Faculty of Computers and Information
Joint Master In Software Engineering Program (JMSE)

Dr. Lamia Abo Zaid

د. لمياء أبوزيد

Based on slides by © Tripathy & Naik

Software Evolution : TOC

1. Introduction to Software Evolution
2. Taxonomy of Software Maintenance and Evolution
3. Evolution and Maintenance Models
4. Reuse and Domain Engineering
5. Program Comprehension
6. Impact Analysis
7. Refactoring
8. Reengineering
9. Legacy Information Systems

Restructuring Code

- The complexity of software can be decreased by improving its internal quality by restructuring the software.
- Software should be continually restructured during and between other maintenance activities so that programmers find it easier and easier to work with it.
- Source code is restructured to improve some of its non-functional requirements, without modifying its functional requirements
  - Readability
  - Extensibility
  - Maintainability
  - Modularity

Refactoring Code

- Restructuring applied on object-oriented software is called refactoring.
- Restructuring means reorganizing software (source code + documentation) to give it a different look, or structure.
- A higher level goal of restructuring is to increase the software value
  - External software value: fewer faults in software is seen to be better by customers
  - Internal software value: a well-structured system is less expensive to maintain
- Simple examples of restructuring are:
  - Pretty printing
  - Meaningful names for variables
  - One statement per line of source code

Activities In a Refactoring Process

The refactoring process has the following activities:

1. Identify what to refactor.
2. Determine which refactoring(s) to apply.
3. Ensure that refactoring preserves the software's behaviour.
4. Apply the refactoring(s) to the chosen entities.
5. Evaluate the impacts of the refactoring(s).
6. Maintain consistency.

Activities In a Refactoring Process - Identify What to Refactor.

1. The programmer identifies what to refactor from a set of high-level software artifacts.
   - source code
   - design documents
   - requirements documents.
2. Next, focus on specific portions of the chosen artifact for refactoring.
   - Specific modules, functions, classes, methods, and data can be identified for refactoring

Activities In a Refactoring Process - Identify What to Refactor.

- The concept of code smell is applied to source code to detect what should be refactored.
- A code smell is any symptom in source code that possibly indicates a deeper problem.
- Examples of code smell are:
  - Duplicate code
  - long parameter list
  - Long methods
  - Large classes
  - Message chain. E.g. student.getID().getRecord().getGrade(course) → student.getGrade(course)

Activities In a Refactoring Process - Determine Which Refactoring(s) Should be Applied

R1: Rename method print to process in class PrintServer.

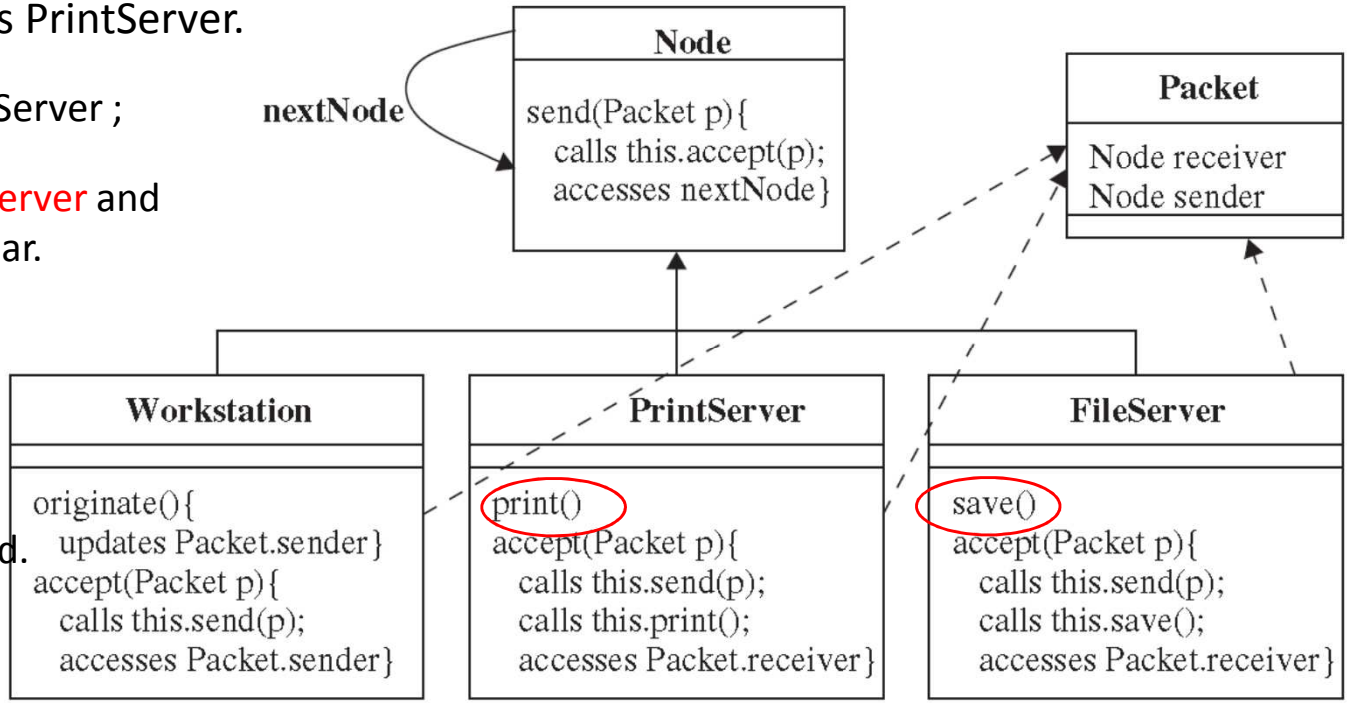
R2: Rename method save to process in class FileServer ;

R3: Create a superclass called Server from PrintServer and FileServer, because their behaviors are very similar.

R4: Pull up method accept from classes PrintServer and FileServer to the superclass Server created with R3.

R5: Encapsulate field receiver in class Packet, so that another class cannot directly access this field. advantages are:

1. increased modularity of the system
2. data packets can be modified without modifying the classes that use data packets.

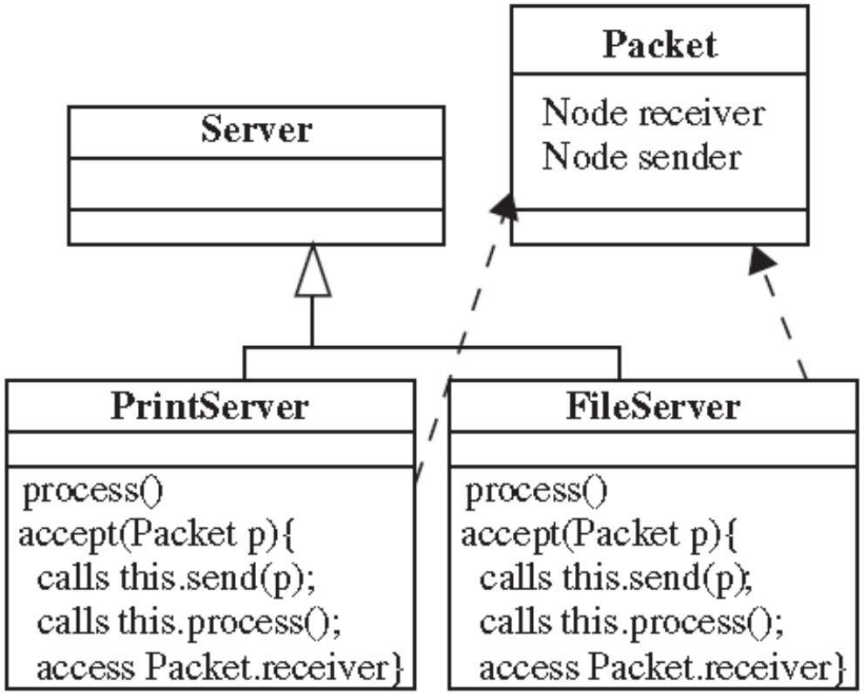


Activities In a Refactoring Process - Determine Which Refactoring(s) Should be Applied

R1: Rename method print to process in class PrintServer.

R2: Rename method save to process in class FileServer ;

R3: Create a superclass called Server from PrintServer and FileServer, because their behaviors are very similar.



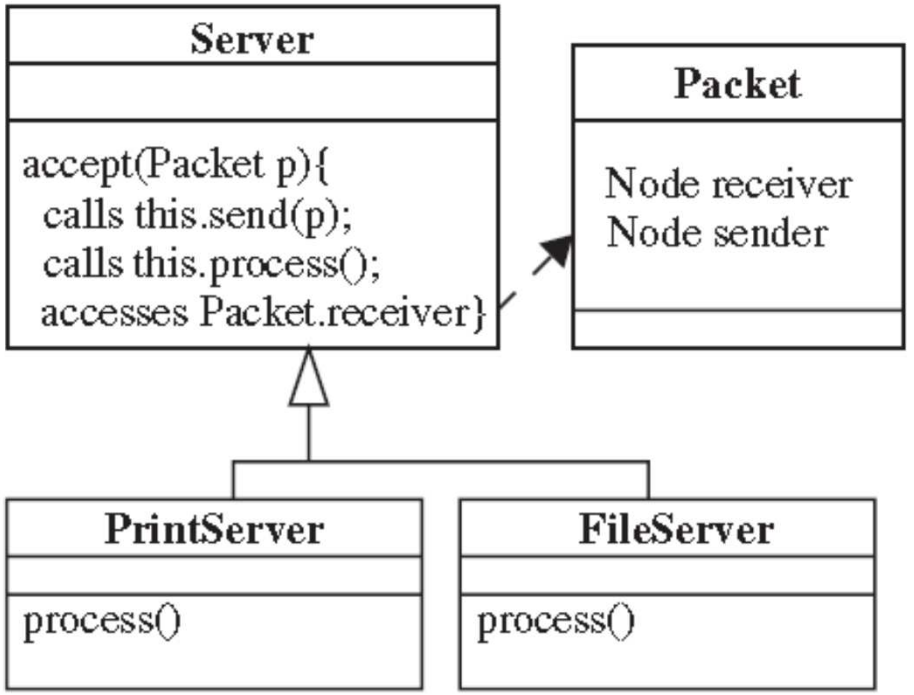
Activities In a Refactoring Process - Determine Which Refactoring(s) Should be Applied

R1: Rename method print to process in class PrintServer.

R2: Rename method save to process in class FileServer ;

R3: Create a superclass called Server from PrintServer and FileServer, because their behaviors are very similar.

R4: Pull up method accept from classes PrintServer and FileServer to the superclass Server created with R3.



Activities In a Refactoring Process - Determine Which Refactoring(s) Should be Applied

R1: Rename method print to process in class PrintServer.

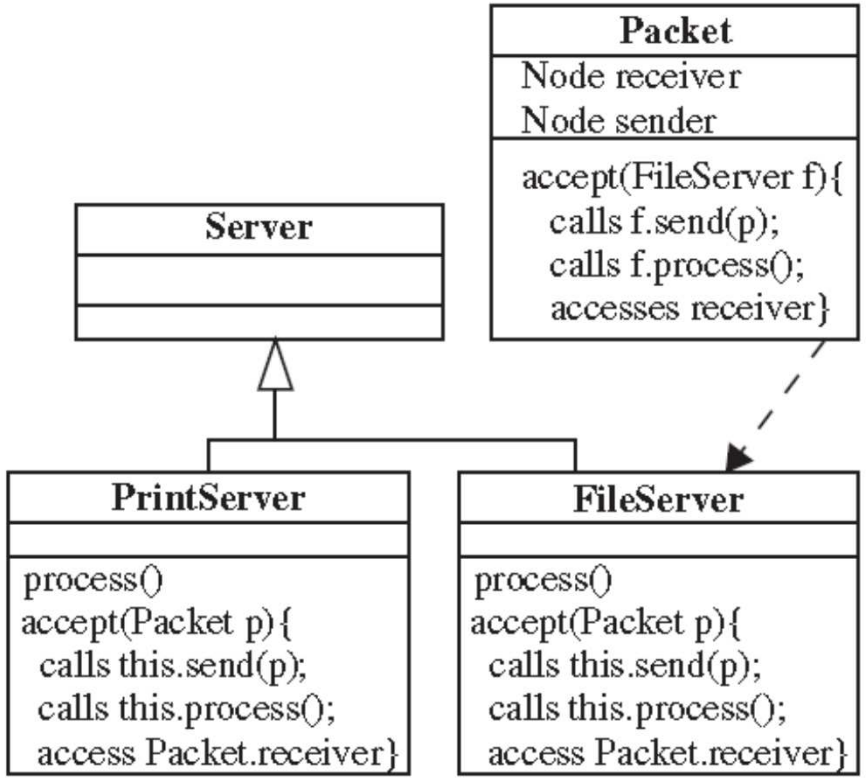
R2: Rename method save to process in class FileServer ;

R3: Create a superclass called Server from PrintServer and FileServer, because their behaviors are very similar.

R4: Move method accept from class PrintServer to class Packet, because method accept directly accesses the field receiver in class Packet.

- An advantage of moving accept from PrintServer to class Packet is that data packets themselves will decide what actions to take.

R5: Move method accept from class FileServer to class Packet .



Activities In a Refactoring Process - Determine Which Refactoring(s) Should be Applied

- A subset of the entire set of refactorings need to be carefully chosen because of the following reasons.
  - Some refactorings must be applied together.
    - Example: R1 and R2 are to be applied together.
  - Some refactorings must be applied in certain orders.
    - Example: R1 and R2 must precede R3.
  - Some refactorings can be individually applied, but they must follow an order if applied together.
  - Some refactorings are mutually exclusive.

Activities In a Refactoring Process - Determine Which Refactoring(s) Should be Applied

- Tool support is needed to identify a feasible subset of refactorings.
- The following two techniques can be used to analyze a set of refactorings to select a feasible subset.

1. Critical pair analysis

- Given a set of refactorings, analyze each pair for conflicts. A pair is said to be conflicting if both of them cannot be applied together.
  - Example: R4 and R6 constitute a conflicting pair.

2. Sequential dependency analysis

- In order to apply a refactoring, one or more refactorings must be applied before.
- If one refactoring has already been applied, a mutually exclusive refactoring cannot be applied anymore.
  - Example: after applying R1, R2, and R3, R4 becomes applicable. Now, if R4 is applied, then R6 is not applicable anymore.

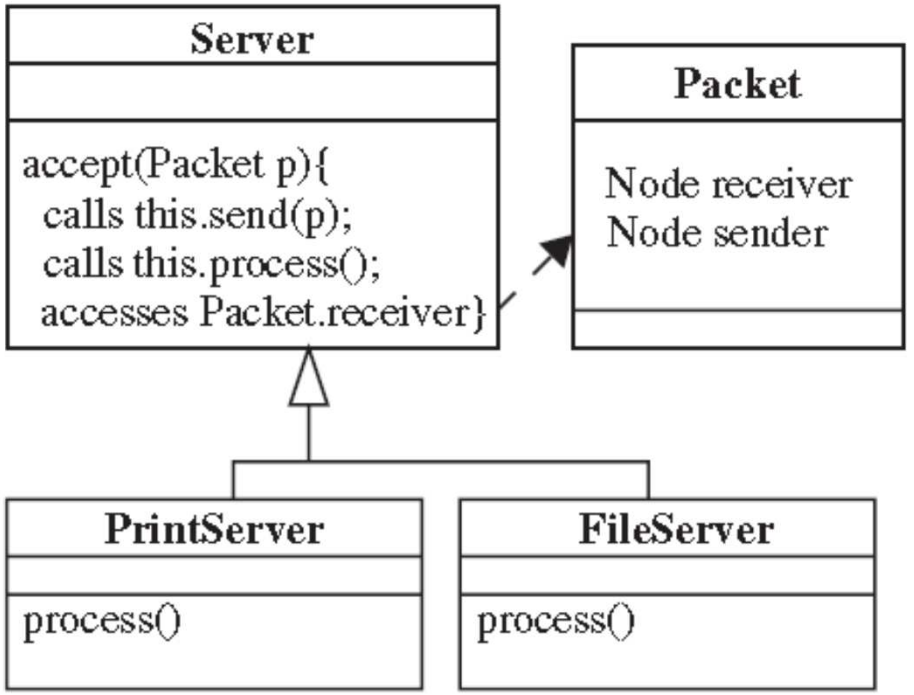
Activities In a Refactoring Process - Apply The Refactoring(s) to The Chosen Entities

R1: Rename method print to process in class PrintServer.

R2: Rename method save to process in class FileServer ;

R3: Create a superclass called Server from PrintServer and FileServer, because their behaviors are very similar.

R4: Pull up method accept from classes PrintServer and FileServer to the superclass Server created with R3.



Activities In a Refactoring Process - Ensure That Refactoring Preserves The Software's Behavior

- Ideally, the input/output behaviour of a program after refactoring is the same as the behaviour before refactoring.
- In many applications, preservation of non-functional requirements is necessary.
- A non-exclusive list of such non-functional requirements is as follows:
  - Temporal constraints: A temporal constraint over a sequence of operations is that the operations occur in a certain order.
    - For real-time systems, refactorings should preserve temporal constraints.
  - Resource constraints: The software after refactoring does not demand more resources: memory, energy, communication bandwidth, and so on.
  - Safety constraints: It is important that the software does not lose its safety properties after refactoring.

Activities In a Refactoring Process - Ensure That Refactoring Preserves The Software's Behavior

Two pragmatic ways of showing that refactoring preserves the software's behavior.

- Testing
  - Exhaustively test the software before and after applying refactorings, and compare the observed behavior on a test-by-test basis.
- Verification of preservation of call sequence
  - Ensure that the sequence(s) of method calls are preserved in the refactored program.

Activities In a Refactoring Process - Evaluate The Impacts Of The Refactorings

- Refactorings impact both internal and external qualities of software.
- Some examples of internal qualities of software are:
  - size, complexity, coupling, cohesion, and testability
- Some examples of external qualities of software are
  - performance, reusability, maintainability, extensibility, robustness, and scalability

Activities In a Refactoring Process - Evaluate the impacts of the Refactorings

- Refactoring techniques are highly specialized, with one technique improving a small number of quality attributes.
- For example:
  - some refactorings eliminate code duplication
  - some raise reusability
  - some improve performance
  - some improve maintainability.

Activities In a Refactoring Process -Maintain Consistency of Software Artifacts

- A software system is described by many artifacts at different levels of abstractions.
  - Those artifacts include requirements documents, design documents, source code, and test suites.
- If one kind of artifact is changed, then it is important to change some or all of the other artifacts so that consistency is maintained across the artifacts.
  - For example, changes in source code may require changes in the design documents and the test suites.
- The concept of change propagation is used to deal with inconsistencies across different software artifacts

More Examples of Refactoring

- More examples are:

1. Substitute algorithm
2. Replace parameter with methods
3. Push Down Method
4. Parameterize Methods

More Examples of Refactoring 1

- Substitute algorithm
  - Replace algorithm X with algorithm Y

  Because

  - implementation of Y is clearer than X;
  - Y performs better than X; and (iii) standardization bodies want X to be replaced with Y.
  - Algorithm substitution is easier if both X and Y have the same input-output behaviors.

More Examples of Refactoring 2

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 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.

More Examples of Refactoring 2

- The refactored code segment :

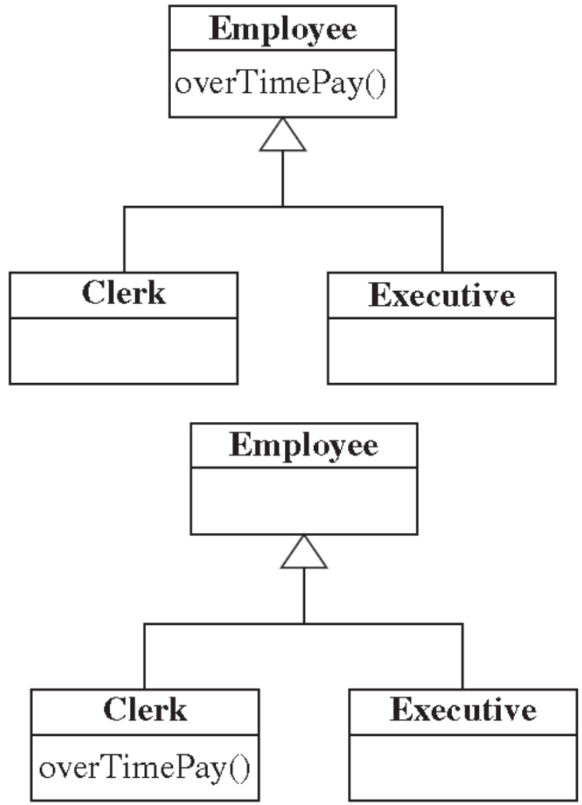
```
int person;
        :
        // person is initialized here;
        :
        int BMI = bodyMassIndex(person);
```

- The advantage of this refactoring
  - It reduces the number of parameters passed to methods.
  - Avoids errors while passing long parameter lists.

More Examples of Refactoring 3

Push Down Method

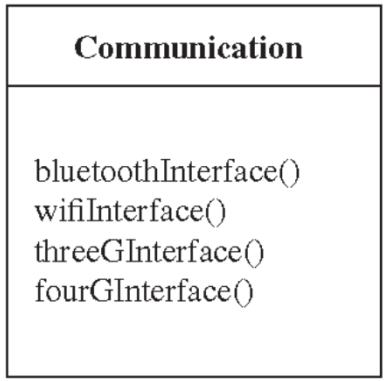
- Assume that Executive and Clerk are two subclasses of the superclass Employee
- Method overTimePay has been defined in Employee class.
- If overTimePay is used in the Clerk class, but not in the Executive class, then the programmer can push down overTimePay to the Clerk class

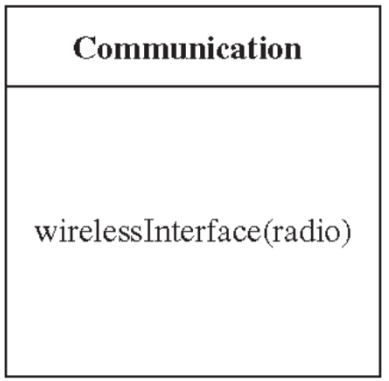


More Examples of Refactoring 4

❑ 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.
- The Communication class with four methods: bluetoothInterface, wifiInterface, threeGInterface, and fourGInterface.
- The Communication class with just one method wirelessInterface with one parameter, namely, radio.
- The method wirelessInterface can be invoked with different values of radio so that the wirelessInterface method can in turn invoke different radio interfaces.





Questions

?