Revisiting Context-Based Code Smells Prioritization: On Supporting Referred Context

Natthawute Sae-Lim, Shinpei Hayashi, and Motoshi Saeki

Department of Computer Science
Tokyo Institute of Technology
INTRODUCTION
**Code smell**[^1]

An indicator of a design flaw or a problem in the source code

- One of the factors that cause technical debt 😞
- Increases code component’s fault-proneness 😞

---

```java
public class Employee...
private void printSalary(){
    int salary;
    if (wHours > 40)
        salary = (40*1000)+((wHours-40)*1500);
    else
        salary = wHours*1000;
    println("Salary: " + salary);
}
private void printIncomeTax(){
    int incomeTax;
    int salary;
    if (wHours > 40)
        salary = (40*1000)+((wHours-40)*1500);
    else
        salary = wHours*1000;
    incomeTax = salary*0.08;
    println("Tax: " + incomeTax);
}
private int getSalary(){
    int salary;
    if (wHours > 40)
        salary = (40*1000)+((wHours-40)*1500);
    else
        salary = wHours*1000;
    return salary;
}
```

**Refactoring**

**Duplicated Code**

(Extract Method)

```
public class Employee...
private void printSalary(){
    println("Salary: " + getSalary());
}
private void printIncomeTax(){
    int incomeTax = getSalary() * 0.08;
    println("Tax: " + incomeTax);
}
private int getSalary(){
    int salary;
    if (wHours > 40)
        salary = (40*1000)+((wHours-40)*1500);
    else
        salary = wHours*1000;
    return salary;
}
```

The number of code smell is overwhelming
Code Smells Prioritization

  Vidal et al.

- [SBSE 2013] Prioritization of Code Anomalies based on Architecture Sensitiveness
  Arcoverde et al.

  Fontana et al.

- [ICPC 2016] Context-Based Code Smells Prioritization for Prefactoring
  Sae-Lim et al.

- [ICSE 2016] Technical Debt Prioritization using Predictive Analytics
  Codabux et al.
CONTEXT-BASED CODE SMELLS PRIORITIZATION
I need to implement feature X in method A()

Problem:

Results from existing smell detector do not fit in this situation
Our technique

Smells that are relevant to developers’ context
Bugzilla

Change descriptions

Source code

Impact analysis

TraceLab[1]

Code smell detection

inFusion[2]

Scoring

List of modules

List of smells

Prioritized smells

Empirical Study

Our technique can prioritize code smells occurring in the modules that are going to be modified.

Conclusion
Software change process[1]

1. Initiation
2. Concept Location
3. Impact Analysis
4. Prefactoring
5. Actualization
6. Postfactoring
7. Conclusion

Mylyn

Task and application lifecycle management (ALM) framework for Eclipse.
Developer selects text in editor

```
<InteractionEvent
Delta="null"
EndDate="2009-09-08 18:34:51.838 PDT"
Interest="1.0"
Kind="edit"
Navigation="null"
OriginId="org.eclipse.jdt.ui.CompilationUnitEditor"
StartDate="2009-09-08 18:34:51.838 PDT"
StructureHandle="=org.eclipse.mylyn.internal.context.
ui{IContextUiHelpIds.java"
StructureKind="java"
/>
```
Software change process[1]

- **Initiation**
  - Concept Location
  - Impact Analysis

- **Prefactoring**
  - Identify a module to be modified
  - Identify a full set of modules to be modified

- **Actualization**
  - Modify source code

- **Postfactoring**

- **Conclusion**

Is our technique useful for referred context?

EMPIRICAL STUDY
Subject: Mylyn Task 3.07-3.21

Prioritized smells

VCS

Code smells

Modification-based oracle

Compare

Compare

Reference-based oracle

Interaction History

Code smells
Is our technique useful for referred modules?

Our technique can be useful to support both modified modules and referred modules.
## Top 10 results

<table>
<thead>
<tr>
<th>Rank</th>
<th>Smell Type</th>
<th>Class Name</th>
<th>#RI</th>
<th>#MI</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>God</td>
<td>TasksUiInternal</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>God</td>
<td>TasksUiPlugin</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>God</td>
<td>TaskListIndex</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>God</td>
<td>AbstractTaskEditorPage</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>God</td>
<td>TaskDataManager</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>God</td>
<td>TracRepositoryConnector</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>God</td>
<td>AttachmentUtil</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>God</td>
<td>SynchronizeTasksJob</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>Data</td>
<td>TaskData</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>God</td>
<td>BugzillaRepositoryConnector</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

#RI = Number of referring issues  
#MI = Number of modifying issues
CONCLUSION
Messages

Context-based code smells prioritization

Modified Context

Referred Context

Can support both types of context