

Software Refactoring

Overview

- What is refactoring?
- When to apply refactoring?
- How to apply refactoring?
- Refactoring types
- Obstacles of applying refactorings

Refactoring

- Definition
 - The process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure
- Major source: Martin Fowler et al. 1999
 - “Refactoring: Improving the Design of Existing Code”

Goals of Refactoring

- Improve software design
- Make software easier to understand
- Help to find bugs
- Help to program faster

When to Apply Refactoring?

- Design
 - Requirements get changed
 - More variations are revealed or expected
- Implementation
 - Add function
 - Need to fix bugs
 - Do code review
- As software evolves, more refactorings are applied

Implementation Refactoring

- Make code changes
- Run tests to ensure semantics is preserved
- If every test is passed, move on
- Otherwise, fix the problem or undo the change

Decide refactoring to apply

- Identify some "bad code smells"
 - Duplicated code, long method, large class, long parameter list, ...
- Match the bad code with known refactoring patterns to decide what refactoring to apply
- Note that bad code smells may appear gradually as software evolves

Refactoring Types

- Extract method/class/interface/...
- Inline method/class/interface/...
- Move field/method/...
- Pull up field/method/...
- Remove method/parameter/...
- ...

Extract Class

```
public class Customer {  
    private String name;  
    private String phoneAreaCode;  
    private String phoneNumber;  
}
```



```
public class PhoneInfo {  
    private String areaCode;  
    private String number;  
}  
  
public class Customer {  
    private String name;  
    private PhoneInfo phone;  
}
```

- Too much phone info detail as part of the Customer class violates high cohesion principle
- Split the class into two to keep phone info in a separate class

Extract Interface

```
public class Customer {  
    private String name;  
    private String getName() {  
        return name;  
    }  
    public void setName(String string) {  
        name = string;  
    }  
    public String toXML() {  
        return "<Customer><Name>" + name + "</Name></Customer>";  
    }  
}
```

Extract Interface (cont.)

- Motivation
 - Some clients only use a subset of a class' responsibilities
 - E.g., toXML()
 - There are several classes providing certain functions in common
 - E.g., Customer, Employee, Product
- Solution
 - Declare the subset of operations in a separate interface

```
public interface SerializableToXML{  
    public abstract String toXML();  
}
```

```
public class Customer implements SerializableToXML {  
    private String name;  
    private String getName() {  
        return name;  
    }  
    public void setName(String string) {  
        name = string;  
    }  
    public toXML() {  
        return "<Customer><Name>" + name +  
            "</Name></Customer>";  
    }  
}
```

- **Benefits**

- Information hiding: only expose the relevant operations to specific clients

Obstacles of Refactoring

- Problem: Databases
- Reason
 - Business applications are tightly coupled to database schema
 - With business logic change, you may also need to change database schema and migrate data
- Advice
 - Put a separate layer between object model and database model

Obstacle 2: Changing Interfaces

- Problem: Some refactoring, such as “**Rename Method**”, modify interfaces
- Reasons: There is a problem if the interface is used by code that you cannot find and change
- Advice
 - Don't publish interfaces prematurely
 - Maintain both the old and new interfaces for a while

Obstacle 3: Design Changes That Are Difficult to Refactor

- Problem: some design decisions are so central that you cannot count on refactoring to change your mind later
- Advice:
 - Think about design alternatives and assess difficulty of refactoring from one to another
 - Go with the simplest one if refactoring is not difficult; otherwise, put more effort into design