Making an evolvable software: *Refactoring*

Dr. Ah-Rim Han
Today’s Topic

- Software Engineering
- Refactoring
  - Why Need Refactoring?
  - What is Refactoring?
  - Refactoring Process
  - Bad Smells: Software Design Problems
  - Refactoring Types
  - Refactoring Assessment: Maintainability
    - Coupling and Cohesion metrics
  - Research Trends
- Doing Ph.D.
Software Engineering?
IEEE’s Standard 610.12-1990: Glossary of Software Engineering Terminology

- **Software engineering is defined as the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software**
Definition

- David Lorge Parnas
  - *Software engineering is defined as the multi-person construction of multi-version software*
In My Point of View...

- Aims at providing the automated tools, techniques, processes to assist developers, managers, and stakeholders for systematic software development
도요타 프리우스 190만대 리콜 '소프트웨어 결함' tbs 교통방송 | 2014.02.12 (수) 오전 4:47
도요타는 지난 2012년에도 740만대가 넘는 차량을 리콜했으며, 출시 2월에도 소프트웨어 결함으로 190만대를 리콜한 바 있다. 전문가들은 이번 리콜로 도요타가 5억 8000만달러, 우리나라로 약...}

도요타 277개 차종 676만대 리콜...사상 최대 규모 SBS CNBC | 2014.04.10 (목) 오전 8:25
또한 도요타는 지난 2월에도 하이브리드 승용차 프리우스 190만대를 소프트웨어(SW) 결함 문제로 리콜하기로 했다. 도요타는 이로써 지난 2월 하이브리드 승용차 프리우스 190만대를 소프트웨어 결함 문제로...}

도요타 639만대 리콜(회수 무상수리)… 세계 1 역사상 초대로 큰 규모 조선일보 | 2014.04.09 (수) 오후 4:39
도요타는 2012년에도 743만대를 리콜하였고, 출시 2월에도 하이브리드 승용차 프리우스 190만대를 소프트웨어(SW) 결함으로 리콜한 바 있다. 포르투는 업계 사정이 포르투나 소프트웨어 결함으로...
Refactoring
Why Need Refactoring?

Software changes

- Introducing new functionalities
- Correcting bugs
- Adapting new environments
  - New OS, new hardware
- Providing better qualities
  - Better performance, better reliability, …

→ Changes often take place without consideration of the design rationale due to time constraints

→ Therefore, the design quality of the software may degrade overtime
Why Need Refactoring?

Cost of change curve

Cost of Change

Requirements  Analysis & Design  Coding  Testing  Production

http://www.agilemodeling.com/essays/costOfChange.htm

© All copyrights are reserved by Ah-Rim Han, Ph.D. at Korea University (arhan@korea.ac.kr)
Why Need Refactoring?

- Refactoring improves the design of software
- Refactoring makes software easier to understand
- Refactoring helps you program faster

- These help to fix bugs and accommodating changes in a easier and faster way, which improves maintainability of the software

- At the end, this reduces maintenance costs
What is Refactoring?

- **What is refactoring?**
  - **Refactoring (noun):** a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior.
  - **Refactor (verb):** to restructure software by applying a series of refactorings without changing its observable behavior.

Refactoring Process

- Identify places where the software should be refactored
- Determine which refactoring(s) should be applied
- Guarantee that the applied refactoring preserves behavior
- Apply the refactoring
- Assess the effect of the refactoring on quality (e.g., maintainability, testability, understandability)
- Maintain the consistency between the refactored program code and other software artifacts

Refactoring Process

- Source code
- Identification
- Candidate Refactorings
  - Accessment
  - Selection
- Selected Refactorings

| Fit. Func. 1 | 7 | 8 | 6 | 3 | 10 | 8 |
| Fit. Func. 2 | 8 | 7 | 4 | 5 | 10 | 9 |
Bad Smells

- Divergent Change
  - When one class is commonly changed in different ways for different reasons
  - Solution: Extract Class, Move Method

© All copyrights are reserved by Ah-Rim Han, Ph.D. at Korea University (arhan@korea.ac.kr)
Bad Smells

❖ Shotgun Surgery
  - Is similar to divergent change but the opposite
    - Divergent change is one class that suffers many kinds of changes, and shotgun survey is one change that alters many classes
  - You have to make a lot of little changes to a lot of different classes
  - Solution: Move Method, Move Field, Inline Class

❖ Feature Envy
  - A method that seems more interested in a class other than the one it actually is in
  - Solution: Move Method
Refactoring Types

✿ Moving Features Between Objects
  • Extract Class
  • Hide Delegate
  • Inline Class
  • Introduce Foreign Method
  • Introduce Local Extension
  • Move Field
  • Move Method
  • Remove Middle Man

✿ Composing Methods
  • Extract Method
  • Inline Method
  • Inline Temp
  • Introduce Explaining Variable
  • Remove Assignments to Parameters
  • Replace Method with Method Object
  • Replace Temp with Query
  • Split Temporary Variable
  • Substitute Algorithm
Refactoring Types

Organizing Data

- Change Bidirectional Association to Unidirectional
- Change Reference to Value
- Change Unidirectional Association to Bidirectional
- Change Value to Reference
- Duplicate Observed Data
- Encapsulate Collection
- Encapsulate Field
- Replace Array with Object

- Replace Data Value with Object
- Replace Magic Number with Symbolic Constant
- Replace Record with Data Class
- Replace Subclass with Fields
- Replace Type Code with Class
- Replace Type Code with State/Strategy
- Replace Type Code with Subclasses
- Self Encapsulate Field
Refactoring Types

❖ Simplifying Conditional Expressions

- Consolidate Conditional Expression
- Consolidate Duplicate Conditional Fragments
- Decompose Conditional
- Introduce Assertion
- Introduce Null Object
- Remove Control Flag
- Replace Conditional with Polymorphism
- Replace Nested Conditional with Guard Clauses

❖ Dealing with Generalization

- Collapse Hierarchy
- Extract Interface
- Extract Subclass
- Extract Superclass
- Form Template Method
- Pull Up Constructor Body
- Pull Up Field
- Pull Up Method
- Push Down Field
- Push Down Method
- Replace Delegation with Inheritance
- Replace Inheritance with Delegation
Refactoring Types

Making Method Calls Simpler
- Add Parameter
- Encapsulate Downcast
- Hide Method
- Introduce Parameter Object
- Parameterize Method
- Preserve Whole Object
- Remove Parameter
- Remove Setting Method
- Rename Method
- Replace Constructor with Factory Method
- Replace Error Code with Exception
- Replace Exception with Test
- Replace Parameter with Explicit Methods
- Replace Parameter with Method
- Separate Query from Modifier

Big Refactorings
- Convert Procedural Design to Objects
- Extract Hierarchy
- Separate Domain from Presentation
- Tease Apart Inheritance
- The Nature of the Game
Refactoring Types

Extract Class

- You have one class doing work that should be done by two
- Create a new class and move the relevant fields and methods from the old class into the new class

```
Person
  name
  officeAreaCode
  officeNumber
  getTelephoneNumber

Person
  name
  getTelephoneNumber

Telephone Number
  areaCode
  number
  getTelephoneNumber
```

© All copyrights are reserved by Ah-Rim Han, Ph.D. at Korea University (arhan@korea.ac.kr)
Refactoring Types

- Inline Class
  - A class isn't doing very much.
  - → Move all its features into another class and delete it.
Refactoring Types

Pull Up Method

- You have methods with identical results on subclasses.
- Move them to the superclass.
Refactoring Types

- **Form Template Method**
  - You have two methods in subclasses that perform similar steps in the same order, yet the steps are different.
  - Get the steps into methods with the same signature, so that the original methods become the same. Then you can pull them up.
Maintainability

- Definition (from ISO 9126)
  - The capability of the software product to be modified.
  - Modifications may include corrections, improvements or adaptation of the software to changes in environment, and in requirements and functional specification

- Maintainability is influenced by a lot of sub-qualities
Coupling

- a measure of how strongly dependent one software unit is on other software units
  - unit = class, package, module, method, application, etc.
Cohesion

- a measure of how strongly related and focused the responsibilities and provided behavior of a software unit are

→ Good design = high maintainability, low coupling and high cohesion
Cohesion Metrics

- **Method Similarity Cohesion (MSC) [0]**
  \[
  MSC(C) = \frac{2}{n(n-1)} \sum_{i=1}^{2} \frac{IV_c}{IV_t} i,
  \]
  where class C has n methods, and for a pair of methods, \(IV_c\) and \(IV_t\) stand for the common (i.e., intersect set) and total instance (i.e., union set) variables used by the pair of methods repeatedly. Since there are \(\frac{n(n-1)}{2}\) distinct combinations of pairs of methods in a class, \(i\) ranges from 1 (i.e., first pair) to \(\frac{n(n-1)}{2}\) (i.e., last pair), and \(\frac{IV_c}{IV_t}\) indicates the similarity of the pair of methods, respectively.

- **Lack of Cohesion in Methods (LCOM) [1]**
- **Cohesion Among Methods in Class (CAMC) [2]**
Coupling Metrics

- Message Passing Coupling (MPC) [3]
  - Counts static method calls for all invoked methods in the import direction

- Request For a Class (RFC) [1]
  - Counts static method calls for distinct methods in the import direction

- Coupling Between Objects (CBO) [1]
  - Counts static method calls for distinct methods in both directions.

- Coupling Factor (CF) [4]
  - The coarse-grained metrics / measured based on the number of coupled classes, not on the methods.
Research Trends

Inheritance restructuring

Refactoring

Approach for supporting refactoring activities

Search-based refactoring

Refactoring opportunity identification

- Focus on implementation issues (e.g., maximizing sharing and minimizing duplication at method or expression level)

- Start to focus on improving software design quality; therefore, consider higher levels such as methods and classes

- Provide methods such as
  - Design flaw detection (or bad smell detection)
  - Evaluation of refactoring effect on design quality
  - Program behavior preservation, etc.

- Want to automate the full refactoring process (without human intervention) by treating OO design as an optimization problem

- Provide the method for automated identification (i.e., suggestion) of specific refactoring opportunities to resolve specific design problems or to improve specific design quality (which does not depend on random choice)
Doing Ph.D.
Life in the laboratory

- Attend and have a seminar
- Discuss research topics in the SIG (Study In Group)
- Attend conferences
  - International Conference on Software Engineering (2006)
  - Korea Computer Congress (KCC)
  - Korean Conference on Software Engineering
  - Asia Pacific Conference on Software Engineering (2013)

Berri Sohem (COCOMO)
Lionel Briand
Gregg Rothermel
Richard Taylor

© All copyrights are reserved by Ah-Rim Han, Ph.D. at Korea University (arhan@korea.ac.kr)
How Hard... But, It’s Worth!!!

How Grad School Is Just Like Kindergarten

All Day Napping Is Acceptable
There Is Constant Adult Supervision

You Get Cookies For Lunch
Most Common Activity: Cutting And Pasting

There Are No Grades (You Just Have To Play Well With Others)
Crying For Your Mommy Is Normal

© All copyrights are reserved by Ah-Rim Han, Ph.D. at Korea University (arhan@korea.ac.kr)
Qualification to be Succeed in Ph.D

- Fundamental basis
  - Algorithm, data base, automata, system programming, compiler, graphics, artificial intelligence, network, operating systems, computer architecture, stochastic, …
- English skills (presentation, discussion, writing a paper, …)
- Communication skills
- Self-motivated
References

- Lecture notes from Dr. Miryung Kim
  - [http://users.ece.utexas.edu/~miryung/teaching/EE461L-Fall2013/main.html](http://users.ece.utexas.edu/~miryung/teaching/EE461L-Fall2013/main.html)

- Lecture notes
  - [http://kurser.lobner.dk/dSoftArk/Slides/w44-45/4_3_maintainability.pdf](http://kurser.lobner.dk/dSoftArk/Slides/w44-45/4_3_maintainability.pdf)

- Refactoring materials:
  - [http://sourcemaking.com/refactoring](http://sourcemaking.com/refactoring)

  - [http://martinfowler.com/refactoring/](http://martinfowler.com/refactoring/)


