SMA:
Software Modeling and Analysis

Practical Session

Week 09

Assignment 08

Discussion

A08 - Exercise 01 | Code smells

a) Choose two different code smells and explain them.

Class Data Should be Private, Complex Class, Spaghetti Code, ...

b) What is the fundamental problem in developers code smell perception?

The perception is a subjective matter. *Anaïs Nin* sums it up quite nicely: "We don't see things as they are, we see things as we are."

c) What is “association rule mining” in the context of HIST?

Association rules represent correlations between subsets of methods in the same class that frequently change together. The discovery of all association rules within one or more projects is then called "association rule mining".

A08 - Exercise 02 | Test code smells

a) Choose one test code smell and explain it.

Eager Test, Ignored Test, Sleepy Test, ...

b) Find and explain the test code smell in the test below. (2 pts)

```
public void testDataIsVariable() throws Throwable {
    JSTerm term = new JSTerm();
    term.makeVariable();
    term.add((Object) "");
    jSTerm0.matches(jSTerm0);
    assertEquals(false, term.isGround());
    assertEquals(true, term.isVariable());
}
```

multiple assert statements in a single test method might cause unexpected outcomes
= code is harder to debug and maintain

A08 - Exercise 03 | Detection of eager tests

Extract all `JUnit3` tests from `modelWeka` that suffer from the “Eager Test” code smell.

```
tests := modelWeka allModelMethods select: #isJUnit3Test.
eagerTests := tests select: [ : m |
        | asserts astNode |
        asserts := OrderedCollection new.
        astNode := m gtASTNode.
        astNode
                ifNotNil: [ astNode
                        allNodesOfType: JavaMethodInvocationNode
                        do: [ :node |
                                (node name value beginsWith: 'assert')
                                        ifTrue: [ asserts add: node ] ] ].
        asserts size > 1 ].
```

Assignment 09

Preview

A09 - Exercise 01 | Theory (6 pts)

a) What is the difference between static and dynamic analysis?

b) Suppose you want to analyze the code that a method downloads arbitrarily from the internet. Can you perform such an inspection with static analyses? Why?

c) Suppose you want to analyze the code that a method downloads arbitrarily from the internet. Can you perform such an inspection with dynamic analyses? Why?

A09 - Exercise 01 | Theory

d) Choose a static analysis scenario where it is crucial to have no false positives, but false negatives can be accepted. Explain.

e) Choose a statically-typed language, and briefly describe what makes it statically-typed.

f) Choose a dynamically-typed language, and briefly describe what makes it dynamically-typed.

g) Is the collection of variable name declarations in the method body of Java's `String.println(Strings)` intraprocedural or interprocedural? Explain why.

A09 - Exercise 02 | Control flow graphs

a) Draw a CFG by hand (or with a flow chart tool) for the following code block: (1.5 pts)

```
int a = 2;
int b = 3;
if (a == 2) {
   b = a + b;
} else {
   b = a++;
}
return a + b;
```

A09 - Exercise 02 | Control flow graphs

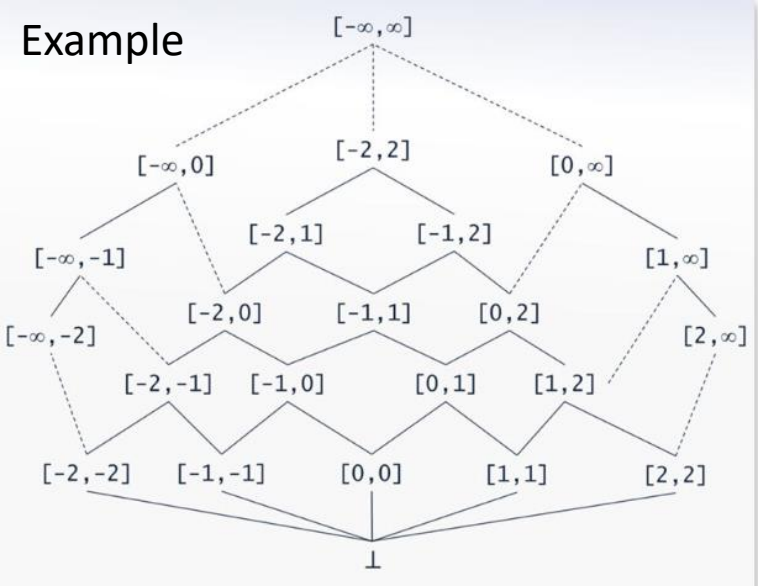
b) Draw a CFG by hand (or with a flow chart tool) for the following code block: (2.0 pts)

```
public boolean addBalances(int money) {
  if (money > 0) {
    acquire(lock);
    int newMoneyValue = this.money + money;
    if (newMoneyValue < MAX_MONEY) {
      this.money = newMoneyValue;
      release(lock);
      return true;
    } else {
      release(lock);
      return false;
    }
  } else {
    return false;
  }
}
```

A09 - Exercise 02 | Control flow graphs

c) What is the cyclomatic complexity of both code blocks, i.e., from task a) and task b)? You should use the formula from A08. (1 pt)

d) Create the interval CFG for both code blocks, i.e., from task a) and task b). You can find an example CFG at the bottom of slide 20 (page 32). (2 pts)



A09 - Exercise 03 | Template methods (6 pts BONUS)

Find all template methods in `modelWeka` with the help of `#gtASTNodes`, and plot them with `GtMondrian`.

NB: Template methods are abstract methods (in abstract classes).

Step 1: Select all methods that are in the namespace scope `weka::core`.

Step 2: Of those methods, find those that are abstract. Use `gtASTNode` for the AST traversal.

Step 3: Visualize the found methods in Step 2 with `GtMondrian`.

Use the following parameters:
shape type: `BlElement` with a size that represents `#children` of each method
shape geometry: `BlCircle`
shape background: all methods that have more than three elements in `children` should be in red, the others in gray

A09 - Exercise 03 | Template methods

The final plot...