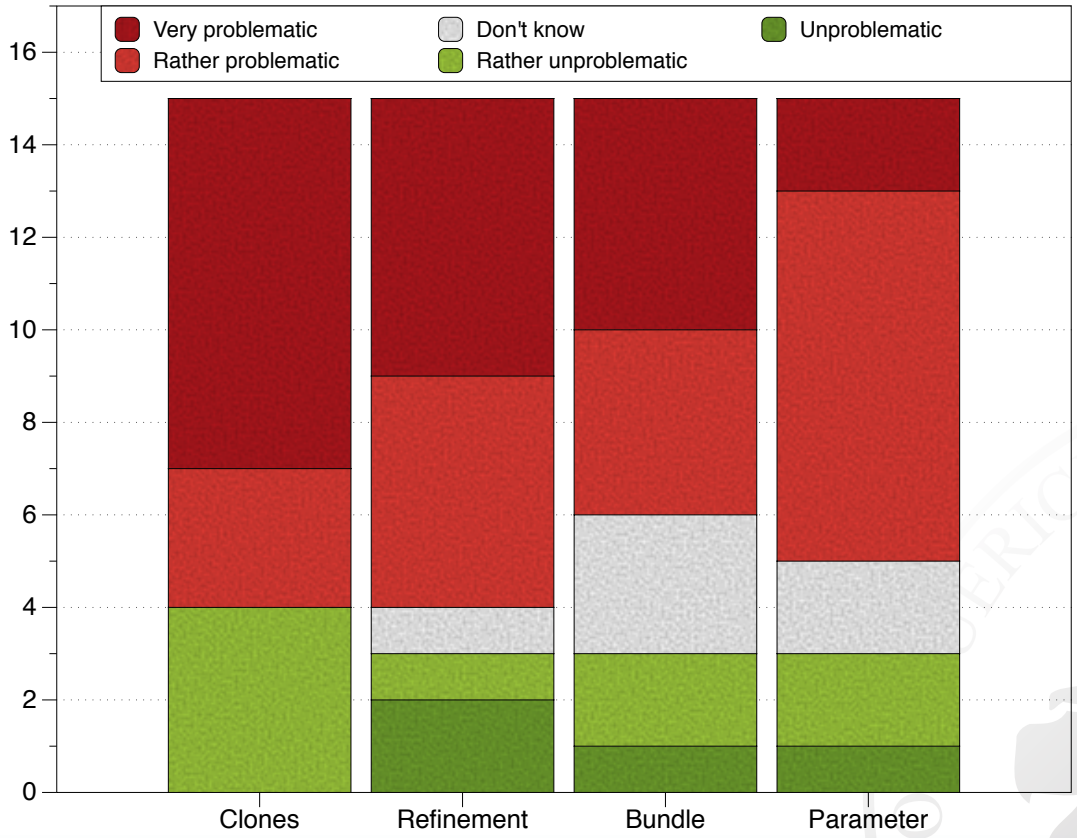
Evaluation: Questionnaire Results (3)

Q2: *Are our smells problematic with respect to maintenance?*



Evaluation: Questionnaire Results (4)

Q2: *Are our smells problematic with respect to evolution?*



Evaluation: Questionnaire Results (5)

[Inter-Feature Code Clones] "Our industry partner is struggling with inter-feature code clones due to a lack of awareness. . . . "

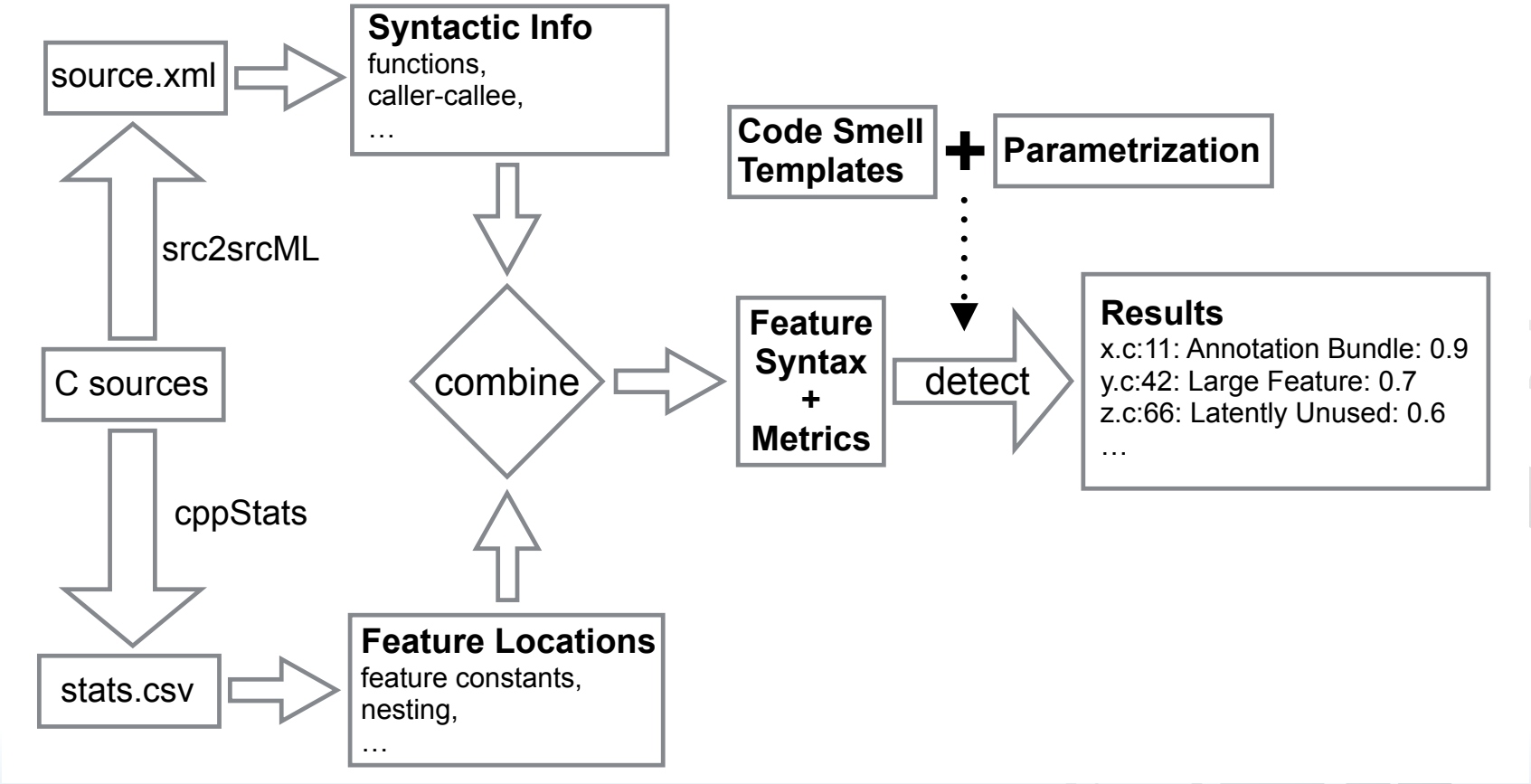
[Annotation Bundle] ". . . in Linux, I have observed that in some cases a lot of #ifdefs are used in a method and some of them are nested making the method longer and more complicated."

Conclusion

- Source code has structure
  ➥ Code smells describe bad ways to structure code
- Variability mechanism adds additional dimension
  ➥ New opportunities to chose bad structures
- **Goal:** Make traditional code smells *variability-aware*
- Considered effect of variability mechanisms on established single system code smells
- Questionnaire responses indicate that presented smells . . .
  - occur "in the wild"
  - are considered problematic for certain aspects (program comprehension, maintainability, evolvability)

Future Work: Detection (1)

Variability-aware code smell detection in C code with `#ifdefs`



Future Work: Detection (2)

Example: Annoation Bundle

1. Identify function locations
2. Metrics:
   - Lines of code (loc)
   - Lines of feature code ($lofc$)
   - Number of feature constants ($nofc$)
   - Nesting depth of annotations (nd)
   - ...
3. Combine metrics, applying weights $w_1, w_2, \ldots, w_n$

Example detector:

$$bundleness(f) = w_1 * lofc(f)/loc(f) + w_2 * nofc(f) + w_3 * nd(f)$$

More Future Work

- More variability-aware smells:
  - Adapt more single-system smells to SPL context
  - What about anti-patterns, architectural patterns?
    How would you describe a god feature?
  - Incorporate other aspects, e. g. organizational structure, coordination requirements
- Smell removal ➥ variability-aware refactorings
- Link variability-aware code smells to actual maintenance problems
  - Repository mining,
  - Mining issue trackers,
  - . . . ?