Understanding Program Comprehension

Software Composition Seminar

Aleksandar Ilic
Luka Hamza

Mentor: Nevena Milojkovic

Autumn semester 2016/2017
Outline

1. Introduction
2. Participants
3. Interview setup
4. Static vs Dynamic analysis
5. Results
6. Conclusion
Introduction

- How do developers think?
- Experiment
- Coping with unknown code
Participants

- 12 (6+6) participants
- Academic and professional background
- Different levels of experience in Java
Interview setup

- Unknown complex project
- 30 minutes
- Undo Command bug
- Static and dynamic environment
Interview setup

Static debugging:

● No debugger

● No printing of values during runtime

Dynamic debugging:

● No constraints
Static debugging

- Different approaches in bug-localization
- Undirected flow of thoughts (e.g. undo/redo )

```java
public boolean undo() {
    if (!super.undo()) {
        return false;
    }

    FigureEnumeration fe = getAffectedFigures();
    while (fe.hasNextFigure()) {
        Figure f = fe.nextFigure();
        if (getOriginalValue(f) != null) {
            getAttribute();
        }
    }

    return true;
}
```
Dynamic debugging

- Undirected flow of thoughts - up to a point
- Polymorphic code

```java
public void execute() {
    Undoable lastUndoable = um.popUndo();
    // Execute undo
    boolean hasBeenUndone = lastUndoable.undo();
    // Add to redo stack
    if (hasBeenUndone && lastUndoable.isRedoable()) {
        um.pushRedo(lastUndoable);
    }
}
```
Sillito’s paper

- “Asking and Answering Questions during a Programming Change Task” - Jonathan Sillito [2008]
- 44 different groups of questions
- Relation with our experiment
Results

- Q8: Where does this type fit in the type hierarchy?
- Q10: Where is this field declared in the type hierarchy?
- Q12: Where is this method called or type referenced?
Polymorphism

- Challenges of polymorphism
- Problems in static environment
- Different hierarchy types
Polymorphism

- Shallow but wide hierarchy
- 24 different types of UndoActivity
- Breaking point for debugging

```java
public void execute() {
    Undoable lastUndoable = um.popUndo();
    // Execute undo
    boolean hasbeenUndone = lastUndoable.undo();
    // Add to redo stack
    if (hasbeenUndone && lastUndoable.isRedoable()) {
        um.pushRedo(lastUndoable);
    }
}```
Discussion

● Debugger is not always utilized properly
● Coping with polymorphism
● Experience in industry means a lot
● Threats to validity
Conclusion

- Developers’ bottleneck
- Importance of this study
- Future work
Summary

- Static vs dynamic
- Polymorphism
- Industry experience is important