REFACTORING

CSH3E3 #6

Tim Dosen KK SIDE

Sub Bahasan

- Pengertian Refactoring
- Levels of Software Changes
- Why & When Refactoring
- Catalog & Category
- Big Refactoring
- Tools, Challenges & Limitations
- Tugas Eksplorasi

Sub Bahasan 1
Pengertian Refactoring

Mathematics: Factor

- fac·tor
  - One of two or more quantities that divides a given quantity without a remainder, e.g., 2 and 3 are factors of 6; a and b are factors of ab
- fac·tor·ing
  - To determine or indicate explicitly the factors of

SE: Factoring

- fac·tor
  - The individual items that combined together form a complete software system:
    - identifiers
    - contents of function
    - contents of classes and place in inheritance hierarchy
- fac·tor·ing
  - Determining the items, at design time, that make up a software system

Refactoring

- *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* [Fowler'99]
- *A program restructuring operation to support the design, evolution, and reuse of object oriented frameworks that preserve the behavioural aspects of the program* [Opdyke'92]

Specifics

- Source to source transformation
- Remain inside the same language, e.g., C++ to C++
- Does not change the programs behavior
- Originally designed for object-oriented languages, but can also be applied to non-object oriented language features, i.e., functions

Sub Bahasan 2
Levels of Software Changes

Levels of Software Changes

- **Low Level**
  - Change lines of code
  - e.g., Changes in (a least) two classes
- **Intermediate Level**
  - Change design (factoring)
  - e.g., Move a member function
- **High Level**
  - Features to be added to a system
  - e.g., New feature

Relationship to Design

- Not the same as “cleaning up code”
  - May cause changes to behavioral aspects
  - Changes often made in a small context or to entire program
- Key element of entire process in agile methodologies
- Views design as an evolving process
- Strong testing support to preserve behavioral aspects

Quick Examples

- Introduce Explaining Variable
- Rename Method
- Move Method
- Pullup Method
- Change Value to Reference
- Remove Parameter
- Extract Hierarchy

- LIHAT : **www.refactoring.com**

Sub Bahasan 3
Why & When Refactoring

Code "smells"

- Duplicated Code
- Long Method
- Large Class
- Long Parameter List
- Divergent Change
- Shotgun Surgery
  - (change one place $\rightarrow$ must change others)
- Feature Envy
- Data Clumps
- Primitive Obsession
- Switch Statements
- Parallel Inheritance Hierarchies
- Lazy Class
- Speculative Generality
- Temporary Field
- Message Chains
- Middle Man
- Inappropriate Intimacy
- Alternative Classes with Different Interfaces
- Incomplete Library Class
- Data Class
- Refused Bequest
  - (subclass doesn't use inherited members much)
- Comments

Why: Design Preservation

- Code changes often lead to a loss of the original design
- Loss of design is cumulative:
  - Difficulties in design comprehension ->
    Difficulties in preserving design ->
    More rapid decay of design
- Refactoring improves the design of existing code

Why: Comprehension

- Developers are most concerned with getting the program to work, not about future developers
- Refactoring makes existing code more readable
- Increases comprehension of existing code, leading higher levels of code comprehension
- Often applied in stages

Why: Debugging

- Greater program comprehension leads to easier debugging
- Increased readability leads to the discovery of possible errors
- Understanding gained during debugging can be put back into the code

Why: Faster Programming

- Counterintuitive argument made by Fowler
- Good design is essential for rapid development
- Poor design allows for quick progress, but soon slows the process down
  - Spend time debugging
  - Changes take longer as you understand the system and find duplicate code

When?

- Adding Functionality
  - Comprehension of existing program
  - Preparation for addition
- Debugging
  - Comprehension of existing program
- Code Review
  - Preparation for suggestions to other programmers
  - Stimulates other ideas

Sub Bahasan 4
Catalog & Category

Catalog

- Collected by Martin Fowler
- Refactoring entry composed of:
  - Name
  - Summary
  - Motivation
  - Mechanics
  - Examples
- Based on Java

Categories

- Composing Methods
  - Creating methods out of inlined code
- Moving Features Between Objects
  - Changing of decisions regarding where to put reponsibilities
- Organizing Data
  - Make working with data easier

Categories II

- Simplifying Conditional Expressions
- Making Method Calls Simpler
  - Creating more straightforward interfaces
- Dealing with Generalization
  - Moving methods around hierarchies
- Big Refactorings
  - Refactoring for larger purposes

Composing Methods

- Extract Method
- Inline Method
- Inline Temp
- Replace Temp with Query
- Introduce Explaining Variables
- Split Temporary Variable
- Remove Assignments to Parameters
- Replace Method with Method Object
- Substitute Algorithm

Extract Method

- Create a new method, and name it after the intention of the method (name it by what it does, not by how it does it)
- Copy the extracted code from the source method into the new target method
- Scan the extracted code for references to any variables that are local in scope to the source method

Extract Method II

- See whether any temporary variables are used only within this extracted code. If so, declare them in the target method as temporary variables
- Look to see whether any local-scope variable are modified by the existing code (See Split Temporary Variable and Replace Temp with Query)
- Pass into the target method as parameters local scope variables that are read from the extracted code

Extract Method III

- Compile when you have dealt with all the locally-scoped variables
- Replace the extracted code in the source method with a call to the target method
- Compile and test

Extract Method

```
String name;

void printOwing(double amount) {
        printBanner();

        // print details
        std::cout << “name” << _name << std::endl;
        std::cout << “amount” << _amount << std::endl;
}
/**********************************************************/
void printOwing(double amount) {
        printBanner();
        printDetails(amount);
}

void printDetails(double amount) {
        std::cout << “name” << _name << std::endl;
        std::cout << “amount” << _amount << std::endl;
}
```

Inline Method

```
int getRating() {
        return moreThanFiveLateDeliveries() ? 2 : 1;
}

bool moreThanFiveLateDeliveries() {
        return _numberOfLateDeliveries > 5;
}
```

```
int getRating() {
        return (_numberOfLateDeliveries > 5) ? 2 : 1;
}
```

Inline Temp

```
double basePrice = anOrder.basePrice();
return basePrice > 1000;
```

```
return anOrder.basePrice() > 1000;
```

Remove Assignments to Parameters

```
int discount (int inputVal, int quantity, int yearToDate) {
    if (inputVal > 50) inputVal -= 2;
    ...

int discount (int inputVal, int quantity, int yearToDate) {
    int result = inputVal;
    if (inputVal > 50) result -= 2;
   ...
```

c

Moving Object Features

- Move Method
- Move Field
- Extract Class
- Inline Class
- Hide Delegate
- Remove Middle Man
- Introduce Foreign Method
- Introduce Local Extension

Organizing Data

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

Organizing Data II

- Replace Magic Number with Symbolic Constant
- Encapsulate Field
- Encapsulate Collection
- Replace Record with Data Class
- Replace Type Code with Class
- Replace Type Code with Subclasses
- Replace Type Code with State/Strategy
- Replace Subclass with Fields

Simplifying Conditional Expr.

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

Example

- Decompose Conditional

```
if (date.before (SUMMER_START) || date.after(SUMMER_END))
   charge = quantity * _winterRate + _winterServiceCharge;
else
   charge = quantity * _summerRate;
```

```
if (!isSummer(date))
   charge = winterCharge(quantity);
else
   charge = summerCharge(quantity);
```

Simplifying Method Calls

- Rename Method
- Add Parameter
- Remove Parameter
- Seperate Query from Modifier
- Parameterize Method
- Replace Parameter with Explicit Methods
- Preserve Whole Object
- Replace Parameter with Method

Simplying Method Calls II

- Introduce Parameter Object
- Remove Setting Method
- Hide Method
- Replace Constructor with Factory Method
- Encapsulate Downcast
- Replace Error Code with Exception
- Replace Exception with Test

Dealing with Generalization

- Pull Up Field
- Pull Up Method
- Pull Up Constructor Body
- Push Down Method
- Push Down Field
- Extract Subclass
- Extract Superclass
- Extract Interface

Dealing with Generalization II

- Collapse Hierarchy
- Form Template Method
- Replace Inheritance with Delegation
- Replace Delegation with Inheritance

Sub Bahasan 5
Big Refactoring

Big Refactorings

- Tease Apart Inheritance
  - Split an inheritance hierarchy that is doing two jobs at once
- Convert Procedural Design to Objects
- Separate Domain from Presentation
  - GUI classes that contain domain logic
- Extract Hierarchy
  - Create a hierarchy of classes from a single class where the single class contains many conditional statements

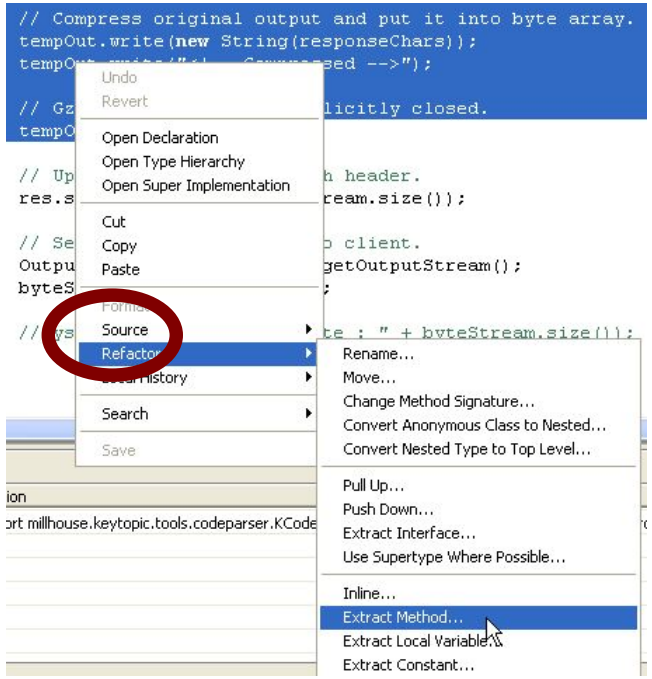
Convert Procedural Design to OO

- Take each record type and turn it into a dumb data object with accessors
- Take all procedural code and put it into a single class
- Take each long method and apply *Extract Method* and the related factorings to break it down. As you break down the procedures use Move Method to move each one to the appropriate dumb data class
- Continue until all behavior is removed from the original class

Sub Bahasan 6
Tools, Challenges & Limitations

IDE support for refactoring

- Eclipse and Visual Studio support:
  - variable / method / class renaming
  - method or constant extraction
  - extraction of redundant code snippets
  - method signature change
  - extraction of an interface from a type
  - method inlining
  - providing warnings about method invocations with inconsistent parameters
  - help with self-documenting code through auto-completion



Other Tools

- Smalltalk Refactoring Browser
  - Development environment written in Smalltalk
  - Allows for Smalltalk source code to transform Smalltalk source code
  - Comments as a first-class part of the language
- XRefactory
  - Allows standard refactorings for C++

Challenges

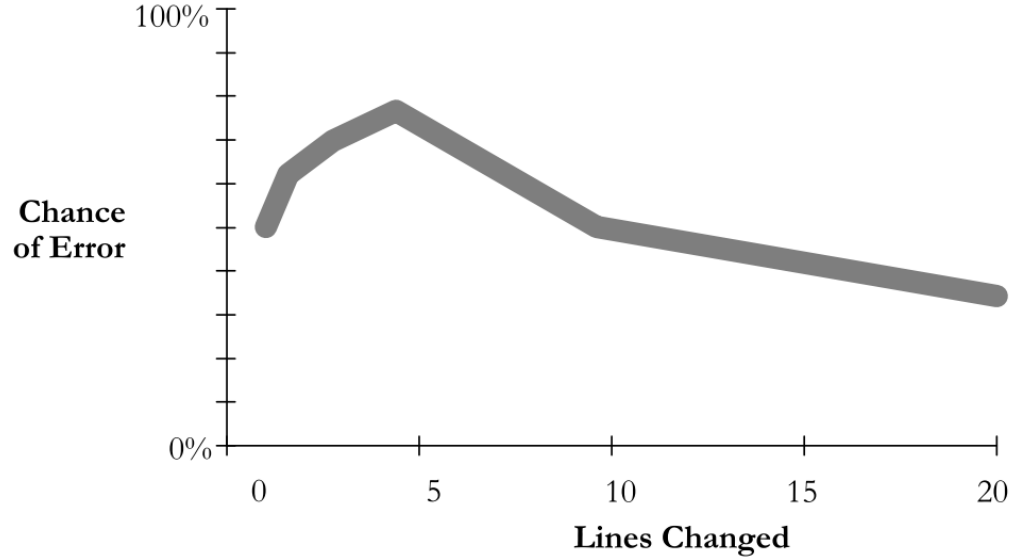
- Preservation of documentary structure (comments, white space etc.)
- Processed code (C, C++, etc.)
- Integration with test suite
- Discovery of possible refactorings
- Creation of task-specific refactorings

Limitations

- Tentative list due to lack of experience
- Database
  - Database schema must be isolated, or schema evolution must be allowed
- Changing Published Interfaces
  - Interfaces where you do not control all of the source code that uses the interface
  - Must support both old and new interfaces
  - Don't publish interfaces unless you have to

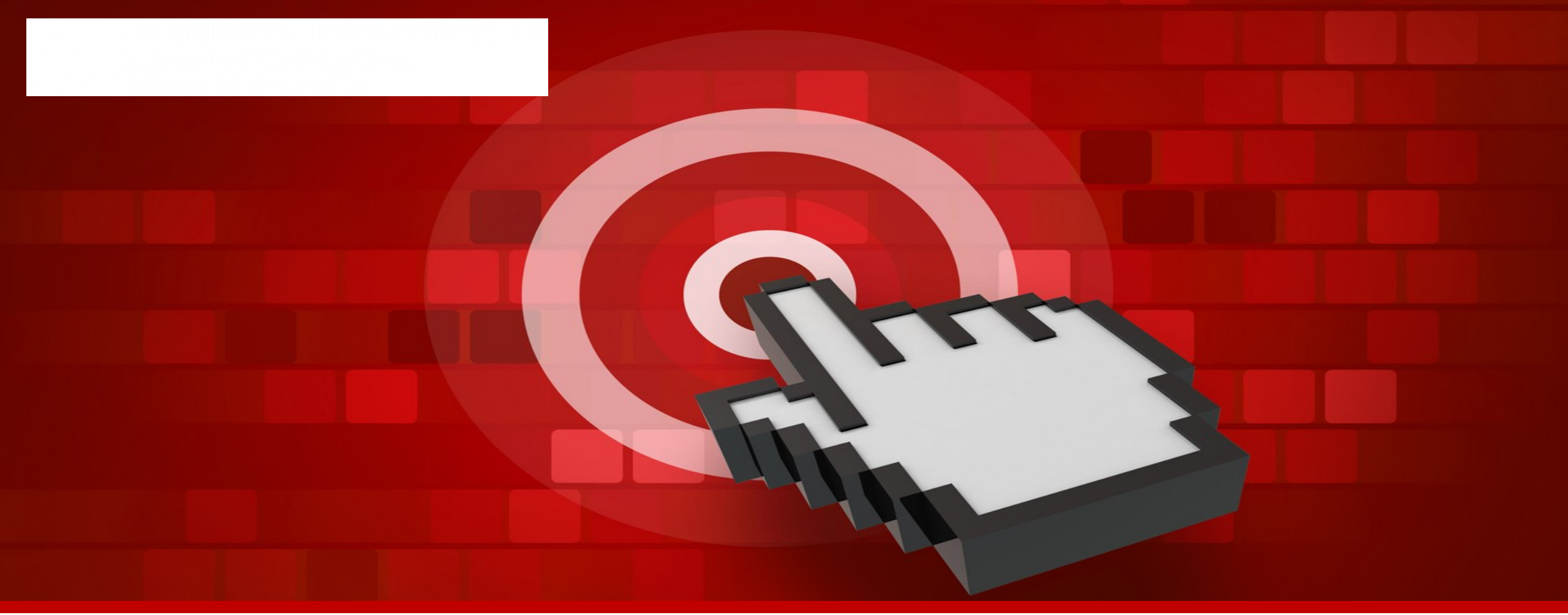
Dangers of refactoring

- code that used to be ...
  - well commented, now (maybe) isn't
  - fully tested, now (maybe) isn't
  - fully code reviewed, now (maybe) isn't
- easy to insert a bug into previously working code (regression!)
  - a small initial change can have a large chance of error



Tugas Eksplorasi

- **Buat Code OOP untuk satu form (selain login) yang melakukan berikut ini : Validate all data**
  - Existence checks
  - Data-type checks
  - Domain checks
  - Combination checks
  - Self-checking digits
  - Format checks
- Periksa ulang code OOP Anda, apakah sudah paling efisien
- Submit sebelum kuliah minggu depan



THANK YOU