Refactoring - Formalisms and Examples

Idaho State University | Computer Science

Isaac Griffith

CS 4423 and CS 5523
Department of Computer Science
Idaho State University

ROAR

Idaho State University
Computer Science

Outcomes

After today's lecture you will:

- Be able to describe methods of formalizing refactorings
  - Assertions
  - Graph Transformations
  - Software Metrics
- Understand the effects of refactoring through example

ROAR

Refactoring Formalisms

CS 4423/5523

ROAR

Formalisms for Refactoring

- Three key formalisms for refactoring are:
  - assertions:
    - Assertions are useful in verifying the assumptions made by programmers.
  - graph transformation:
    - Graph transformation is useful in viewing refactorings as applications of transformation rules.
  - metrics:
    - Metrics are useful in quantifying to what extent the internal and external properties of software entities have changed.

Assertions

- Programmers make assumptions about the behavior of programs at specific points, and those assumptions can be tested by means of assertions.
- An assertion is specified as a Boolean expression which evaluates to true or false.
- Three kinds of assertions:
  - invariants;
  - preconditions; and
  - postconditions.

Assertions

- Invariant
  - An **invariant** is an assertion that evaluates to **true** wherever in the program it is invoked.
  - A **class invariant** is an invariant that all instances of that class must satisfy.
- Precondition
  - A **precondition** is a condition that must be satisfied **before** a computation is performed.
- Postcondition
  - A **postcondition** is a condition that must be satisfied **after** a computation is performed.

Assertions

- Invariants, preconditions, and postconditions can be applied to test the behavior preserving property of refactorings.
- Examples of invariant in the context of transformation of database schema is:
  - All instance variables of a class, whether defined or inherited, have distinct names.
  - All methods of a class, whether defined or inherited, have distinct names.
- **Note:** Static checking of preconditions, postconditions, and invariants is computationally expensive.

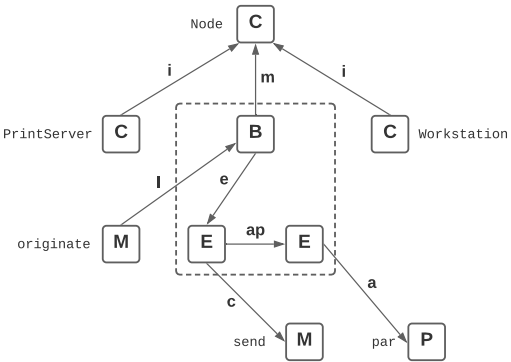
Graph Transformation

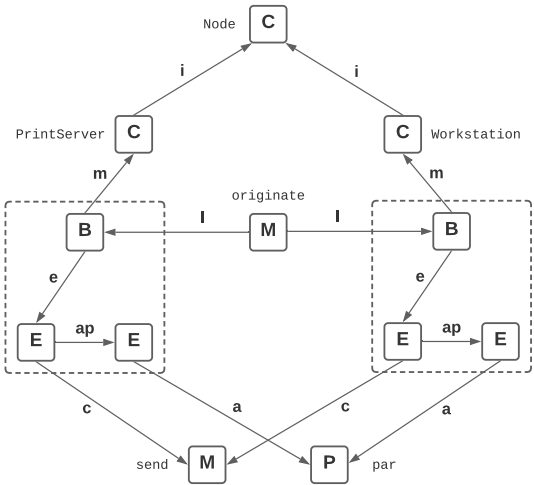
- Programs, class diagrams, and statecharts can be viewed as **graphs**, and refactorings can be viewed as **graph production rules**.
- Classes (C), method signatures (M), block structures (B), variables (V), parameters (P), and expressions (E) are represented by **typed nodes** in a graph.
- The possible relationships among the nodes are:
  - method lookup (l)
  - membership (m)
  - expression (e)
  - formal parameter (fp)
  - call (c)
  - update (u)
  - inheritance (i)
  - (sub)type (t)
  - actual parameter (ap)
  - cascaded expression (·)
  - variable access (a)

Idaho State University
Computer Science

Graph Transformation

- The **Push-Down-Method** refactoring has been applied to method **originate** to obtain a new graph





Software Metrics

- Software metrics can be used to quantify the internal and external qualities of software.
- A module consists of many components; each component provides a defined functionality used by other components.
- Measure the **strength of togetherness** of components within a module to decide whether or not some components should stay in the same module.

Software Metrics

- Two metrics considered are:
  - cohesion
  - coupling
- **Cohesion:** This metric is used to represent the strength of togetherness in the same module.
- **Coupling:** This metric is used to represent the strength of dependency between separate modules.

More Refactoring Examples

CS 4423/5523

ROAR

More Examples of Refactoring

- More examples are intuitively explained here.
  - Substitute algorithm;
  - Replace parameter with methods;
  - Push Down Method;
  - Parameterize Methods;

Substitute algorithm

- Replace algorithm `X` with algorithm `Y` because:
  1. implementation of `Y` is clearer than `X`
  2. `Y` performs better than `X`
  3. standardization bodies want `X` to be replaced with `Y`
- Algorithm substitution is easier if both `X` and `Y` have the same input-output behaviors.

Replace parameters with methods

Consider the following code segment, where the method `bodyMassIndex` has two formal parameters.

```
int person;
:
// person is initialized here;
:
int bodyMass = getMass(person);
int height = getHeight(person);
int BMI = bodyMassIndex(bodyMass, height);
:
```

- The above code segment can be rewritten such that the new `bodyMassIndex` method accepts one formal parameter, namely, `person`, and internally computes the values of `bodyMass`and `height`.

Replace parameters with methods

- The refactored code segment has been shown in the following:

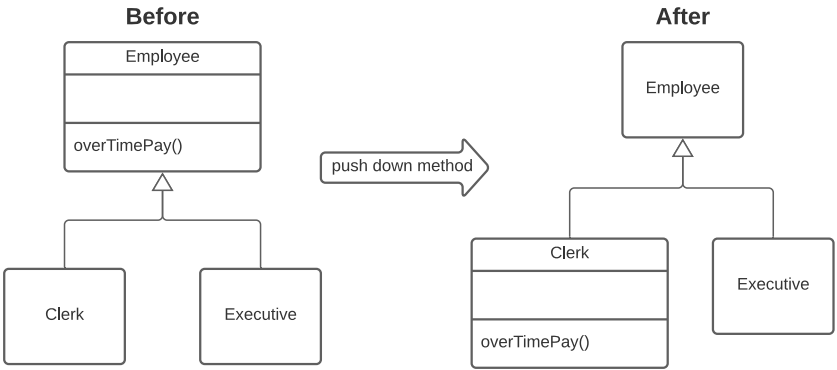
```
int person;
:
// person is initialized here;
:
int BMI = bodyMassIndex(person);
:
```

- The advantage of this refactoring is that it reduces the number of parameters passed to methods.
- Such reduction is important because one can easily make errors while passing long parameter lists.

Idaho State University
Computer Science

Push Down Method

- Assume that `Executive` and `Clerk` are two subclasses of the superclass `Employee`
- Method `overTimePay` has been defined in `Employee` class
- If `overTimePay` is used in the `Clerk` class, but not in the `Executive` class, then the programmer can push down `overTimePay` to the `Clerk` class



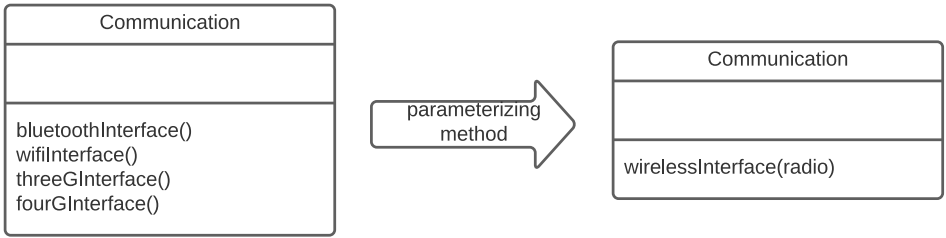
ROAR

Parameterize Methods

- Sometimes programmers may find multiple methods performing the same computations on different input data sets.
- Those methods can be replaced with a new method with additional formal parameters.

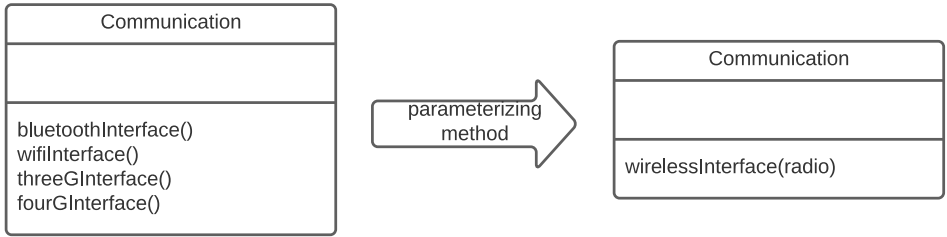
Parameterize Methods

- We start with the `Communication` class with four methods: `bluetoothInterface`, `wifiInterface`, `threeGInterface`, and `fourGInterface`.



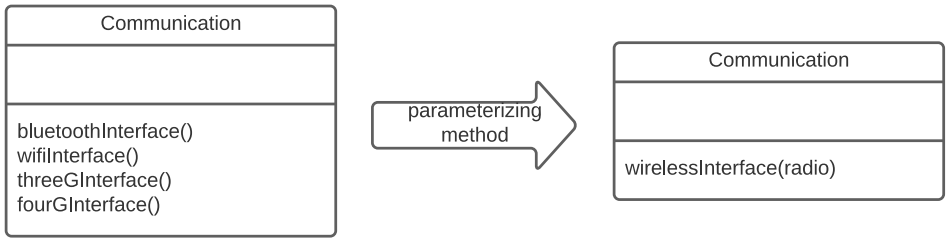
Parameterize Methods

- After refactoring we have the `Communication` class with just one method, `wirelessInterface`, with one parameter, `radio`.



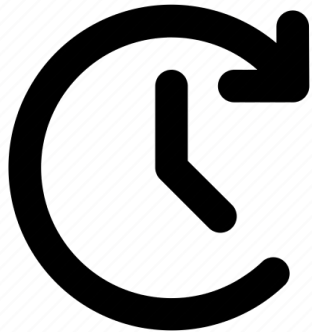
Parameterize Methods

- The method `wirelessInterface` can be invoked with different values of `radio` so that the `wirelessInterface` method can in turn invoke different `radio` interfaces.



For Next Time

- Review EVO Chapter 7.3 - 7.4
- Read EVO Chapter 7.5 - 7.6
- Watch Lecture 18



Are there any questions?