Security Concern Refactoring: Increasing and Assessing the Security Level of Existing Software

Katsuhisa Maruyama
Department of Computer Science
Ritsumeikan University
maru@cs.ritsumei.ac.jp
Background

Software security is ever-increasingly becoming a serious issue since software comes to play an essential role in the real world and our lives much depend on software (Viega and McGraw, 2001)

Nevertheless,

- There are an enormous number of vulnerable software programs with unnecessary privileges, known flaws, or weak access control settings on resources

- Security characteristics of well-designed software programs will deteriorate due to recurring modifications

- Security vulnerabilities are easily inserted while on software modification
Approaches to Software Security

To remove risks posed by attacks which try to exploit security vulnerabilities:

• Secure software will be created from scratch
  ✓ Hard to in advance design and implement secure software

• Existing software will be directly changed into secure one
  by partially modifying its design or code
  ✓ Not perfectly remove security vulnerability from a large number of running programs in the world
  ✓ Can decreases the likelihood of threats

• Design or code of existing software will be improved without assessing its security level
  ✓ Not eliminate all kinds of interfusion of vulnerabilities resulting from improvement
  ✓ Can know of decreasing of security level while improving
Security Concern Refactoring

Most modification of software affects (increases or decreases) its security characteristics.

Refactoring is a special pattern of software modification, which changes the internal structure of existing code without changing its external behavior by applying a series of behavior-preservation transformations.

- **Secure refactoring** which increases the security level of existing code by its transformations.
- **Security-aware refactoring** which assess the effects of the security level of existing code in its transformations.

Remove and/or reveal inadvertent security flaws in programs that could be accidentally or intentionally exploited to damage assets.
Refactoring Examples

**Move Filed**
Moves a field in the class on which it is defined to another class that frequently uses the field

**Encapsulate Field**
Makes a public field private and provides its accessors

**Pull Up Field**
Moves the same field of sibling classes to their common superclass
Secure Refactoring

A change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior (Fowler, 1999)

A change made to the internal structure of software to make it more secure without changing its observable behavior

Directly removes vulnerable fragments from existing code

E.g., excessive disclosure of sensitive data
needless flexibility of code

insecure code

behavior-preservation transformations

more secure code

understandability(?)
modifiability(?)

security(+)

not traditional refactoring

meeting at FAU, December 14, 2007
Security-Aware Refactoring

A change made to the internal structure of software to make it easier to understand and cheaper to modify without both changing its observable behavior and missing unexpected changes of its security level.

E.g., change of accessibility setting of sensitive data

Reveal the impact of the change on the security level of existing code

traditional refactoring

more maintainable code

understandability(+) modifiability(+)

security(+,-, 0)

deteriorate code

behavior-preservation transformations

security?
Definitions (in this work)

Security = Protection of assets (e.g., sensitive information)

Classification of protection measures:
• Prevention
  ✓ prevents assets from being damaged
• Detection
• Reaction

Deal with two of several security aspects:
• Confidentiality
  ✓ Prevention of unauthorized disclosure of sensitive data
• Integrity
  ✓ Prevention of unauthorized modification of sensitive data

The security level of software is measured based on the potential existence of vulnerable code (inadvertent code flaws) which might violate the confidentiality and integrity.
Java Security

- A **Sandbox model** which is responsible for protecting resources according to user-defined security policy
  - However, it cannot block violation by inadvertent code
- **Four possible access levels** specified by access modifiers
  - **private**: accessible from only inside the class defining the construct
  - **(default)**: accessible from any class in the same package as the class defining the construct
  - **protected**: accessible from the class defining the construct, classes within the same package as the defining class, or subclasses of the defining class
  - **public**: accessible from any class
- **The final modifier** so as that any program code cannot accidentally change the state of an element
  - A final variable can be set only once
  - A final field can be set only once by a constructor
  - A final method cannot be neither overridden nor hidden
  - No subclass can be derived from a final class
Secure Refactoring
Secure Refactoring Transformations

(1) **Introduce Immutable Field**
   - Protects internal data stored in a field from modification
   - The field must be declared final and its data is assigned only once in a constructor of the class defining the field

(2) **Replace Reference with Copy**
   - Protects internal data stored in an immutable but mutable-typed field from modification
   - Passes a copy of the original data instead of a reference to an instance storing the data

(3) **Prohibit Overriding**
   - Removes excessive use of polymorphic calls by subclassing
   - The method or class must be declared final so that it is not intended to be overridden or inherited

(4) **Clear Sensitive Value Explicitly**
   - Prevents an adversary from stealing internal data stored in the memory
   - The original data must be replaced with dummy as early as possible if the data will not be used in the future
Motivating Code: Prohibit Overriding (1/3)

// before refactoring (insecure)

class Member {
    public String getInfo(Password pw) {
        if (pw.check()) {
            return "Secret Info";
        } else {
            return "No Info";
        }
    }
}

class Password {
    private final String secretPasswd = "SECRET";
    private final String passwd;
    public Password(String pw) {
        passwd = pw;
    }
    public boolean check() {
        return passwd.equals(secretPasswd);
    }
}

// attacking code

class FakePassword extends Password {
    ...
    public boolean check() {
        return true;
    }
}

Member member;
Password fake = new FakePassword("X");
System.out.println(member.getInfo(fake));
Security Smell: Prohibit Overriding (2/3)

In a target class $C$, there is a call to a method $m$ of a class $Ci$ for an instance $i$ which is given by external (possibly malicious) code, and any class can be derived from $Ci$.

Situation in which the refactoring should be applied
Concise Procedure: Prohibit Overriding (3/3)

1. Find all subclasses of $Ci$ and check to see if respective methods of the subclasses override $m$.
   ⇒ If overriding methods exist, consider applying the “Replace Inheritance with Delegation (352)” to replace them with delegation methods. If some of them remain, cancel this refactoring.

2. Add the keyword `final` to $m$ in its declaration. If there is no subclass of $Ci$, make $Ci$ final.

```java
// after refactoring (more secure)
final class Password {
    private final String secretPasswd = "SECRET";
    private final String passwd;
    public Password(String pw) { ... }
    public final boolean check() {
        return passwd.equals(secretPasswd);
    }
}
```

Class $Ci$, Method $m$
Security Smell: Clear Sensitive Value (1/4)

Security smell:
In a target class $C$, there is a method $m$ which keeps the sensitive value of a local variable $v$ until the execution of $m$ is terminated.

Code before refactoring

```java
// before refactoring (insecure)
class Member {
    public void printMessage(int score) {
        if (score < 60) {
            print(score);
            printMessage();
        }
    }
}
```

A stealer might be able to observe the value of $score$ through memory.

The value of $score$ will not be used in the future but still remains in the memory.

Local variable (parameter) $v$

Target class $C$
Concise Procedure: Clear Sensitive Value (2/4)

1. Check to see if the value of $v$ is stored in an immutable instance.
   ⇒ If an immutable instance is used, replace it with the use of a mutable instance.
2. Find final-use statements for $v$ by using control flow and data flow analysis.
3. Insert new statements that update each value of $v$ with the dummy value.

$sv$: a statement either writing or reading the value of $v$
$bv$: a conditional statement a branch of which includes $sv$ ($bv$ dangles $sv$)
$P(nv)$: a reachable path from a node which is one of nodes executed immediately after $nv$ ($= sv$ or $bv$) on control flow graph (CFG)
**Final-Use Nodes: Clear Sensitive Value (3/4)**

A final-use node is defined as a final-use node for \( v \) if there is at least one \( P(nv) \) which does not include any nodes which correspond to statements using (writing or reading) the value of \( v \).

**Roughly,** all reachable nodes from \( nv \) do not use \( v \).

**Control flow graph (CFG)**
Final-Use Nodes: Clear Sensitive Value (4/4)

// before refactoring (insecure)
class Member {
    public void printMessage(int score) {
        if (score < 60) {
            print(score);
            printMessage();
        }
    }
}

// after refactoring (more secure)
class Member {
    public void printMessage(int score) {
        if (score < 60) {
            print(score);
            score = -1;
            printMessage();
        } else {
            score = -1;
        }
    }
}
Secure-Aware Refactoring
Criteria for Security-Aware Refactoring

1. Accessibility settings specified in code
   - Lattice of access settings for classes, methods, and fields

2. Location where internal data is stored or observed
   - Lattice of residence for fields, local variables, parameters, and return values

3. Possibility of Subclassing
   - Can we create subclasses or overridden methods?

```java
class C {
    private int f;
    public int m(int p) {
        int v;
        return v;
    }
}
class B extends C {
}
```
Criterion of Access Level

Class diagram

High

(*, private)  { X }
↑
(*, default)  { X, A, B }
↑
(default, protected)  { X, A, B, (D) }
(through I1)
↑
(default, public)  { X, A, B, (D) }
(through I1/I2)
↑
(public, protected)  { X, A, B, C, (D) }
(through I1)
↑
(public, public)  { X, A, B, C, D }

Low

X: target class
A, B, C, D: related class
I1, I2: interface

Settings for a class
Settings for a field/method
Criterion of Locality

Local variable exists only inside the block

Parameter can be used in the called method and its overriding methods

Return value

Field

Return value of static method call

Static field

Need to prepare suitable parameters to make method call a success

→ return value > field

Static members can be accessible by using class and member names

→ instance member > static one
Preliminary Study

- Used Fowler’s catalog (Fowler, 1999)
- Examined transformations of all the 72 refactorings
- Operations of the transformations were classified into 7 primitive modifications:
  Create/Delete/Move/Extract/Inline/Encapsulate/Attribute change
Findings

Readable level indicates if external code can directly or indirectly internal data (Confidentiality).

Writable level indicates if external code can directly or indirectly modify Internal data (Integrity).

Table 1. The effects on the security level by the application of refactorings

<table>
<thead>
<tr>
<th>Refactorings</th>
<th>Readable level</th>
<th>Writable level</th>
<th>Locality</th>
<th>Subclassing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extract Method (110)</td>
<td>( m_{\text{move}} )</td>
<td>( m_{\text{move}} )</td>
<td>( p_{\text{extract}}, r_{\text{extract}} )</td>
<td>( p_{\text{delete}}, r_{\text{delete}} )</td>
</tr>
<tr>
<td>Inline Method (117)</td>
<td>( c_{\text{move}} )</td>
<td>( c_{\text{move}} )</td>
<td>( f_{\text{extract}} )</td>
<td>( f_{\text{extract}} )</td>
</tr>
<tr>
<td>Replace Temp with Query (120)</td>
<td>( c_{\text{move}} )</td>
<td>( c_{\text{move}} )</td>
<td>( f_{\text{extract}} )</td>
<td>( f_{\text{extract}} )</td>
</tr>
<tr>
<td>Replace Method with Method Object (135)</td>
<td>( c_{\text{move}} )</td>
<td>( c_{\text{move}} )</td>
<td>( f_{\text{extract}} )</td>
<td>( f_{\text{extract}} )</td>
</tr>
<tr>
<td>Move Method (142)</td>
<td>( m_{\text{move}} )</td>
<td>( m_{\text{move}} )</td>
<td>( p_{\text{extract}}, r_{\text{extract}} )</td>
<td>( p_{\text{delete}}, r_{\text{delete}} )</td>
</tr>
<tr>
<td>Move Field (146)</td>
<td>( c_{\text{move}} )</td>
<td>( c_{\text{move}} )</td>
<td>( f_{\text{extract}} )</td>
<td>( f_{\text{extract}} )</td>
</tr>
<tr>
<td>Extract Class (149)</td>
<td>( c_{\text{move}} )</td>
<td>( c_{\text{move}} )</td>
<td>( f_{\text{extract}} )</td>
<td>( f_{\text{extract}} )</td>
</tr>
<tr>
<td>Inline Class (154)</td>
<td>( c_{\text{move}} )</td>
<td>( c_{\text{move}} )</td>
<td>( f_{\text{extract}} )</td>
<td>( f_{\text{extract}} )</td>
</tr>
<tr>
<td>Encapsulate Field (206)</td>
<td>( c_{\text{move}} )</td>
<td>( c_{\text{move}} )</td>
<td>( f_{\text{extract}} )</td>
<td>( f_{\text{extract}} )</td>
</tr>
<tr>
<td>Encapsulate Collection (208)</td>
<td>( c_{\text{move}} )</td>
<td>( c_{\text{move}} )</td>
<td>( f_{\text{extract}} )</td>
<td>( f_{\text{extract}} )</td>
</tr>
<tr>
<td>Replace Type Code with Subclass (223)</td>
<td>( c_{\text{move}} )</td>
<td>( c_{\text{move}} )</td>
<td>( f_{\text{extract}} )</td>
<td>( f_{\text{extract}} )</td>
</tr>
<tr>
<td>Replace Type Code with State/Strategy (227)</td>
<td>( c_{\text{move}} )</td>
<td>( c_{\text{move}} )</td>
<td>( f_{\text{extract}} )</td>
<td>( f_{\text{extract}} )</td>
</tr>
<tr>
<td>Replace Subclass with Field (232)</td>
<td>( c_{\text{move}} )</td>
<td>( c_{\text{move}} )</td>
<td>( f_{\text{extract}} )</td>
<td>( f_{\text{extract}} )</td>
</tr>
<tr>
<td>Decompose Conditional (238)</td>
<td>( m_{\text{attr}_\text{change}} )</td>
<td>( m_{\text{attr}_\text{change}} )</td>
<td>( p_{\text{create}}, r_{\text{create}} )</td>
<td>( p_{\text{create}}, r_{\text{create}} )</td>
</tr>
<tr>
<td>Replace Conditional with Polymorphism (255)</td>
<td>( c_{\text{move}} )</td>
<td>( c_{\text{move}} )</td>
<td>( f_{\text{extract}} )</td>
<td>( f_{\text{extract}} )</td>
</tr>
<tr>
<td>Introduce Null Object (280)</td>
<td>( c_{\text{move}} )</td>
<td>( c_{\text{move}} )</td>
<td>( f_{\text{extract}} )</td>
<td>( f_{\text{extract}} )</td>
</tr>
<tr>
<td>Add Parameter (275)</td>
<td>( m_{\text{move}} )</td>
<td>( m_{\text{move}} )</td>
<td>( p_{\text{extract}}, r_{\text{extract}} )</td>
<td>( p_{\text{delete}}, r_{\text{delete}} )</td>
</tr>
<tr>
<td>Remove Parameter (277)</td>
<td>( c_{\text{move}} )</td>
<td>( c_{\text{move}} )</td>
<td>( f_{\text{extract}} )</td>
<td>( f_{\text{extract}} )</td>
</tr>
<tr>
<td>Parameterize Method (283)</td>
<td>( c_{\text{move}} )</td>
<td>( c_{\text{move}} )</td>
<td>( f_{\text{extract}} )</td>
<td>( f_{\text{extract}} )</td>
</tr>
<tr>
<td>Preserve Whole Object (288)</td>
<td>( c_{\text{move}} )</td>
<td>( c_{\text{move}} )</td>
<td>( f_{\text{extract}} )</td>
<td>( f_{\text{extract}} )</td>
</tr>
<tr>
<td>Introduce Parameter Object (296)</td>
<td>( c_{\text{move}} )</td>
<td>( c_{\text{move}} )</td>
<td>( f_{\text{extract}} )</td>
<td>( f_{\text{extract}} )</td>
</tr>
<tr>
<td>Remove Setting Method (300)</td>
<td>( c_{\text{move}} )</td>
<td>( c_{\text{move}} )</td>
<td>( f_{\text{extract}} )</td>
<td>( f_{\text{extract}} )</td>
</tr>
<tr>
<td>Hide Method (303)</td>
<td>( m_{\text{attr}_\text{change}} )</td>
<td>( m_{\text{attr}_\text{change}} )</td>
<td>( p_{\text{create}}, r_{\text{create}} )</td>
<td>( p_{\text{create}}, r_{\text{create}} )</td>
</tr>
<tr>
<td>Pull Up Field (320)</td>
<td>( m_{\text{move}} )</td>
<td>( m_{\text{move}} )</td>
<td>( p_{\text{extract}}, r_{\text{extract}} )</td>
<td>( p_{\text{delete}}, r_{\text{delete}} )</td>
</tr>
<tr>
<td>Pull Up Method (322)</td>
<td>( c_{\text{move}} )</td>
<td>( c_{\text{move}} )</td>
<td>( f_{\text{extract}} )</td>
<td>( f_{\text{extract}} )</td>
</tr>
<tr>
<td>Push Down Method (328)</td>
<td>( c_{\text{move}} )</td>
<td>( c_{\text{move}} )</td>
<td>( f_{\text{extract}} )</td>
<td>( f_{\text{extract}} )</td>
</tr>
<tr>
<td>Push Down Field (329)</td>
<td>( c_{\text{move}} )</td>
<td>( c_{\text{move}} )</td>
<td>( f_{\text{extract}} )</td>
<td>( f_{\text{extract}} )</td>
</tr>
<tr>
<td>Extract Subclass (330)</td>
<td>( c_{\text{move}} )</td>
<td>( c_{\text{move}} )</td>
<td>( f_{\text{extract}} )</td>
<td>( f_{\text{extract}} )</td>
</tr>
<tr>
<td>Extract Superclass (336)</td>
<td>( c_{\text{move}} )</td>
<td>( c_{\text{move}} )</td>
<td>( f_{\text{extract}} )</td>
<td>( f_{\text{extract}} )</td>
</tr>
<tr>
<td>Collapse Hierarchy (344)</td>
<td>( c_{\text{move}} )</td>
<td>( c_{\text{move}} )</td>
<td>( f_{\text{extract}} )</td>
<td>( f_{\text{extract}} )</td>
</tr>
</tbody>
</table>

Operations

Impact

total 30

meeting at FAU, December 14, 2007
Pull Up Field: moves the same field appearing in sibling classes to their common superclass
Access levels of the moved field: private → protected
(The moved field might be used by the subclasses)
✓ Readable level: decreased
✓ Writable level: decreased
✓ Locality: no change
Remove Setting Method: deletes all methods which can modify a specified field
Access levels of the deleted method: public → (non-existence)
(No code can modify the value of the field related to the deleted methods)
✓ Readable level: no change
✓ Writable level: increased
✓ Locality: no change

<table>
<thead>
<tr>
<th>Refactoring (page number in Fowler’s catalog)</th>
<th>Readable level</th>
<th>Writable level</th>
<th>Locality</th>
<th>Subclassing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extract Method (110)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Inline Method (117)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Replace Temp with Query (120)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Replace Method with Method Object (135)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Move Method (142)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Remove Setting Method (300)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hide Method (303)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parameterize Method (308)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Preserve Whole Object (288)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Introduce Parameter Object (295)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pull Up Field (320)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pull Up Method (322)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Push Down Method (328)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Push Down Field (329)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Extract Subclass (330)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Extract Superclass (336)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Collapse Hierarchy (344)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

↑: surely increase, ↓: surely decrease, /: conditionally increase, \: conditionally decrease
c.ope: operation for a class, m.ope: operation for a method, f.ope: operation for a field
p.ope: operation for a parameter, r.ope: operation for a return value, l.ope: operation for a local variable
Table 1. The effects on the security level by the application of refactorings

<table>
<thead>
<tr>
<th>Refactoring (page number in Fowler’s catalog)</th>
<th>Readable level</th>
<th>Writable level</th>
<th>Locality</th>
<th>Subclassing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extract Method (110)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Inline Method (117)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Replace Temp with Query (120)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Replace Method with Method Object (135)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Preserve Whole Object (288)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Introduce Parameter Object (295)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Remove Setting Method (300)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hide Method (303)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pull Up Field (320)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pull Up Method (322)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Push Down Method (328)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Push Down Field (329)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Extract Subclass (330)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Extract Superclass (336)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Collapse Hierarchy (344)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Extract Method: extracts collective code fragments from an existing method and creates a new method including the extracted fragments
Locality of the variable: local variable → parameter
(A new parameter might be extracted)
Locality of the variable: local variable → return value
(A new return value might be extracted)
Readable level: no change
✓ Writable level: no change
✓ Locality: conditionally decreased

↑: surely increase, ↓: surely decrease, \: conditionally increase, \:\: conditionally decrease
c.ope: operation for a class, m.ope: operation for a method, f.ope: operation for a field
p.ope: operation for a parameter, r.ope: operation for a return value, l.ope: operation for a local variable

(3/4)
Move Method: moves a specified method in the class defining it to another class that frequently uses the method
Access levels of the moved method: **default/protected → public**
(The moved method might be used by the source class existing in the package not including the target class)
- Readable level: conditionally decreased
- Writable level: conditionally decreased
- Locality: no change

<table>
<thead>
<tr>
<th>Refactoring (page number in Fowler’s catalog)</th>
<th>Readable level</th>
<th>Writable level</th>
<th>Locality</th>
<th>Subclassing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extract Method (110)</td>
<td></td>
<td></td>
<td></td>
<td>(p_extract, r_extract)</td>
</tr>
<tr>
<td>Inline Method (117)</td>
<td></td>
<td></td>
<td></td>
<td>(p_delete, r_delete)</td>
</tr>
<tr>
<td>Replace Temp with Query (120)</td>
<td></td>
<td></td>
<td></td>
<td>(r_extract)</td>
</tr>
<tr>
<td>Replace Method with Method Object (135)</td>
<td></td>
<td></td>
<td></td>
<td>(f_extract)</td>
</tr>
<tr>
<td>Move Method (142)</td>
<td>( m\text{move} )</td>
<td>( m\text{move} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Move Field (146)</td>
<td>( f\text{move} )</td>
<td>( f\text{move} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Extract Class (149)</td>
<td>( c\text{extract} )</td>
<td>( c\text{extract} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Inline Class (154)</td>
<td>( c\text{inline} )</td>
<td>( c\text{inline} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Remove Setting Method (300)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hide Method (303)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pull Up Field (320)</td>
<td>( f\text{move} )</td>
<td>( f\text{move} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pull Up Method (322)</td>
<td>( m\text{move} )</td>
<td>( m\text{move} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Push Down Method (328)</td>
<td>( m\text{move} )</td>
<td>( m\text{move} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Push Down Field (329)</td>
<td>( f\text{move} )</td>
<td>( f\text{move} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Extract Subclass (330)</td>
<td>( m\text{move} )</td>
<td>( m\text{move} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Extract Superclass (336)</td>
<td>( f\text{move}, m\text{move} )</td>
<td>( f\text{move}, m\text{move} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Collapse Hierarchy (344)</td>
<td>( f\text{move}, m\text{move} )</td>
<td>( f\text{move}, m\text{move} )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

↑: surely increase, ↓: surely decrease, \( \_ \): conditionally increase, \( \_ \): conditionally decrease
c_ope: operation for a class, m_ope: operation for a method, f_ope: operation for a field
p_ope: operation for a parameter, r_ope: operation for a return value, l_ope: operation for a local variable
Conclusions

- **Secure refactoring** directly increases the security levels of existing code
  - Easily obtain more secure programs with the same behavior as before
- **Security-Aware Refactoring** reveals the impact of changes of existing code on security
  - Avoid unknowingly distributing dangerous code

Remaining issues:
- Complete the implementation of the procedure of each secure refactoring
- Formalize secure refactorings so that they never alter the behavior of existing code
- Explore many or more complex criteria for measuring the effects of the application of refactorings
  - Tried to introduce the concept of information flow
Thank you for your kind attention!

Katsuhisa Maruyama
Department of Computer Science
Ritsumeikan University
maru@cs.ritsumei.ac.jp

Any question and comment?
# Preliminary Experiment & Findings

## Table 1. The effects on the security level by the application of refactorings

<table>
<thead>
<tr>
<th>Refactoring (page number in Fowler’s catalog)</th>
<th>Readable level</th>
<th>Writable level</th>
<th>Locality</th>
<th>Subclassing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extract Method (110)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Inline Method (117)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Replace Temp with Query (120)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Replace Method with Method Object (135)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Move Method (142)</td>
<td>( m_{\text{move}} )</td>
<td>( m_{\text{move}} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Move Field (146)</td>
<td>( f_{\text{move}} )</td>
<td>( f_{\text{move}} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Extract Class (149)</td>
<td>( e_{\text{extract}} )</td>
<td>( e_{\text{extract}} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Inline Class (154)</td>
<td>( c_{\text{inline}} )</td>
<td>( c_{\text{inline}} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Encapsulate Field (206)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Encapsulate Collection (208)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Replace Type Code with Subclass (223)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Replace Type Code with State/Strategy (227)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Replace Subclass with Field (232)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Decompose Conditional (238)</td>
<td>( c_{\text{create}} )</td>
<td>( c_{\text{create}} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Replace Conditional with Polymorphism (255)</td>
<td>( p_{\text{extract}}, e_{\text{extract}} )</td>
<td>( p_{\text{extract}}, e_{\text{extract}} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Introduce Null Object (260)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add Parameter (275)</td>
<td>( m_{\text{attr}_change} )</td>
<td>( m_{\text{attr}_change} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Remove Parameter (277)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parameterize Method (283)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Preserve Whole Object (288)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Introduce Parameter Object (295)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Remove Setting Method (300)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hide Method (303)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pull Up Field (320)</td>
<td>( f_{\text{move}} )</td>
<td>( f_{\text{move}} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pull Up Method (322)</td>
<td>( m_{\text{move}} )</td>
<td>( m_{\text{move}} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Push Down Method (328)</td>
<td>( f_{\text{move}} )</td>
<td>( f_{\text{move}} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Push Down Field (329)</td>
<td>( m_{\text{move}} )</td>
<td>( m_{\text{move}} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Extract Subclass (330)</td>
<td>( f_{\text{move}}, m_{\text{move}} )</td>
<td>( f_{\text{move}}, m_{\text{move}} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Extract Superclass (336)</td>
<td>( f_{\text{move}}, m_{\text{move}} )</td>
<td>( f_{\text{move}}, m_{\text{move}} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Collapse Hierarchy (344)</td>
<td>( c_{\text{create}} )</td>
<td>( c_{\text{create}} )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

↑: surely increase, ↓: surely decrease, \( / \): conditionally increase, \( \backslash \): conditionally decrease

c_ope: operation for a class, m_ope: operation for a method, f_ope: operation for a field
p_ope: operation for a parameter, r_ope: operation for a return value, l_ope: operation for a local variable