DEPARTMENT OF COMPUTER SCIENCE AND SOFTWARE ENGINEERING
CONCORDIA UNIVERSITY

Refactoring

Refactoring

:: What is it?

Definition: Refactoring is a disciplined technique for *restructuring* an existing body of code, altering its internal structure *without changing its external behavior*.

Refactoring **does not fix bugs**, but it may help find bugs by scrutinizing code. It may also reduce further bug occurrence by cleaning-up code.

Refactoring **does not add new functionality** to the system, but it will ease the further adding of new functionality.

Refactoring

:: when?

Refactoring ought to be done continuously as “bad smells” are encountered during programming.

More importantly, when using iterative development, a major refactoring stage should precede the beginning of the development of a new build. This will remove slight design problems and ease the addition of further functionality.

In this case, refactoring counterbalances the productivity-driven software development practices implied by agile incremental software development.

Refactoring

:: why?

Refactoring is usually done to:

improve code readability & comprehensibility

simplify code structure, improve design quality

improve maintainability

improve extensibility.

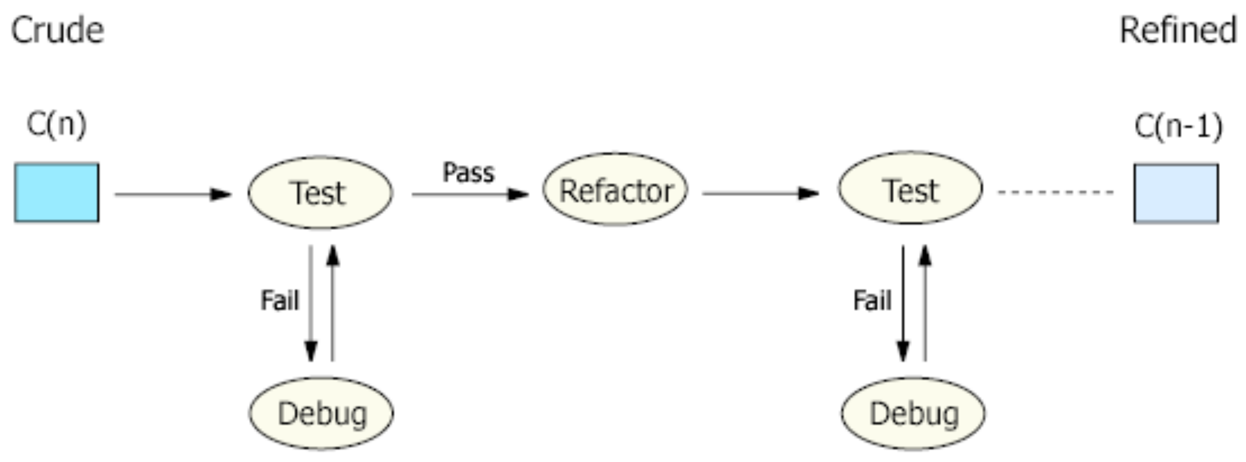
Refactoring

:: how is it done?

Each refactoring is implemented as a small behavior-preserving transformation.

Behavior-preservation is achieved through pre- and post-transformation testing.

Refactoring process: **test-refactor-test**



C(x) := Code with x Number of Smells

Refactoring

:: drawbacks

Cost Overhead: Refactoring is an add-on activity and therefore will incur extra cost in form of time, effort, and resource allocation, especially if elaborated design and code documentation is maintained. However, when done sparingly and only on key issues, its benefits are greater than its overhead. Automated documentation tools will also diminish the overhead.

Requires Expertise: Refactoring requires some expertise and experience and considerable effort in going through the process, especially if proper testing is involved. However, this overhead can be minimized by using automated testing such as with a unit testing framework.

Refactoring

:: examples

Encapsulate Downcast: A method returns an object that needs to be downcasted by its callers. Refactor by moving the downcast to within the method.

```
Object lastReading() {
   return readings.lastElement();
}

Reading lastReading() {
   return (Reading) readings.lastElement();
}
```

:: examples

Consolidate Conditional Expression: You have a sequence of conditional tests with the same result. Refactor by combining them into a single conditional expression and extract it.

```
double disabilityAmount() {
   if (_seniority < 2) return 0;
   if (_monthsDisabled > 12) return 0;
   if (_isPartTime) return 0;
   // compute the disability amount

double disabilityAmount() {
   if (isNotEligibleForDisability()) return 0;
   // compute the disability amount
```

Refactoring

:: examples

Consolidate Duplicate Conditional Fragments: The same fragment of code is in all branches of a conditional expression. Refactor by moving it outside of the expression.

```
if (isSpecialDeal()) {
    total = price * 0.95;
    send();
} else {
    total = price * 0.98;
    send();
}
```

```
if (isSpecialDeal())
    total = price * 0.95;
else
    total = price * 0.98;
send();
```

Refactoring

:: examples

Rename Method: The name of a method does not reveal its purpose. Refactor it by changing the name of the method.

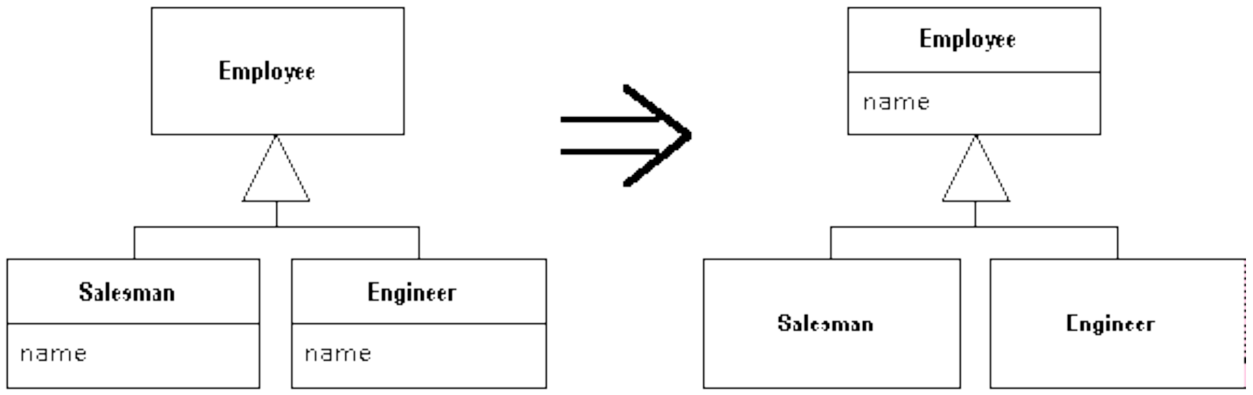
```
int getInvCdtLmt(){
…
}
```

```
int getInvoiceableCreditLimit(){
…
}
```

Refactoring

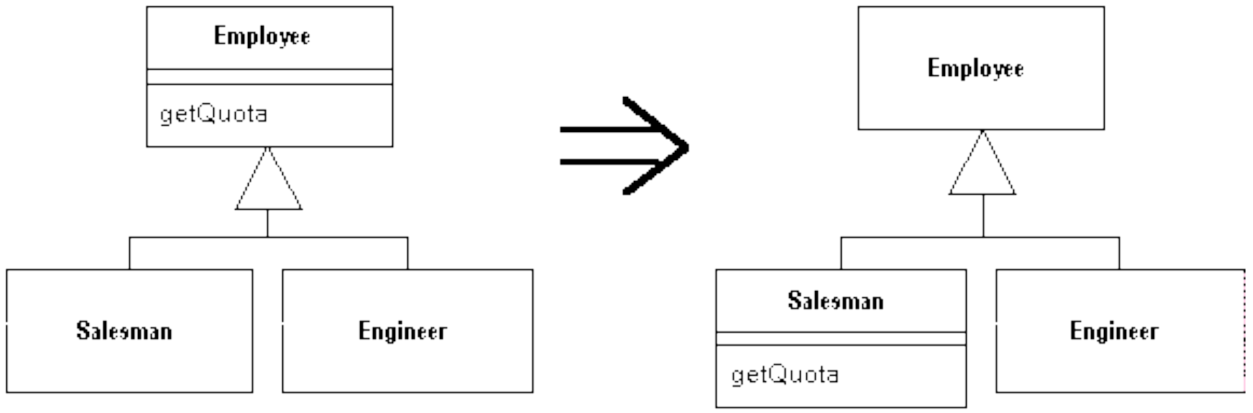
:: examples

Pull Up Field: Two subclasses have the same field. Refactor it by moving the field to the superclass.



:: examples

Push Down Method: Behavior on a superclass is relevant only for some of its subclasses. Refactor it by moving it to those subclasses.



:: practice

- Some refactorings are controversial.
- Some refactorings are arguably not improving code quality.
- Some refactorings can in fact be counter-productive when applied blindly, especially in iterative development, where design is evolving.

Have your team adopt a set of refactorings to be applied, and make sure that refactorings are applied in a productive manner.

Apply in combination with the application of design patterns.

Use refactoring tools to automate changes, e.g. Eclipse

Resources

- Catalog of Refactorings. http://www.refactoring.com/catalog/index.html
- Refactoring Home Page. http://www.refactoring.com/
- Martin Fowler. Refactoring: Improving the Design of Existing Code, Addison-Wesley, 1999. ISBN 0-201-48567-2.
- Kerievsky, Joshua. Refactoring To Patterns. Addison-Wesley, 2004. ISBN 0-321-21335-1.
- Wake, William C.. Refactoring Workbook. Addison-Wesley, 2003. ISBN 0-321-10929-5.