Automating Design Flaw Correction in Object-Oriented Systems

Author:

Dipl. Ing. Iulian Dragos
iuli@lrg.cs.utt.ro
Introduction

- Software systems need to *evolve*
- Changes made on existing systems degrade their design (design decay)
  - Code becomes difficult to understand
  - New functionality is difficult to add
  - Even for small changes, a large part of the system needs to be modified

- Reengineering
  - Reverse engineering
  - Restructuring
  - Forward engineering
Restructuring

- Aims to improve the structure of a system without changing its behavior
- Makes use of:
  - Problem detection [Ciupke99, Marinescu02]
  - Code transformations [Opdyke92, Roberts99, Tokuda99]

The problem

- Large abstraction gap between design flaws and individual code transformations
- No satisfactory approach that links design flaws with ways to correct them
Goal

- Provide a mapping between design flaws and code transformations to remove them

Criteria

- Language independence
- Behavior preservation
- Automation
- Formalization
- Quality Estimation
- Extensibility
- Causality
Correction Strategies

- An exact plan for correcting a design flaw, including different alternatives and trade-offs.
- Operates on entities of a meta-model
- It contains:
  - Interface
  - Nodes
  - Branches
  - Sequences
Coral – a language for expressing Correction Strategies

```plaintext
strategy switch_statement {
  interface {
    Class C;
    List switchStatements;
    Field t;
  }

  node subclasses_or_state (8) {
    branch {
      description("This solution will create subclasses for each branch of the switch statement");
      quality {
        favors(structuredness, self-descriptiveness);
        disfavors();
      }
      replace_switch_with_subclasses(C, switchStatements, t);
    }

    branch {
      description("This will introduce the State pattern to resolve the switch statements"
        "Each switch branch will be represented by a state class.");
      quality {
        favors(structuredness, legibility, extensibility);
        disfavors(traceability);
      }
      replace_switch_with_state(C, sw, m);
    }
  }
}
```
Methodology

Quality Factor 1
Quality Factor 2
Quality Factor 3
....

Quality model

Correction Strategy

Dipl. Ing. Iulian Dragos
Implementation

- A framework for correction strategies, using Java
- Allows the easy implementation of correction strategies expressed in Coral
- Implements the methodology
- Builds upon the recoder framework
- Implemented a few correction strategies (e.g. \textit{switch statement, data class, feature envy})
Evaluation

- Correction strategies
  - Map design flaws to code transformations (*causality, extensibility*)
  - Attach quality criteria to branches (*quality estimation*)
  - Use a meta-model (*language independent*)
  - Expressed in Coral (*formalism*)

- Methodology
  - Shows how to correct design flaws (*behavior preservation, automation*)

- The framework (*automation*)
Summary of contributions

- The concept of *correction strategy*
- The Coral language for expressing correction strategies
- A methodology for correcting design flaws
- A framework for implementing correction strategies
- A list of correction strategies
Open Issues

- Modeling Quality
- Dealing with semantic issues
- Global vs. Local design improvement

Future Work

- Refine the quality model
- Develop more correction strategies
- Real world case studies
Questions, please…
Example

The “switch” problem: large conditional statements on type (or equivalent) information repeated throughout the code [Fowler99, Demeyer02]

- **Solutions:**
  - Subclassing
  - *State* pattern

```
void m() {
    switch (t) {
        case 1:
            // handle this case
            break;
        case 2:
            // handle this case
            break;
        ...
    }
}
```
The Coral Language (continued)

```java
sequence replace_switch_with_state(Class C, SwitchStatement sw, Field t) {
    preconditions {
        // check for name clashes
        System.classes.find(c.name + "State") == nil;
        foreach (branch in sw.branches) {
            System.classes.find(branch.label + "State") == nil;
            C.methods.find("changeState") == nil;
        }
    }

    Map branchClasses;
    Class stateItf = create_class(c.name + "State");

    // for each appearance of the switch in methods of class C..
    foreach (switchStm in switchStatements) {
        // add the method that contains it to the State hierarchy
        Method newMethod = create_member_function(stateItf, switchStm.owner);
        add_function_parameter(newMethod, C, "context");
    }
}
```

Dipl. Ing. Iulian Dragos
Example

private void buildStrategy() {
    subclasses_or_state = new
        DecisionNode(correctionManager, 8);
    Branch subclasses = new Branch(this);
    subclasses.setDescription("This s...");
    subclasses.addToFavors(STRUCTUREDNESS);
    subclasses.addToFavors(SELF_DESCRIP..);
    node.addBranch(subclasses);

    TransformationSequence seq = new
        ReplaceSwitchWithSubclasses(this,
            switchStatements, t);
    subclasses.addTransformation(seq);

    subclasses.setServiceConfig(sc);
    seq.setServiceConfig(sc);
}

node subclasses_or_state (8) {
    branch {
        description("This solu...");
        quality {
            favors(structuredness, self-
                descriptiveness);
            disfavors();
        }
    }

    replace_switch_with_subclasses(C,
        switchStatements, t);
}

Dipl. Ing. Iulian Dragos