INF FAKULTÄT FÜR INFORMATIK

Variability-Aware Code Smells

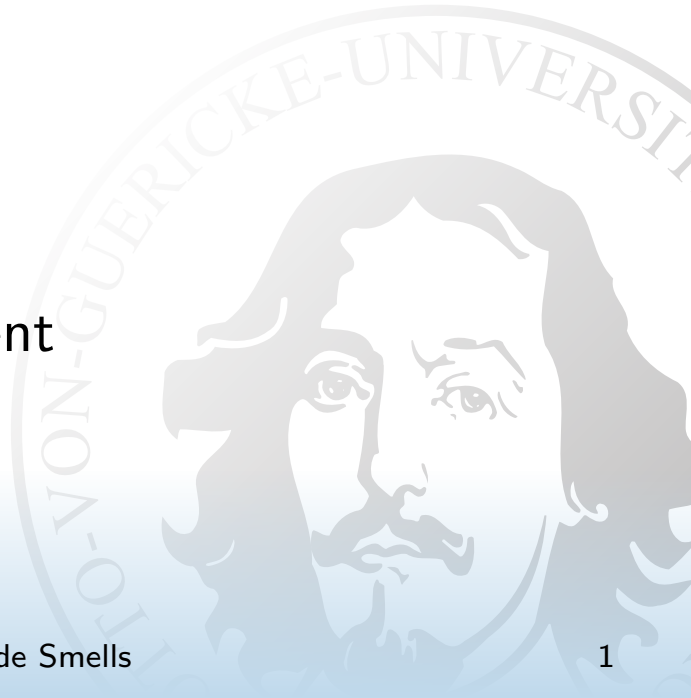
Wolfram Fenske,[1] Sandro Schulze[2]

Monday 5th May, 2014

[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
  - Bug fixing
  - Extension
- Indicator that structure needs improvement



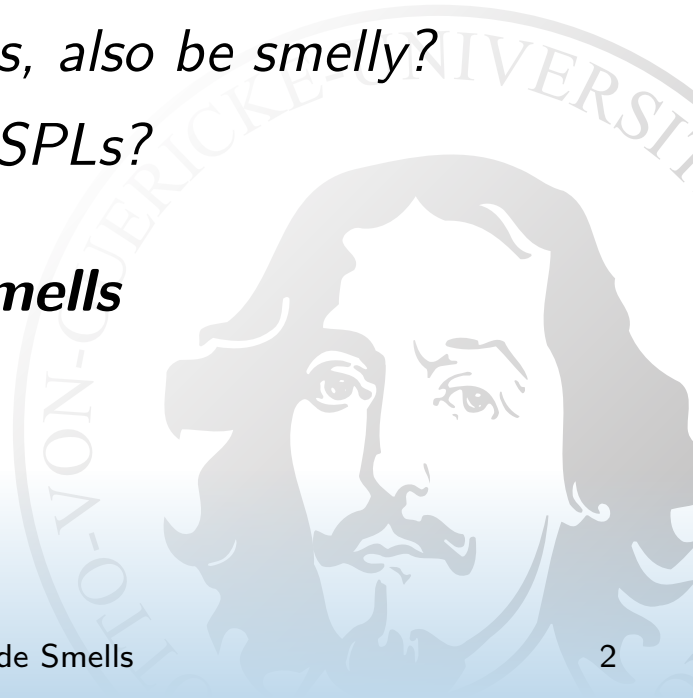
Code Smells (2)

- So far focus on single systems
- Little systematic treatment of smells in SPL context

➥ Are code smells different in SPLs? If so, how?

- **RQ 1:** *How does variability affect existing code smells?*
- **RQ 2:** *Can SPL entities, such as features, also be smelly?*
- **RQ 3:** *Are there code smells specific to SPLs?*

➥ ***Variability-aware code smells***



Inter-Feature Code Clones

Derived from: Duplicated Code

Example: Graph Product Line (FOP)

```
public class Graph {
  void search(Workspace w) {
    VertexIter itr =
        getVertices();
    if (!itr.hasNext())
      return;
    while (itr.hasNext()) {
      Vertex v = itr.next();
      v.init_vertex(w);
    }
    /* More source code... */
  }
  /* More source code... */
}
```

Feature *BFS*

```
public class Graph {
  void search(Workspace w) {
    VertexIter itr =
        getVertices();
    if (!itr.hasNext())
      return;
    while (itr.hasNext()) {
      Vertex v = itr.next();
      v.init_vertex(w);
    }
    /* More duplication... */
  }
  /* Other source code... */
}
```

Feature *DFS*

Switch Statements with Optional Cases

Derived from: Switch Statements

Example: Mozilla NSS Library

```
static CK_RV
pk11_handlePublicKeyObject(PK11Session *
      session, PK11Object *object,
      CK_KEY_TYPE key_type)
{
  /* More code... */
  switch (key_type) {
  /* Handle other cases... */
  case CKK_DH:
    /* Handle CKK_DH... */
    break;
#ifdef NSS_ENABLE_ECC
  case CKK_EC:
    /* Start handling CKK_EC... */
    pubKeyAttr = CKA_EC_POINT;
    derive = CK_TRUE;     /* for ECDH */
    verify = CK_TRUE;     /* for ECDSA */
    encrypt = CK_FALSE;
    recover = CK_FALSE;
    wrap = CK_FALSE;
    break;
#endif /* NSS_ENABLE_ECC */
  default:
    /* Handle default case... */
}
```

```
NSSLOWKEYPublicKey *
pk11_GetPubKey(PK11Object *object,
      CK_KEY_TYPE key_type, CK_RV *crvp)
{
  /* More code... */
  switch (key_type) {
  /* Other code for other cases... */
  case CKK_DH:
    /* Other code to handle CKK_DH... */
    break;
#ifdef NSS_ENABLE_ECC
  case CKK_EC:
    /* Start handling CKK_EC... */
    if (EC_FillParams(arena, &pubKey->u.
          ec.ecParams.DEREncoding, &
          pubKey->u.ec.ecParams) !=
          SECSuccess) break;
    crv = pk11_Attribute2SSecItem(arena
          ,&pubKey->u.ec.publicValue,
          object, CKA_EC_POINT);
    break;
#endif /* NSS_ENABLE_ECC */
  default:
    /* Other default code... */
}
```

Annotation Bundle

Derived from: Long Method

Example: Firefox 28, `memory/mozjemalloc/jemalloc.c`

```
/* More code... */
for (k = 0; k < nreps; k++) {
  switch (opts[j]) {
  case 'a':
    opt_abort = false;
    break;
  case 'A':
    opt_abort = true;
    break;
  case 'b':
#ifdef MALLOC_BALANCE
    opt_balance_threshold >>= 1;
#endif
    break;
  case 'B':
#ifdef MALLOC_BALANCE
    if (opt_balance_threshold == 0)
      opt_balance_threshold = 1;
    else if ((opt_balance_threshold <<
          1) > opt_balance_threshold)
      opt_balance_threshold <<= 1;
#endif
    break;
```

```
/* ...continued from left */
#ifdef MALLOC_FILL
#  ifndef MALLOC_PRODUCTION
  case 'c':
    opt_poison = false;
    break;
  case 'C':
    opt_poison = true;
    break;
#  endif
#endif
  case 'f':
    opt_dirty_max >>= 1;
    break;
  case 'F':
    if (opt_dirty_max == 0)
      opt_dirty_max = 1;
    else if ((opt_dirty_max << 1) != 0)
      opt_dirty_max <<= 1;
    break;
  /* More case clauses... */
  }
/* Even more code... */
```

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 SemanticException {
    // layers extend this method for AST processing
  }
}
```

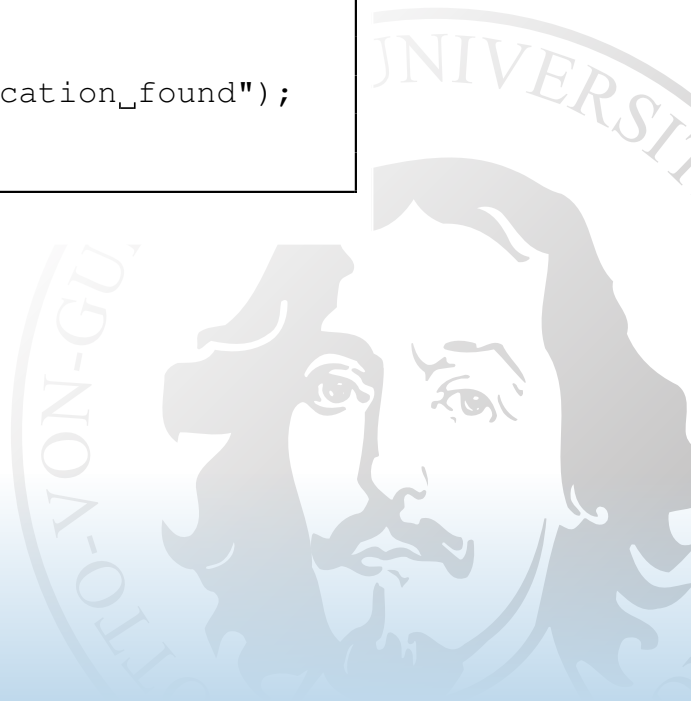
Long Refinement Chain

Derived from: Long Method

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

```
class Main  { // Feature 'dmain'
  public static void process(Model root) throws SemanticException {
  }
}
```

```
class Main  { // Feature 'fillgs'
  public static void process(Model root) throws SemanticException {
    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

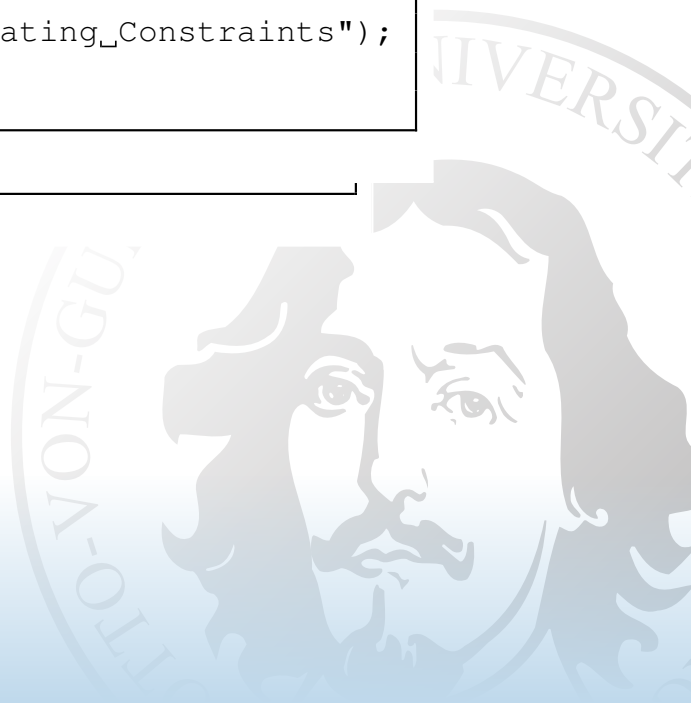
Derived from: Long Method

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

```
class Main  { // Feature 'dmain'
  public static void process(Model root) throws SemanticException {
  }
}
```

```
class Main  { // Feature 'fillgs'
  public static void process(Model root) throws SemanticException {
  }
}
```

```
class Main  { // Feature 'propgs'
  public static void process(Model root) throws SemanticException {
    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

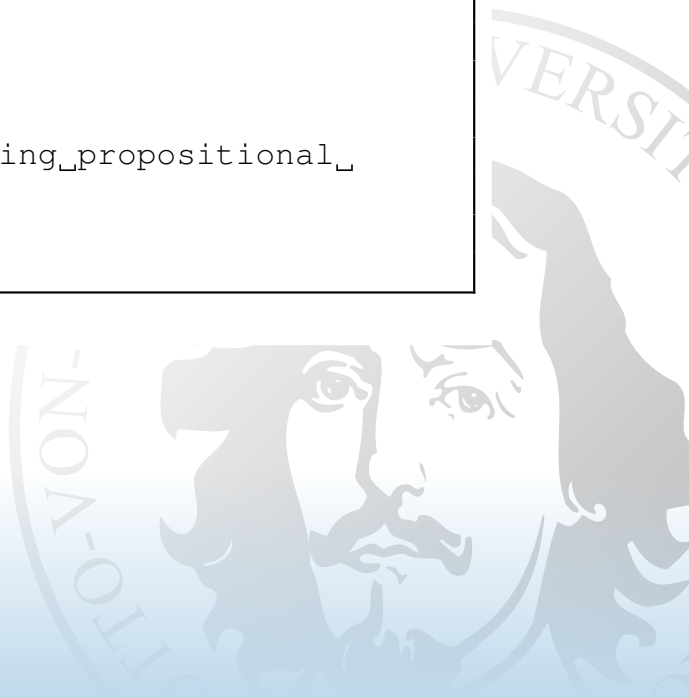
Example: GUIDSL (FOP), method `Main#process(Model)`

```
class Main  { // Feature 'dmain'
  public static void process(Model root) throws SemanticException {
  }
}
```

```
class Main  { // Feature 'fillgs'
  public static void process(Model root) throws SemanticException {
  }
}
```

```
class Main  { // Feature 'propgs'
  public static void process(Model root) throws SemanticException {
  }
}
```

```
class Main  { // Feature 'formgs'
  public static void process(Model root) throws SemanticException {
    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 SemanticException {
  }
}
```

```
class Main  { // Feature 'fillgs'
  public static void process(Model root) throws SemanticException {
  }
}
```

```
class Main  { // Feature 'propgs'
  public static void process(Model root) throws SemanticException {
  }
}
```

```
class Main  { // Feature 'formgs'
  public static void process(Model root) throws SemanticException {
  }
}
```

```
class Main  { // Feature 'clauselist'
  public static void process(Model root) throws SemanticException {
    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 SemanticException {
  }
}
```

```
class Main  { // Feature 'fillgs'
  public static void process(Model root) throws SemanticException {
  }
}
```

```
class Main  { // Feature 'propgs'
  public static void process(Model root) throws SemanticException {
  }
}
```

```
class Main  { // Feature 'formgs'
  public static void process(Model root) throws SemanticException {
  }
}
```

```
class Main  { // Feature 'clauselist'
  public static void process(Model root) throws SemanticException {
  }
}
```

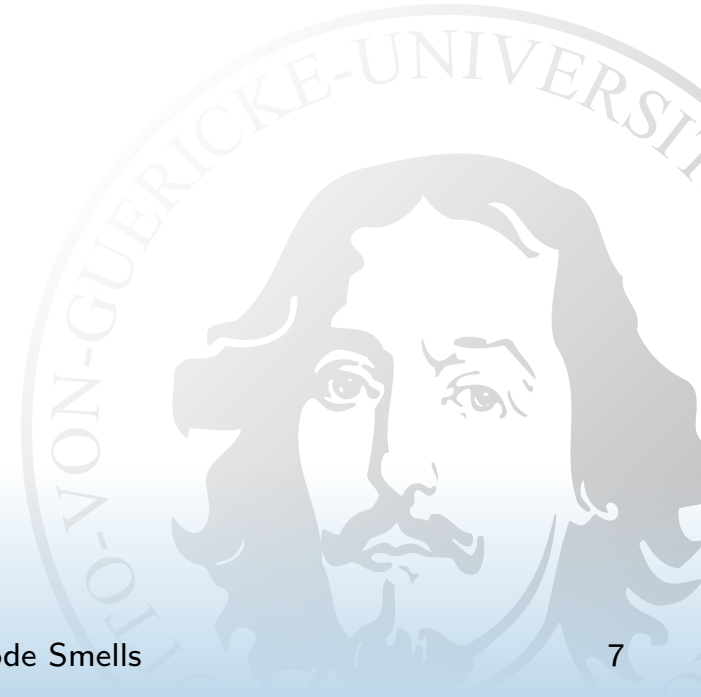
```
class Main  { // Feature 'modelopts'
  public static void process(Model root) throws SemanticException {
    original(m);
    if (modelMode) {
      try { harvestInfo(); }
      catch (IOException e) {
        JOptionPane.showMessageDialog(null,
        "Model Harvesting Error -- see command line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

Interlude: Large Class

History of a Large Class

Class C_{t0}

```
class C {
  void m1() {
    /* ... */
  }
  void m2() {
    /* ... */
  }
}
```



Interlude: Large Class

History of a Large Class

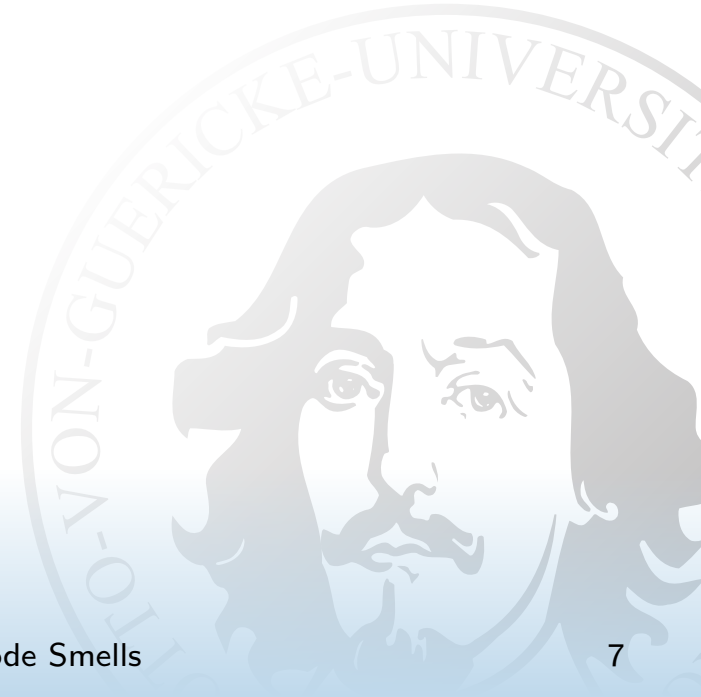
Class C_{t0}

```
class C {
  void m1() {
    /* ... */
  }
  void m2() {
    /* ... */
  }
}
```

$\Rightarrow$

Class C_{t1}

```
class C {
  void m1() {
    /* ... */
  }
  void m2() {
    /* ... */
  }
  void m3() {
    /* ... */
  }
  void m4() {
    /* ... */
  }
}
```



Interlude: Large Class

History of a Large Class

Class C_{t0}

```
class C {
  void m1() {
    /* ... */
  }
  void m2() {
    /* ... */
  }
}
```

$\Rightarrow$

Class C_{t1}

```
class C {
  void m1() {
    /* ... */
  }
  void m2() {
    /* ... */
  }
  void m3() {
    /* ... */
  }
  void m4() {
    /* ... */
  }
}
```

$\Rightarrow$

Class C_{t2}

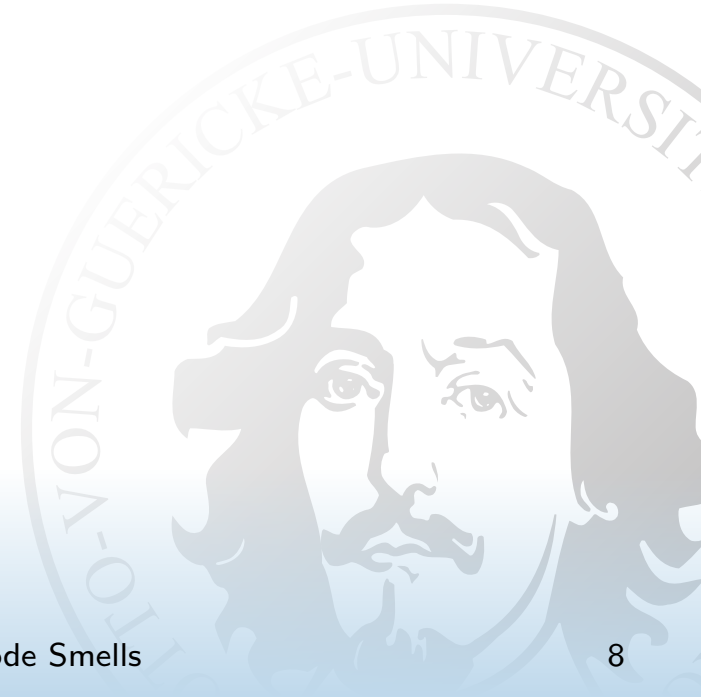
```
class C {
  void m1() {
    /* ... */
  }
  void m2() {
    /* ... */
  }
  void m3() {
    /* ... */
  }
  void m4() {
    /* ... */
  }
  void m5() {
    /* ... */
  }
  void m6() {
    /* ... */
  }
}
```

Large Feature (2)

Derived from: Large Class

Feature F_{t0}

```
class Class1 {
  /* ... */
}

class Class2 {
  /* ... */
}
```



Large Feature (2)

Derived from: Large Class

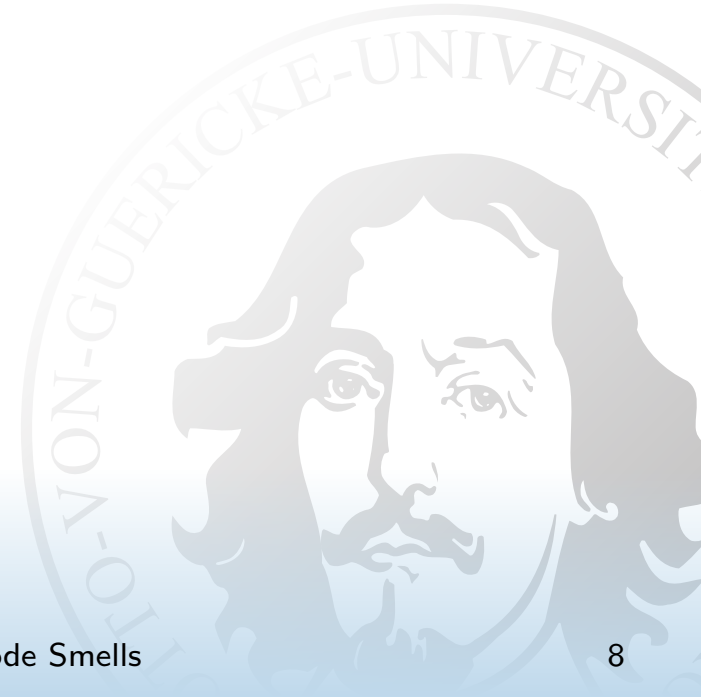
Feature F_{t0}

```
class Class1 {
  /* ... */
}

class Class2 {
  /* ... */
}
```

$\Rightarrow$

Feature F_{t1}

```
class Class1 {
  /* ... */
}

class Class2 {
  /* ... */
}

class Class3 {
  /* ... */
}

refines
class Class4 {
  /* ... */
}
```



Large Feature (2)

Derived from: Large Class

Feature F_{t0}

```
class Class1 {
  /* ... */
}

class Class2 {
  /* ... */
}
```

$\Rightarrow$

Feature F_{t1}

```
class Class1 {
  /* ... */
}

class Class2 {
  /* ... */
}

class Class3 {
  /* ... */
}

refines
class Class4 {
  /* ... */
}
```

$\Rightarrow$

Feature F_{t2}

```
class Class1 {
  /* ... */
}

class Class2 {
  /* ... */
}

class Class3 {
  /* ... */
}

refines
class Class4 {
  /* ... */
}

class Class5 {
  /* ... */
}

refines
class Class6 {
  /* ... */
}
```

Latently Unused Parameter

Derived from: Speculative Generality & Long Parameter List

Example: Graph Product Line (FOP)

```
public class Graph {
  /* More source code ... */

  public void addAnEdge(Vertex start,
      Vertex end, int weight)
  {
    addEdge(start, end, weight);
  }

  public void addEdge(Vertex start,
       Vertex end, int weight)
  {
    addEdge(start, end);
    start.addWeight(weight);
    /* More source code ... */
  }

  /* More source code ... */
}
```

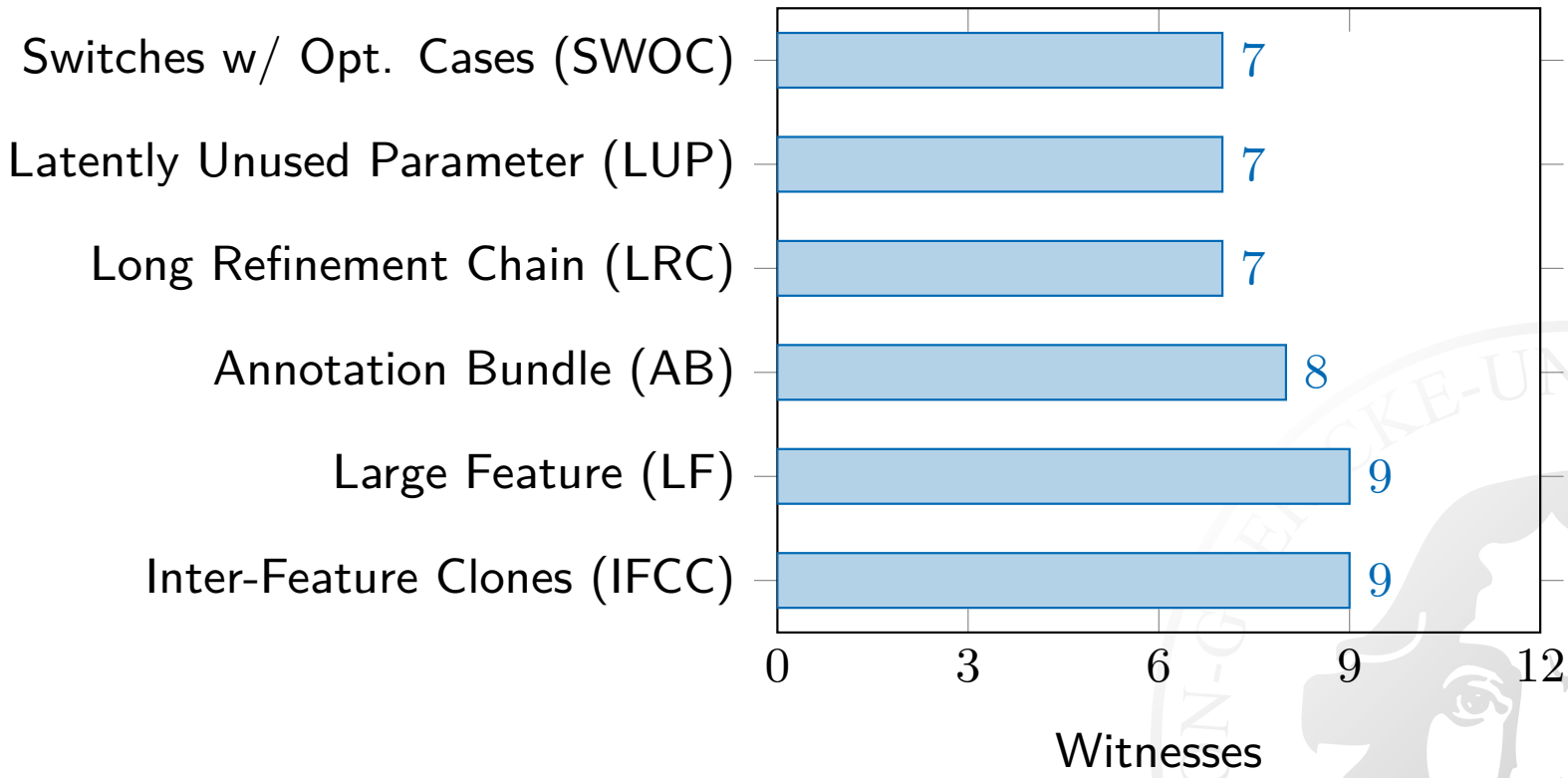
Feature *WeightedOnlyVertices*

```
public class Graph {
  /* More source code ... */

  public void addAnEdge(Vertex start,
      Vertex end, int weight)
  {
    addEdge(start, end);
  }

  public EdgeIfc addEdge(Vertex start,
       Vertex end) {
    start.addAdjacent(end);
    return (EdgeIfc) start;
  }

  /* More source code ... */
}
```

Feature *DirectedOnlyVertices*

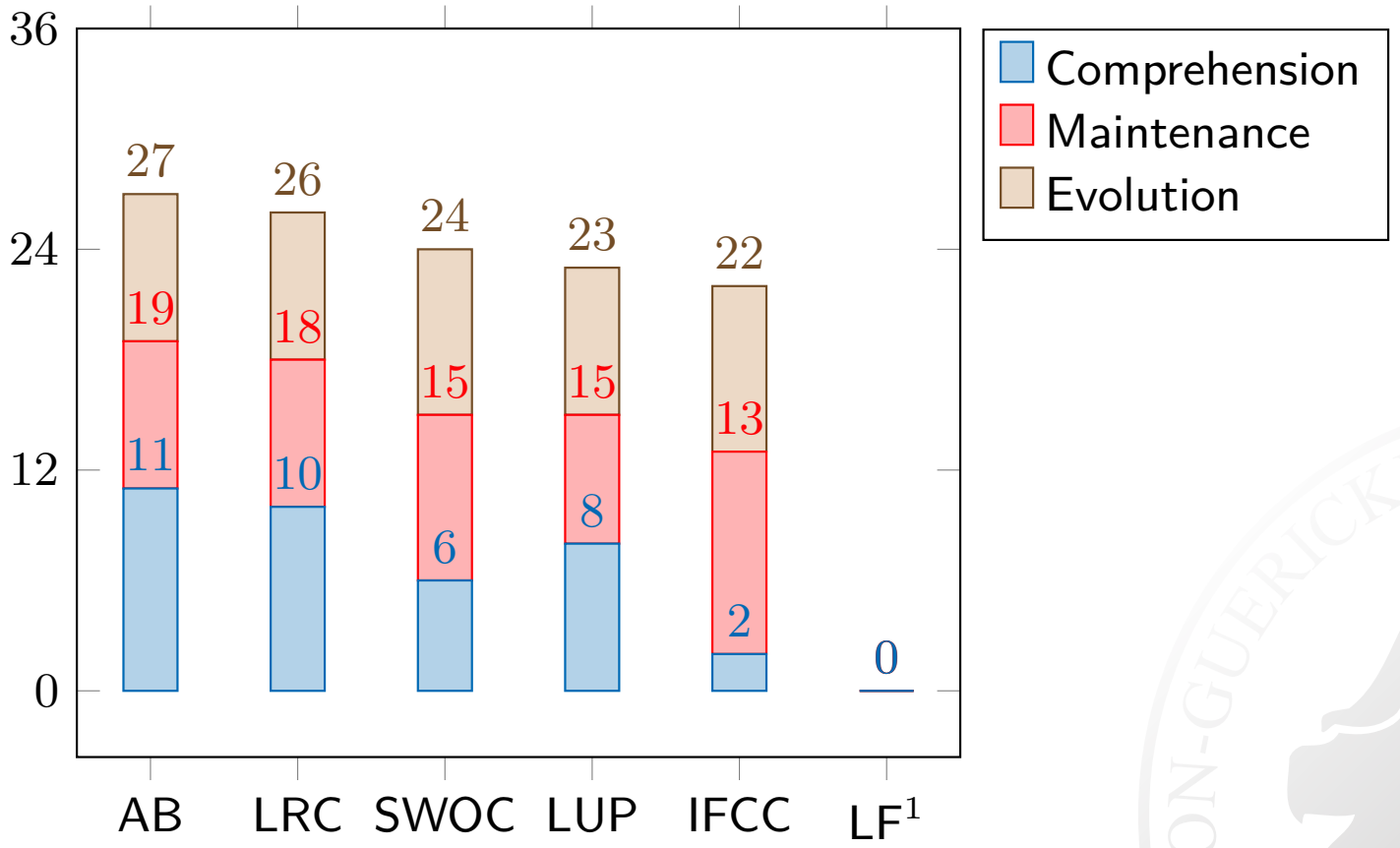
Preliminary Questionnaire Results (1)

Q1: "Have you seen the smell anywhere before?"



Preliminary Questionnaire Results (2)

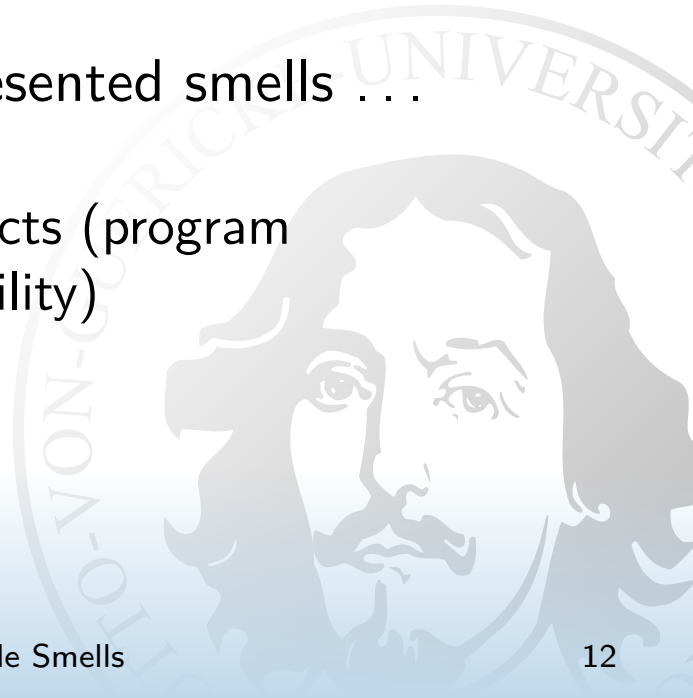
Q2: "How problematic is the smell with respect to . . . "



[1] missing data

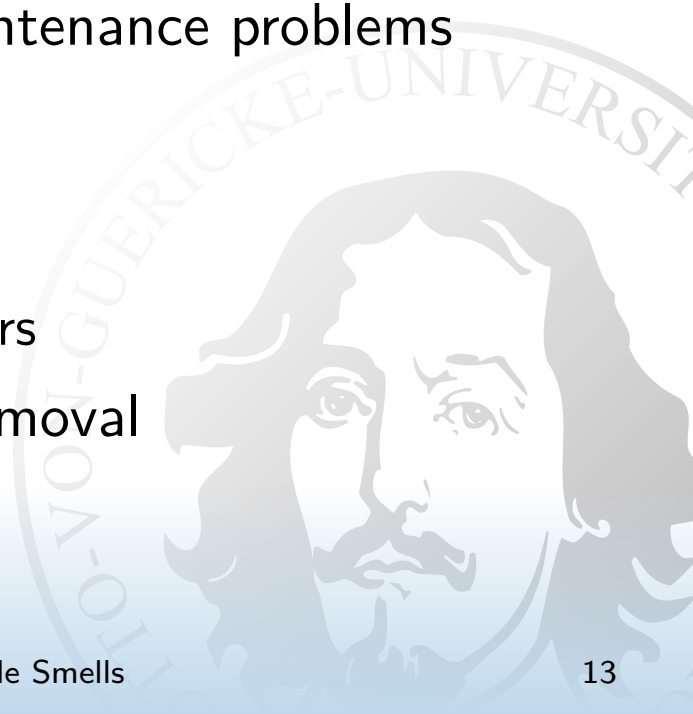
Conclusion

- Two ways to adapt single-system smells to SPL context
  1. Consider variability of involved fields, statements, etc.
  2. Adapt `class`-centric smells to features
- SPL implementation approaches affect smell appearance differently
- Questionnaire responses indicate that presented smells . . .
  - occur “in the wild”
  - are seen as problematic for certain aspects (program comprehension, maintainability, evolvability)



How to Continue?

- Analyze questionnaire results in depth
- More variability-aware smells:
  - Adapt more single-system smells to SPL context
  - How can anti-patterns be adapted?
  - Find new smells & anti-patterns, unique to SPLs
- Link variability-aware code smells to maintenance problems (empirical study)?
- Develop metrics & detection tools?
  - Probably useful for empirical studies
  - Probably not so valuable for practitioners
- Variability-aware refactorings for smell removal

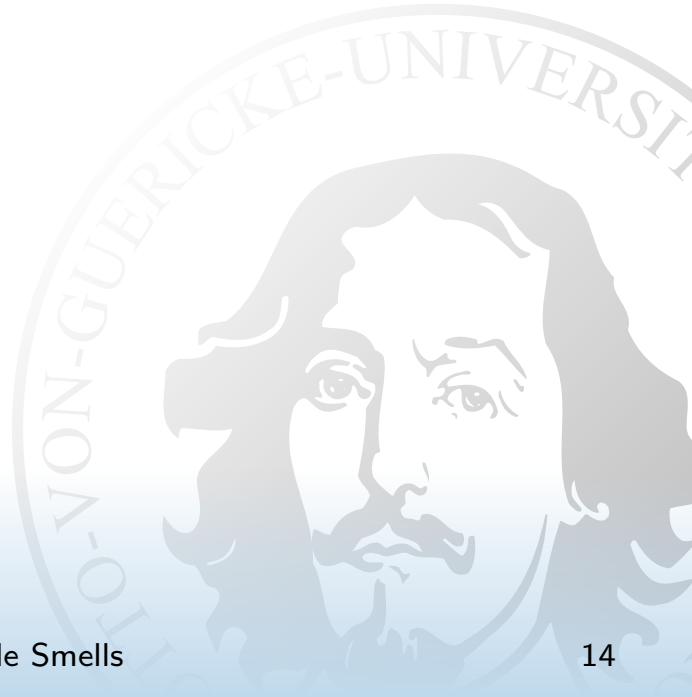


Removing Inter-Feature Code Clones

1. Find new home

```
public class Graph {
  void search(Workspace w) {
    VertexIter itr = getVertices();
    if (!itr.hasNext())
      return;
    while (itr.hasNext()) {
      Vertex v = itr.next();
      v.init_vertex(w);
    }
    /* More common code... */
  }
}
```

New feature *Search*



Removing Inter-Feature Code Clones

1. Find new home

```
public class Graph {
  void search(Workspace w) {
    VertexIter itr = getVertices();
    if (!itr.hasNext())
      return;
    while (itr.hasNext()) {
      Vertex v = itr.next();
      v.init_vertex(w);
    }
    /* More common code... */
  }
}
```

New feature *Search*

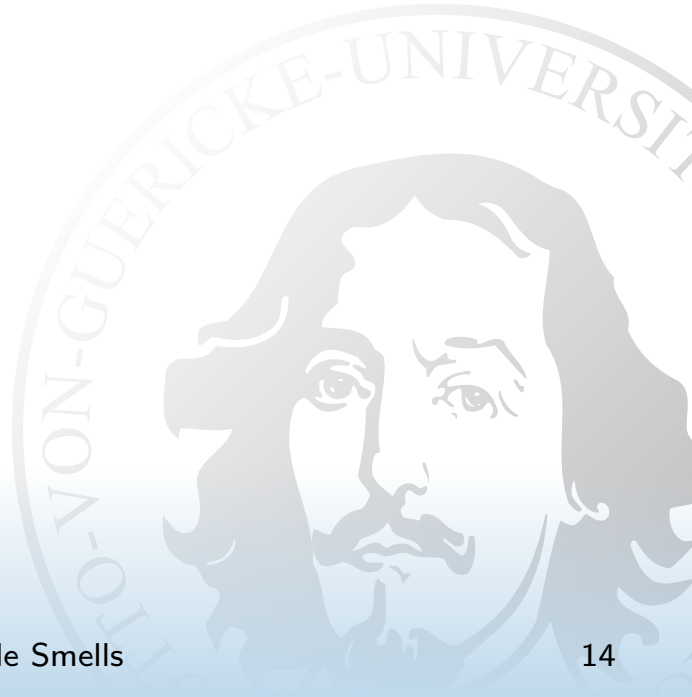
2. Remove duplication

```
public class Graph {
  /* BFS code... */
}
```

Feature *BFS'*

```
public class Graph {
  /* DFS code... */
}
```

Feature *DFS'*



Removing Inter-Feature Code Clones

1. Find new home

```
public class Graph {
  void search(Workspace w) {
    VertexIter itr = getVertices();
    if (!itr.hasNext())
      return;
    while (itr.hasNext()) {
      Vertex v = itr.next();
      v.init_vertex(w);
    }
    /* More common code... */
  }
}
```

New feature *Search*

2. Remove duplication

```
public class Graph {
  /* BFS code... */
}
```

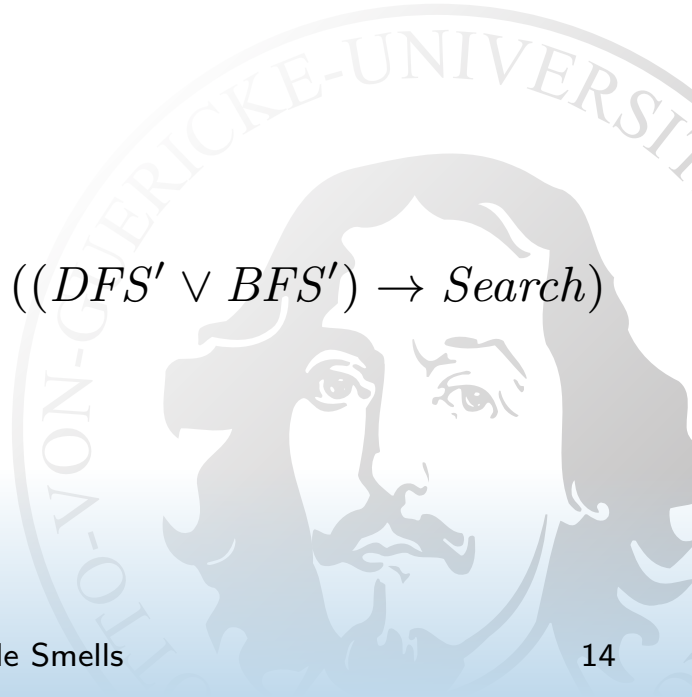
Feature *BFS'*

```
public class Graph {
  /* DFS code... */
}
```

Feature *DFS'*

3. Modify FM

$$FM' := FM \wedge ((DFS' \vee BFS') \rightarrow Search)$$



Switch Statement with Optional Cases in FOP

Listing 1: Switch Statement with Optional Cases in FOP

```
void init(TankManager mgr, int xPos, int yPos, int toolType) {
  original(mgr, xPos, yPos, toolType);
  switch (toolType) {
  case 374:
    original(mgr, xPos * mgr.grain2, yPos * mgr.grain2, 255,
        255, 0, mgr.grain, mgr.grain, toolType);
    break;
  }
}
void init(TankManager mgr, int xPos, int yPos, int toolType) {
  original(mgr, xPos, yPos, toolType);
  switch (toolType) {
  case 371:
    original(mgr, xPos * mgr.grain2, yPos * mgr.grain2, 100,
        149, 237, mgr.grain, mgr.grain, toolType);
    break;
  }
}
```