SE 323 - Software Construction Testing and Maintenance

MODULE ONE
SOFTWARE CONSTRUCTION

CHAPTER THREE
IMPLEMENTING SOFTWARE CONSTRUCTION
PART ONE

SOFTWARE CONSTRUCTION PROCESS AND ACTIVITIES

Module 1
Software Construction

OUTLINE

1. ~~*Overview of Software Construction*~~
2. ~~*Planning/Preparing for Software Construction*~~
3. *Doing Software Construction*
4. ~~*Documenting Software Construction*~~
5. ~~*Measuring/Monitoring Software Construction*~~

Construction Practices -Implementation

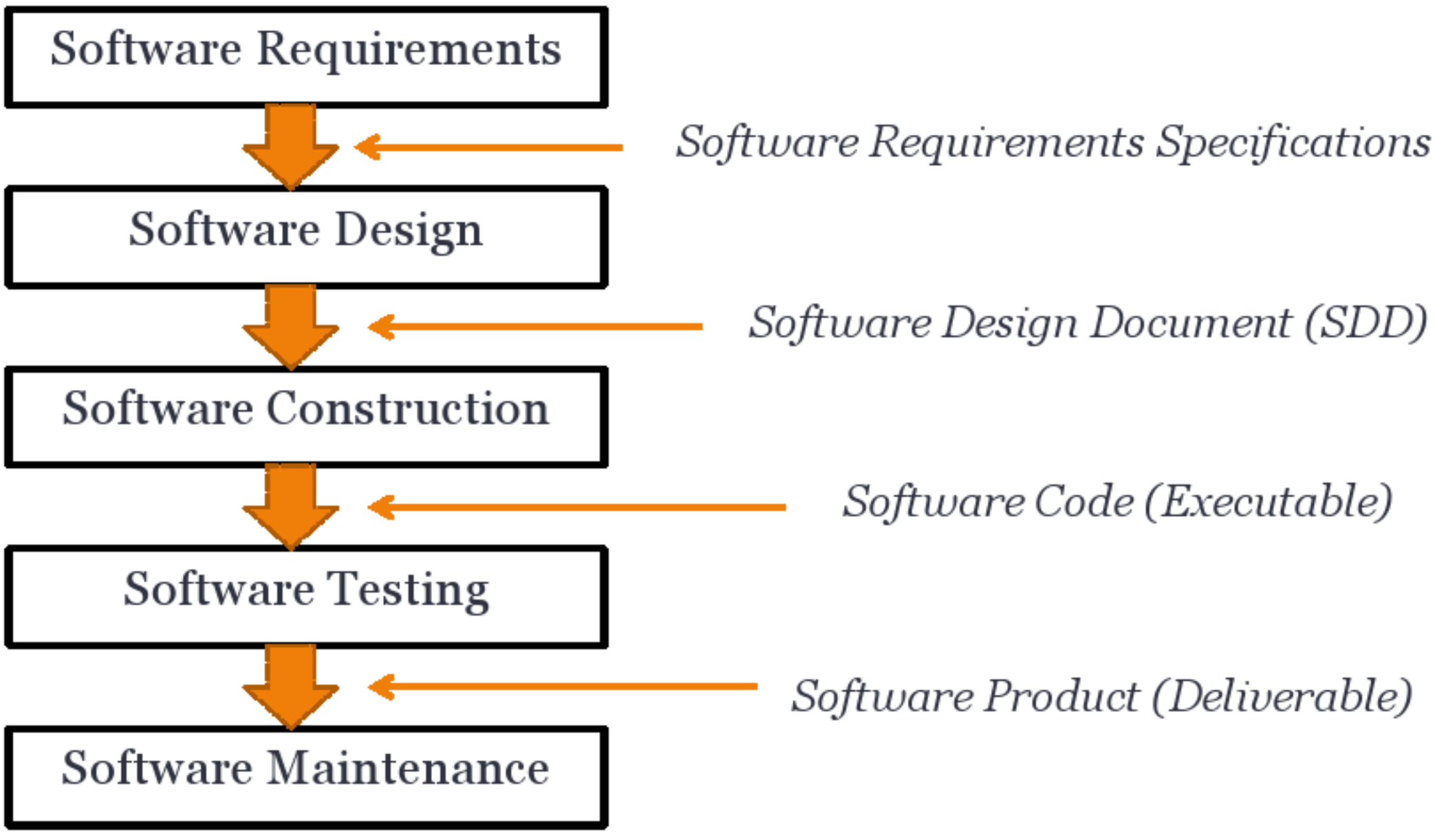
Module 1
Software Construction

OUTLINE

1. ~~*Overview of Software Construction*~~
2. ~~*Planning/Preparing for Software Construction*~~
3. *Doing Software Construction*
   - 3.1 *Software Construction Process & Activities*
   - 3.2 *Software Construction Guidelines & Practices*
4. ~~*Documenting Software Construction*~~
5. ~~*Measuring/Monitoring Software Construction*~~

Construction Practices -Implementation

3.1 Construction Practices -Process & Activities



SC Planning – SDLC – Design

Design Document - SDD (ref: IEEE 1016-1998)

1. Introduction
- 1.1 Purpose
- 1.2 Scope
- 1.3 Definitions, acronyms, and abbreviations

2. References

3. Decomposition Description
- 3.1 Module decomposition
  - 3.1.1 Module 1 description
  - 3.1.2 Module 2 description
- 3.2 Concurrent process decomposition
  - 3.2.1 Process 1 description
  - 3.2.2 Process 2 description
- 3.3 Data decomposition
  - 3.3.1 Data entry 1 description
  - 3.3.2 Data entry 2 description

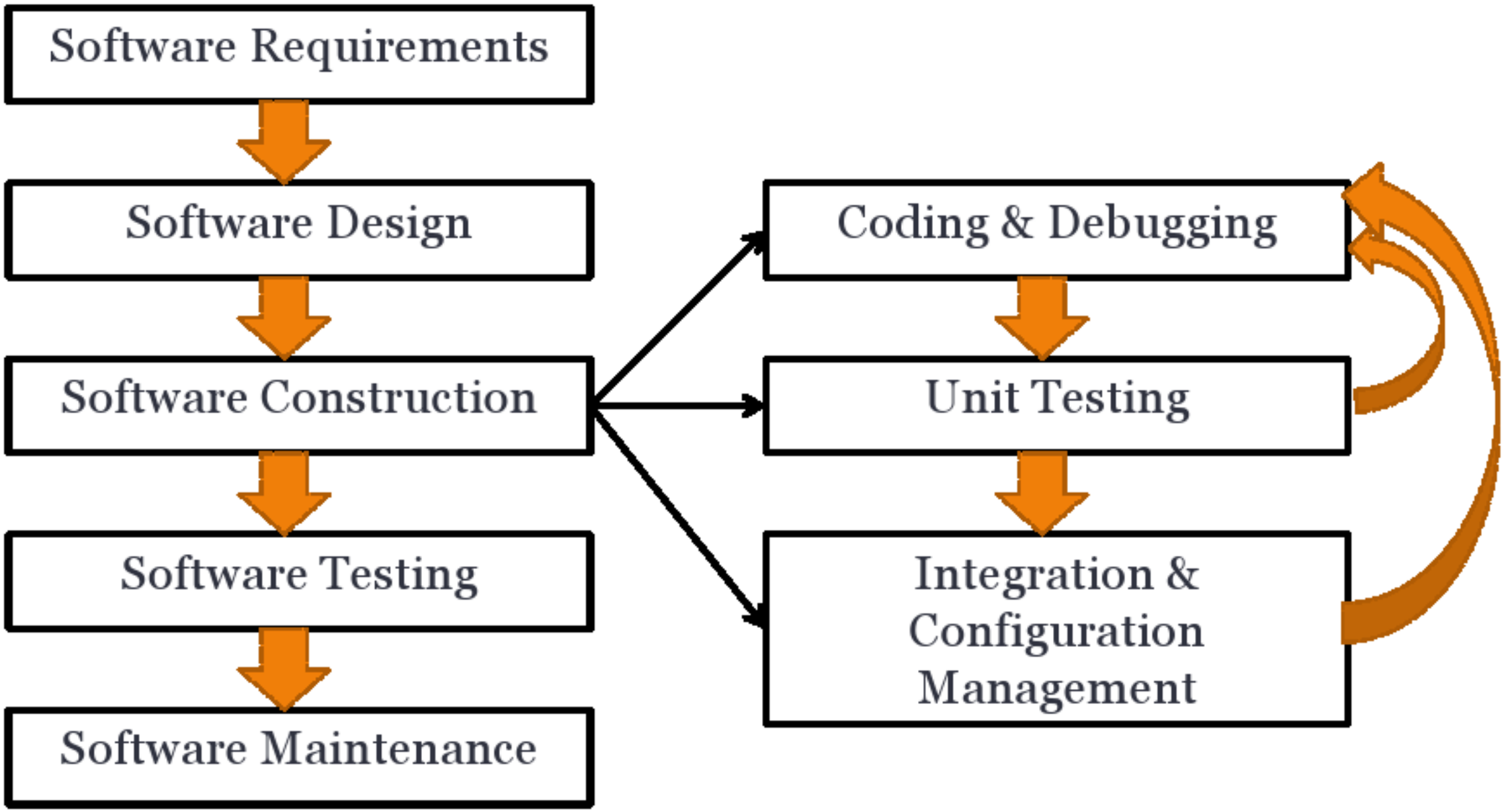
4. Dependency description
- 4.1 Intermodule dependencies
- 4.2 Interprocess dependencies
- 4.3 Data dependencies

5. Interface Description
- 5.1 Module interface
  - 5.1.1 Module 1 description
  - 5.1.2 Module 2 description
- 5.2 Process interface
  - 5.2.1 Process 1 description
  - 5.2.2 Process 2 description

6. Detailed Design
- 6.1 Module detailed design
  - 6.1.1 Module 1 detail
  - 6.1.2 Module 2 detail
- 6.2 Data detailed design
  - 6.2.1 Data entry 1 detail
  - 6.2.2 Data entry 2 detail

Construction Practices -Implementation

3.1 Construction Practices -Process & Activities



Construction Practices -Activities

3.1 Construction Practices -Process & Activities

3.1.1 Coding & Unit Testing

~~3.1.2 Integration & Integration Test~~

~~3.1.3 Configuration Management~~

3.1.1 Software Construction Activities – Integration

Software Programming

- Coding (Programming)

  Design Specs (Document) → Software Code (Program)
  Design Elements → Program/Software Modules

- Integration

  Program/Software Modules → Integrated Software/System

3.1.1 Software Construction Activities - Coding

Software Programming

- Coding Activities
  - Based on SDD*, translating the design specification into the program ***source code***
    - Based on HLD*, implementing the program ***overall structure and interfaces***
    - Based on LLD*, constructing the program ***modules*** which correspond to the specified ***design elements***
  - Compiling, debugging and ***unit testing*** the coded modules

---

* *SDD: Software Design Document; HLD: High Level Design Document; LLD: Low Level Design Document*

3.1.1 Software Construction Activities – Coding

Software Programming

- Coding Example

Implementing OO design elements:

- System Structural Modelling

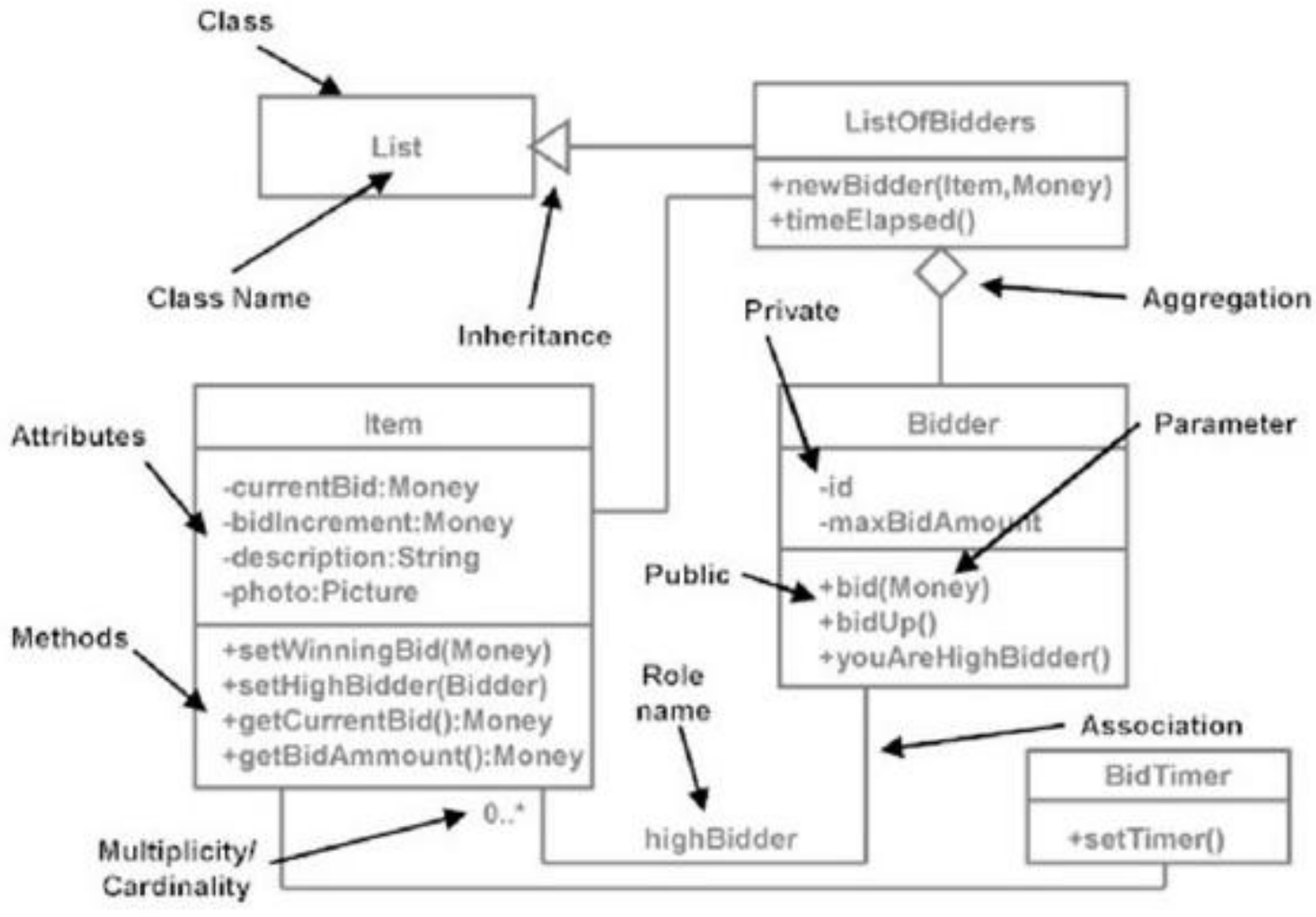
  e.g. UML Class Diagrams
- System Behavioural Modelling

  e.g. UML Sequence Diagrams

Construction Practices - Coding

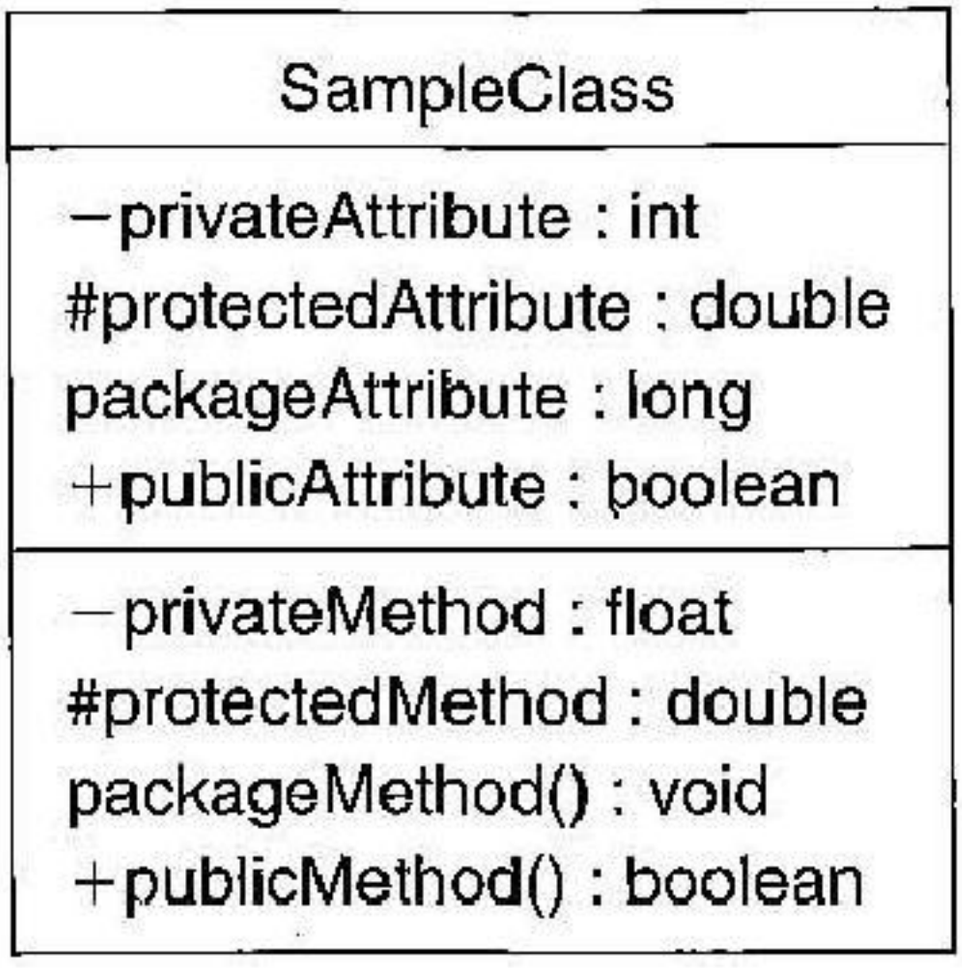
3.1.1 Software Construction Activities - Coding

- Coding Example – UML Class Diagram



Construction Practices -Coding

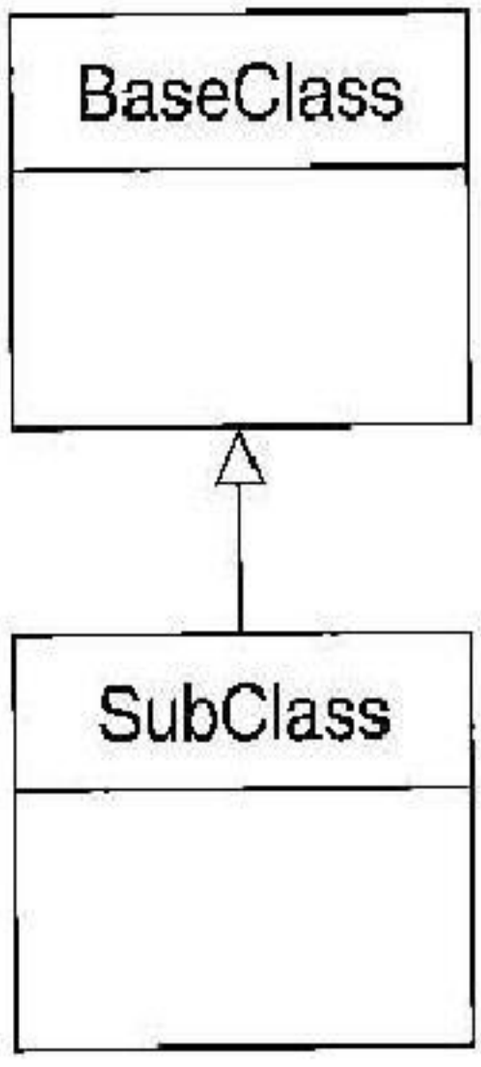
Coding Example: UML class diagram




```
class SampleClass {
    private int privateAttribute;
    protected double protectedAttribute;
    long packageAttribute;
    public boolean publicAttribute;
    public boolean publicMethod(int parameter1) {
        ...
        ...
    }
    private float  privateMethod(byte parameter1, float parameter2) {
        ...
        ...
    }
    protected double protectedMethod() {
        ...
        ...
    }
    void packageMethod(short parameter1) {
        ...
        ...
    }
}   Sample Class
```

Construction Practices -Coding

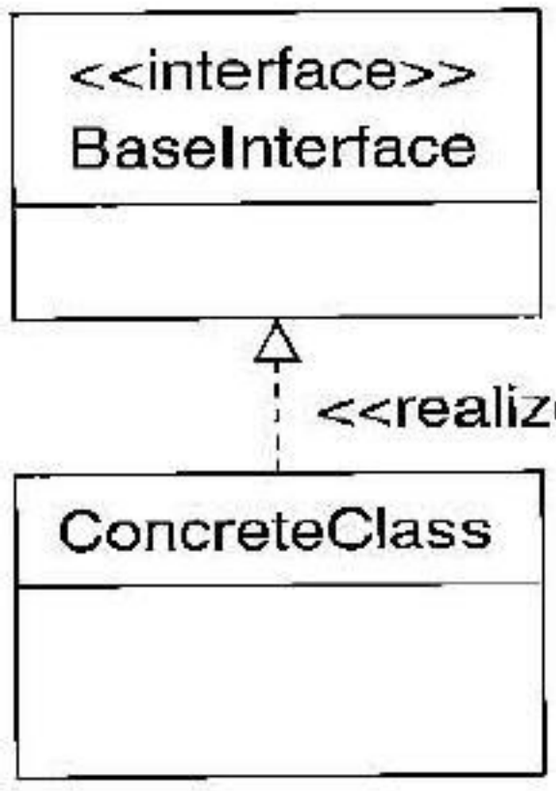
- Coding Example: Inheritance between classes




```
class SubClass extends BaseClass
{
    ...
}
```

Construction Practices -Coding

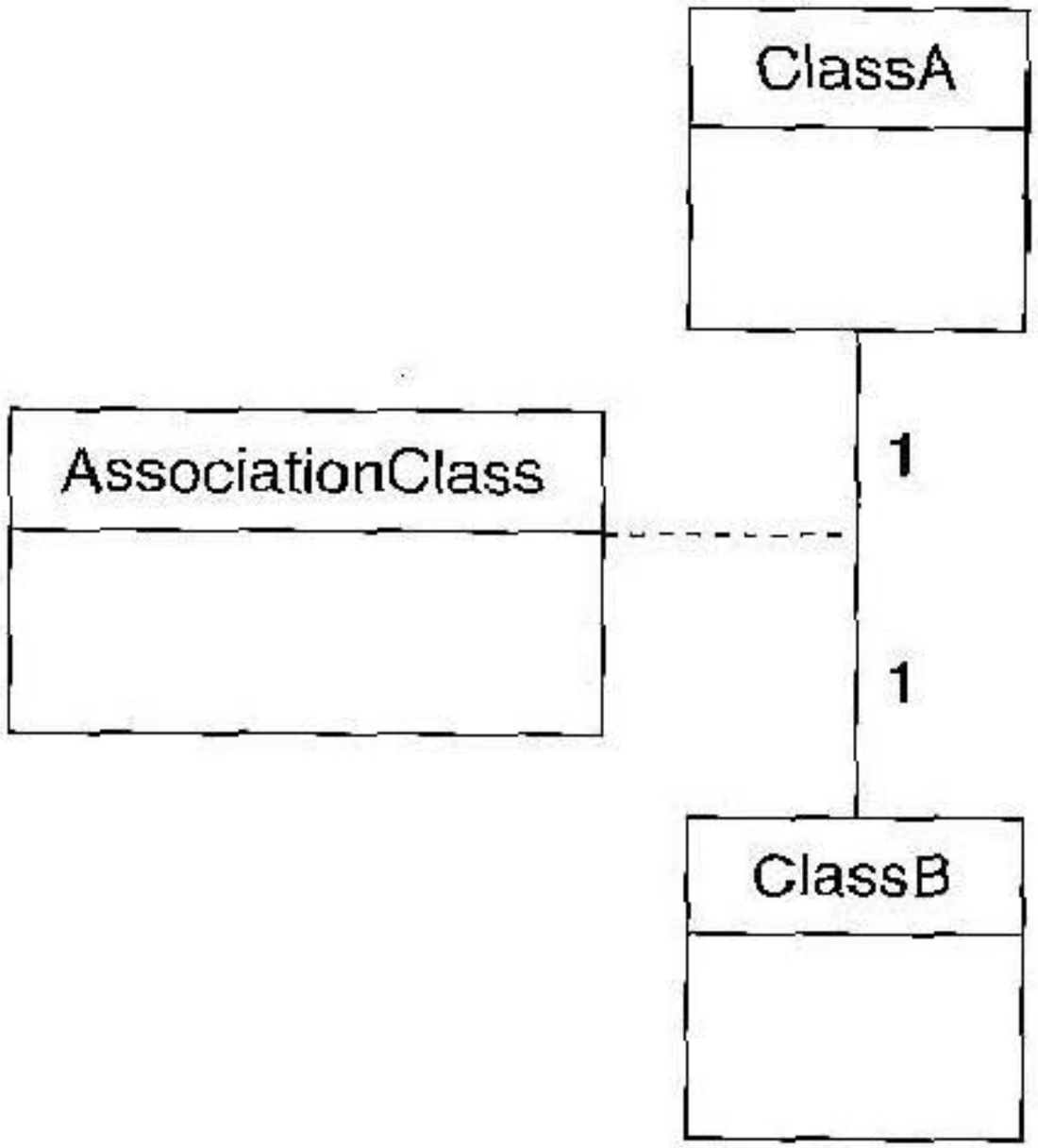
- Coding Example: Inheritance between an interface and a class




```
interface BaseInterface {
    // declaration of methods
    ...
}

class ConcreteClass implements
BaseInterface
{
    // implementation of
    // methods of the
    // interface BaseInterface
    ...
}
```

Construction Practices -Coding

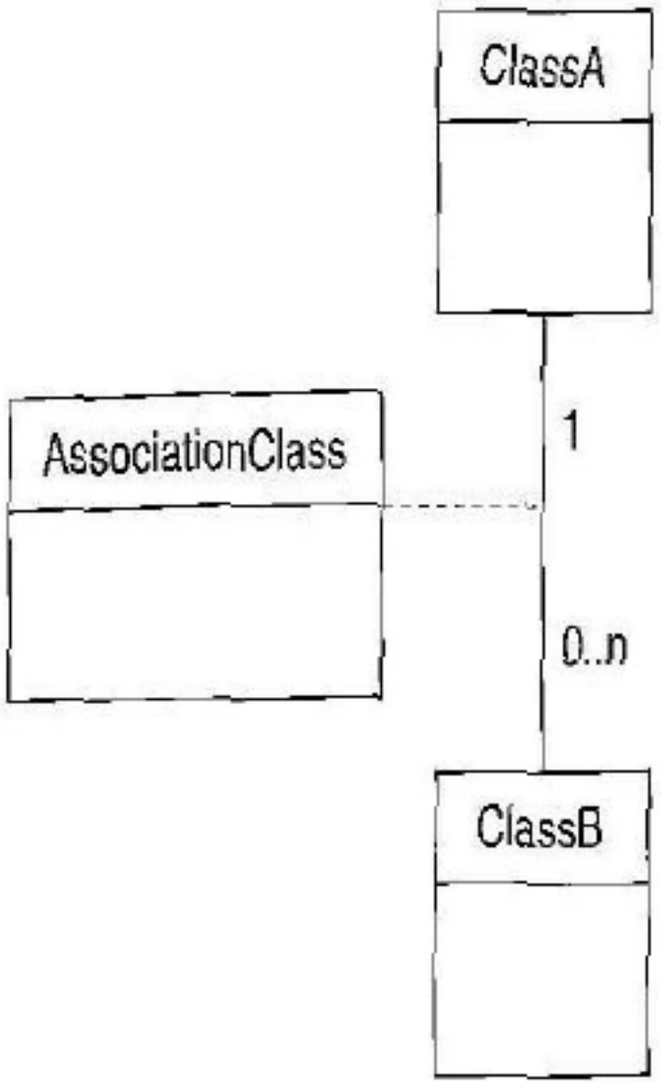
- Coding Example: One-to-one association between classes




```
class ClassA {
    ClassB _b;
    // declare attributes
    // for the
    // association class
    ...
}
class ClassB {
    ClassA _a;
    ...
}
```

Construction Practices -Coding

- Coding Example: One-to-many association between classes

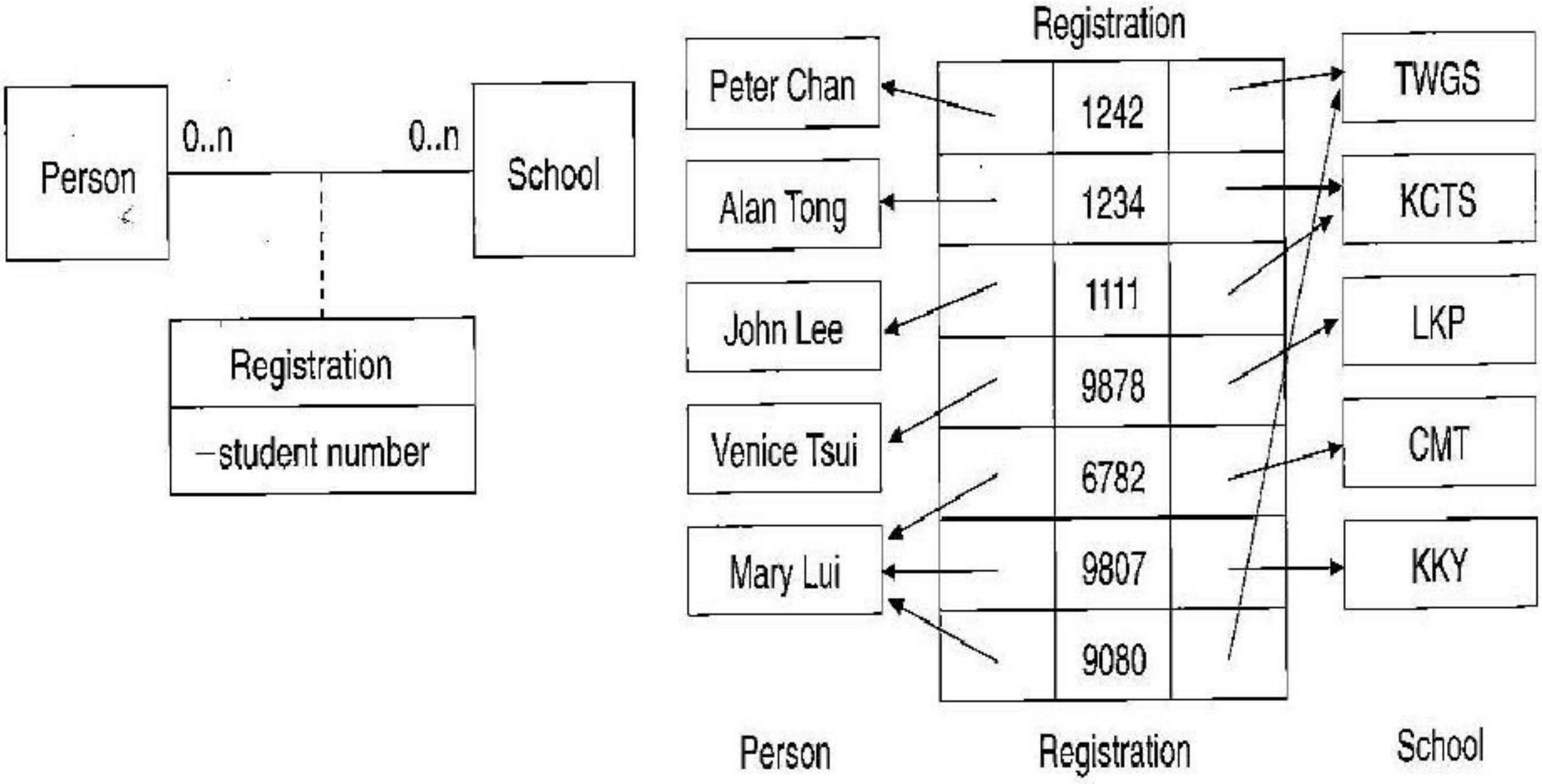



```
class ClassA {
    Vector _b;
    ...
}
class ClassB {
    ClassA _a;
    // declare attributes
    // for association
    // class
}
```

```
import java.util.Vector;
class ClassA {
    Vector _Bs;
    public ClassA() {
        _Bs = new Vector();
        ...
    }
    public Enumeration getBs() {
        return (_Bs.elements() );
    }
    // link a ClassB object to this object
    public void addB(ClassB b) {
        _Bs.add(b);
    }
    // remove the link between ClassB object to this
    // object
    public void removeB(ClassB b) {
        _Bs.remove(b);
    }
    // other functions for searching objects in the
    // vector
    ...
}   // ClassA
```

Construction Practices -Coding

- Coding Example: Many-to-many association between classes



Construction Practices -Coding

- Coding Example: Many-to-many association between classes

```
Class Registration {
    private Person _student;
    private School _school;
    private int _studentNo;

    private Registration(Person student, School school, int studentNo) {
        _school = school;
        _student = student;
        _studentNo = studentNo;
    }

    static public void register(Person student, School school, int studentNo) {
        Registration reg = new Registration(student, school, studentNo);
        school.addRegistration(reg);
        student.addRegistration(reg);
    }

    public void deregister() {
        this._school.removeRegistration(this);
        this._student.removeRegistration(this);
    }

    public School getSchool() {
        return(_school);
    }

    public Person getStudent() {
        return(_student);
    }
} // Registration
```

Construction Practices -Coding

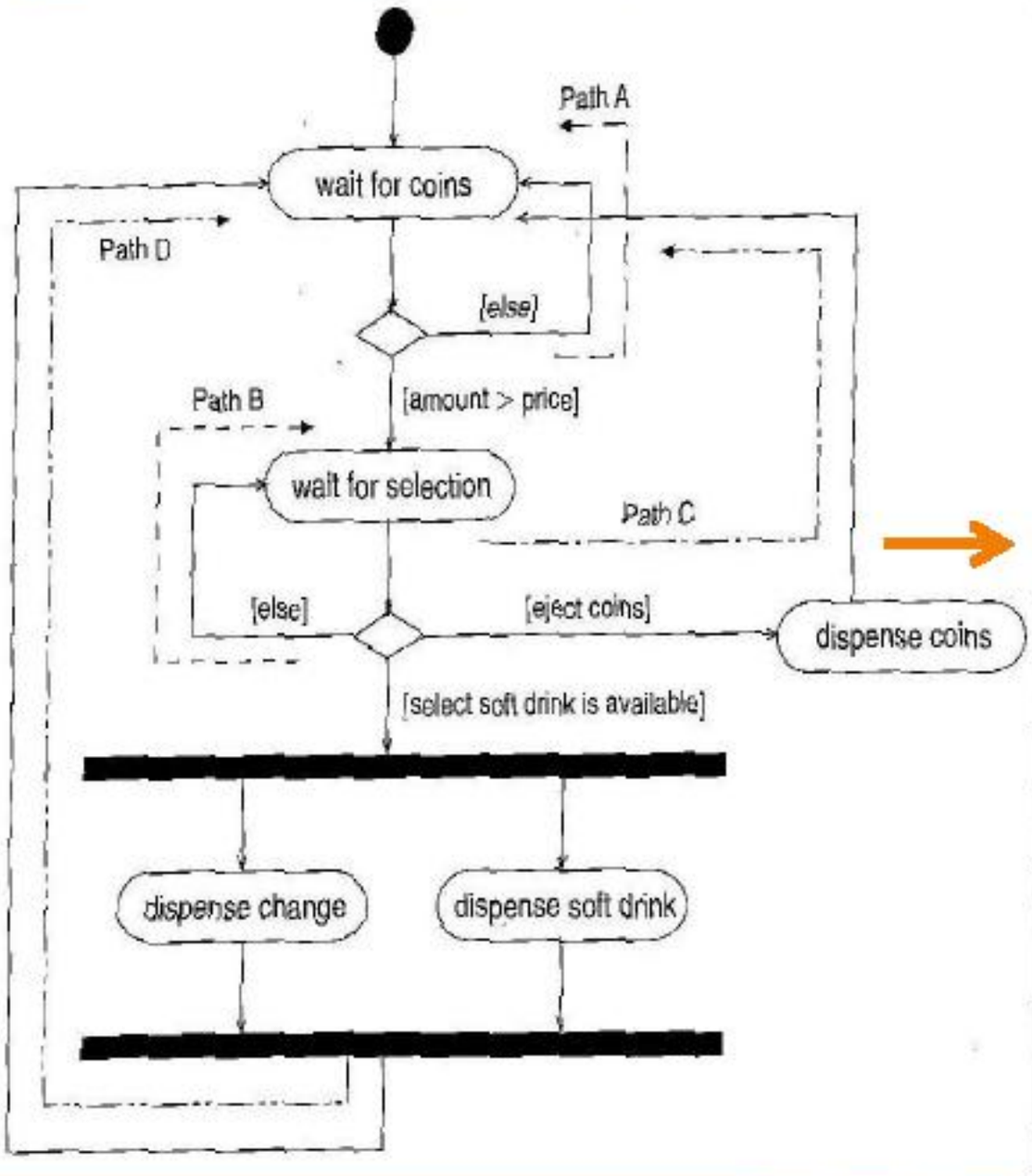
Coding Example: Many-to-many association between classes

```
class School {
    private String _name;
    private Vector _registrations;

    public School(String name) {
        _name = name;
        _registrations = new Vector();
    }

    public void setName(String name) {
        _name = name;
    }

    public String getName() {
        return (_name);
    }

    public void addRegistration(Registration reg) {
        _registrations.add(reg);
    }

    public void removeRegistration(Registration reg) {
        _registrations.remove(reg);
    }

    public Enumeration getStudents() {
        int i;
        Vector students = new Vector();

        for (i = 0; i < _registrations.size(); i++)
            students.add (((Registration)
_registrations.elementAt(i)).getStudent());
        return (students.elements());
    }
}  // School
```

```
class Person {
    private String _name;
    private Vector _registrations;

    public Person (String name) {
        _name = name;
        _registrations = new Vector();
    }

    String getName() {
        return (_name);
    }

    void setName(String name) {
        _name = name;
    }

    public void addRegistration(Registration reg) {
        _registrations.add(reg);
    }

    public void removeRegistration(Registration reg) {
        _registrations.remove(reg);
    }

    public Enumeration getSchools() {
        int i;
        Vector schools = new Vector();

        for (i = 0; i < _registrations.size(); i++)
            schools.add (((Registration)
_registrations.elementAt(i)).getSchool());
        return (schools.elements());
    }
}  // Person
```

Construction Practices -Coding

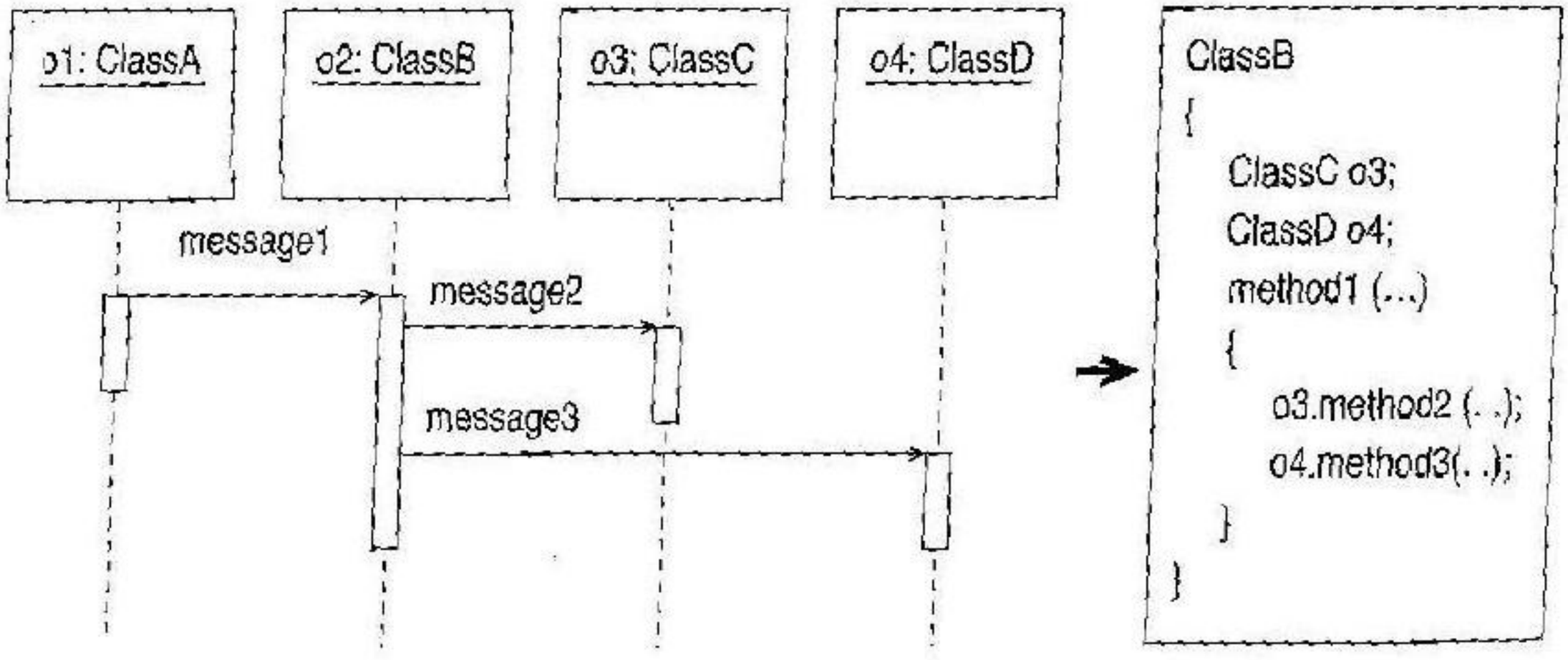
Coding Example: Activity diagram




```
while (true) {
    amount = 0.0;
    while (amount < price) {
        wait for a coin;
        add coin value to amount;
    }
    show all available soft drink;
    while (selection  is not done) {
        wait for selection from user;
        if selection is "eject coins" {
            dispense coins;
            set selection to "done";
        }
        else if selection is a valid soft drink {
            dispense change and dispense soft drink concurrently;
            set selection to "done";
        }
    }
}
```

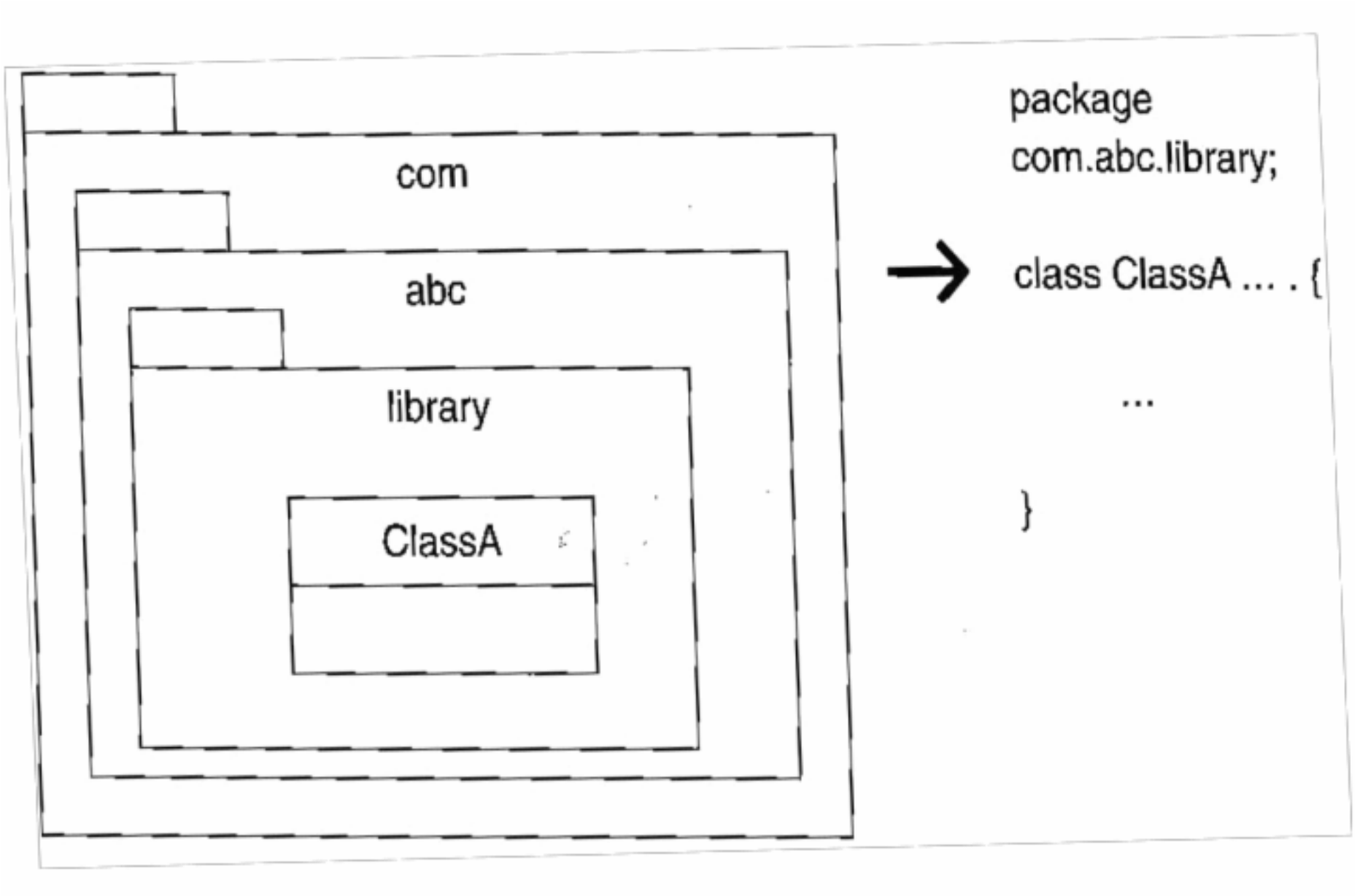
Construction Practices -Coding

- Coding Example: Sequence diagram



Construction Practices -Coding

- Coding Example: Package diagram



Construction Practices -Activities

3.1 Construction Practices -Process & Activities

3.1.1 Coding & Unit Testing

~~3.1.2 Integration & Integration Test~~

~~3.1.3 Configuration Management~~

Construction Practices – Unit Testing

3.3.1 **Software Construction Activities –** Unit Testing

- Unit
  - A program *unit is one or more contiguous program* statements, with a name that other parts of the software use to call it.
  - It is the smallest part of the program that can be tested
  - Units are called *functions* in C and C++, *procedures* in Ada and Pascal, *methods* in Java, and *routines* in Fortran.

Construction Practices – Unit Testing

3.3.1 **Software Construction Activities –** Unit Testing

- Module
  - A module may refer to a unit or a collection of related units that are assembled in a file, package, or class.
  - This corresponds to a file in C, a package in Ada, and a class in C++ and Java
  - Often the term “units” and “modules” can be used interchangeably, i.e. meaning the same thing!

Construction Practices – Unit Testing

3.3.1 **Software Construction Activities –** Unit Testing

- Unit Testing
  - Testing of an individual program unit/module, e.g. routines, methods, classes in isolation
  - Testing at the lowest level of testing
  - Testing at or closest to the program code level
  - Object oriented unit testing:
    - Testing individual method
    - Testing a group of collaborating methods within a class
    - Testing the class

Construction Practices – Unit Testing

3.3.1 **Software Construction Activities –** Unit Testing

- How to do unit testing?
  - DIY - *Test Harness Programs, Test Drivers*
    - Invoke the UUT (unit under test), e.g. method, class
    - Pass the parameters (test data) to the UUT
    - Observe the output from and behaviors of the UUT
  - Tools - *Unit Testing Frameworks*
    - JUnit: Java
    - NUnit: .NET
    - CppUnit: C++
    - PHPUnit : PHP

Construction Practices – Unit Testing

3.3.1 **Software Construction Activities –** Unit Testing

Example – Test Harness for BankAccount Class & Methods

ch03/account/BankAccountTester.java

```
1  /**
2     A class to test the BankAccount class.
3  */
4  public class BankAccountTester
5  {
6     /**
7        Tests the methods of the BankAccount class.
8        @param args not used
9     */
10    public static void main(String[] args)
11    {
12       BankAccount harrysChecking = new BankAccount();
13       harrysChecking.deposit(2000);
14       harrysChecking.withdraw(500);
15       System.out.println(harrysChecking.getBalance());
16       System.out.println("Expected: 1500");
17    }
18 }
```

Program Run

```
1500
Expected: 1500
```

Construction Practices – Unit Testing

3.3.1 **Software Construction Activities –** Unit Testing

Example – BankAccount Class and Methods for Unit Test

ch03/account/BankAccount.java

```
1  /**
2     A bank account has a balance that can be changed by
3     deposits and withdrawals.
4  */
5  public class BankAccount
6  {
7     private double balance;
8
9     /**
10       Constructs a bank account with a zero balance.
11    */
12    public BankAccount()
13    {
14       balance = 0;
15    }
16
17    /**
18       Constructs a bank account with a given balance.
19       @param initialBalance the initial balance
20    */
21    public BankAccount(double initialBalance)
22    {
23       balance = initialBalance;
24    }
25
26    /**
27       Deposits money into the bank account.
28       @param amount the amount to deposit
29    */
30    public void deposit(double amount)
31    {
32       balance = balance + amount;
33    }
34
35    /**
36       Withdraws money from the bank account.
37       @param amount the amount to withdraw
38    */
39    public void withdraw(double amount)
40    {
41       balance = balance - amount;
42    }
43
44    /**
45       Gets the current balance of the bank account.
46       @return the current balance
47    */
48    public double getBalance()
49    {
50       return balance;
51    }
52 }
```

3.1 Software Construction Activities

~~3.1.1 Coding & Unit Testing~~

3.1.2 Integration & Integration Test

~~3.1.3 Configuration Management~~

3.1.2 Software Construction Activities – Integration

Software Integration

- What is Software Integration?
- Benefits of (Good) Integration
- Integration Approaches

3.1.2 Software Construction Activities – Integration

Software Programming

- Coding (Programming)

  Design Specs (Document) → Software Code (Program)
  Design Elements → Program/Software Modules

- Integration

  Program/Software Modules → Integrated Software/System

3.1.2 Software Construction Activities - Coding

Software Programming & Integration

- Coding Activities
  - Based on SDD*, translating the design specification into the program ***source code***
    - Based on HLD*, implementing the program ***overall structure and interfaces***
    - Based on LLD*, constructing the program ***modules*** which correspond to the specified ***design elements***
  - Compiling, debugging and ***unit testing*** the coded modules
- Integration Activities
  - Linking all the ***unit-tested*** modules into a ***single*** integrated system/subsystem
  - Compiling, debugging and ***integration testing*** the integrated system/subsystem

Software Integration - Benefits

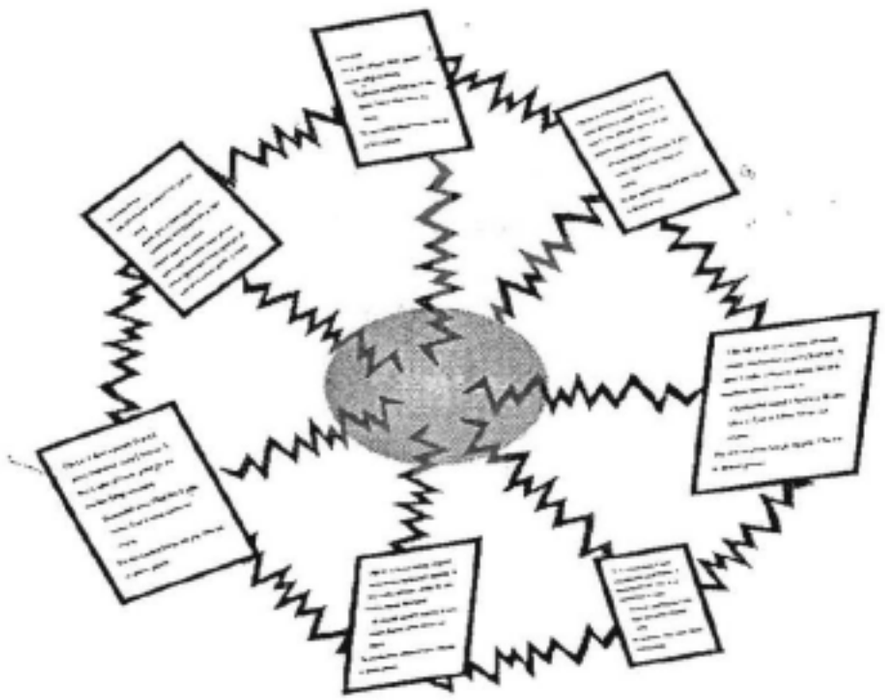
- **Benefits of (Good) Integration**

Software Integration - Benefits

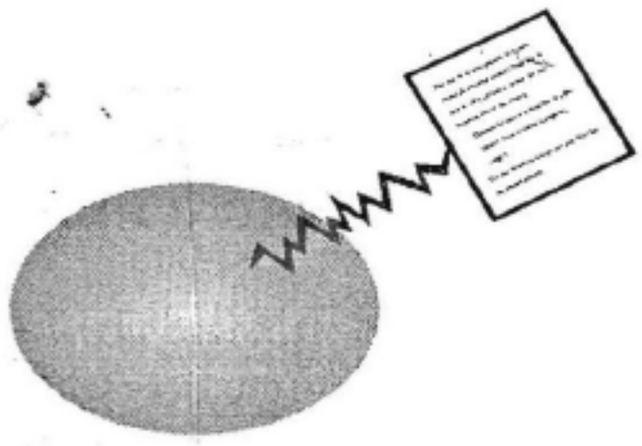
- **Benefits of (Good) Integration**
  - Easier defect diagnosis
  - Fewer defects
  - Less scaffolding
  - Shorter time to first working product
  - Better customer relations
  - Improved morale
  - Improved chance of project completion
  - More reliable schedule estimates
  - More accurate status reporting
  - Improved code quality
  - Less documentation

Software Integration - Approaches

- **Integration Approaches**
  - Phased (Big Bang) Integration
  - Incremental Integration



Phased Integration



Incremental Integration

Software Integration - Approaches

- **Phased (Big Bang) Integration - Approach**

  Step 1: Unit Development

  - Design, code, test and debug each unit/module/class

  Step 2: System Integration

  - Combine all the units into a single overall system

  Step 3: System Test

  - Test and debug the whole system

  Problems with this approach:

  - All problems surface at once
  - Hard to isolate/locate the causes of the problems

Software Integration - Approaches

Incremental Integration - Approach

Step 1: Unit Development

- Develop a small functional part of the system
- Design, code, test and debug a class

Step 2: Skeleton Integration

- Integrate the new unit with the skeleton.
- Test and debug the combination of skeleton and the new unit. Make sure the combination works before adding any new units.

Step 3: System Integration

- Repeat the above process until all units are integrated.

Software Integration - Approaches

- **Incremental Integration - Benefits**
  - Errors are easy to locate
  - The system succeeds early in the project
  - Progress monitoring is improved
  - Customer relation is improved
  - The units of the system are tested more fully
  - The system can be built with a shorter time

Software Integration - Approaches

- **Incremental Integration - Structure**
  - Top level contains:
    - Uppermost modules in the system hierarchy
    - Application/system control logic/loop
    - Top level classes, business-object classes
    - Main window, main user interface menus
    - Object containing *main()*, *WinMain()*
  - Bottom level contains:
    - System interfaces, device interfaces
    - Utility classes

Software Integration - Approaches

Incremental Integration – Strategies

Different ways/order of integrating components into a complete system:

- Top-Down Integration
- Bottom-Up Integration
- Sandwich Integration
- Risk-Oriented Integration
- Feature Oriented Integration
- T-Shaped Integration

Software Integration - Approaches

- Top-Down Integration – Using "Stubs"

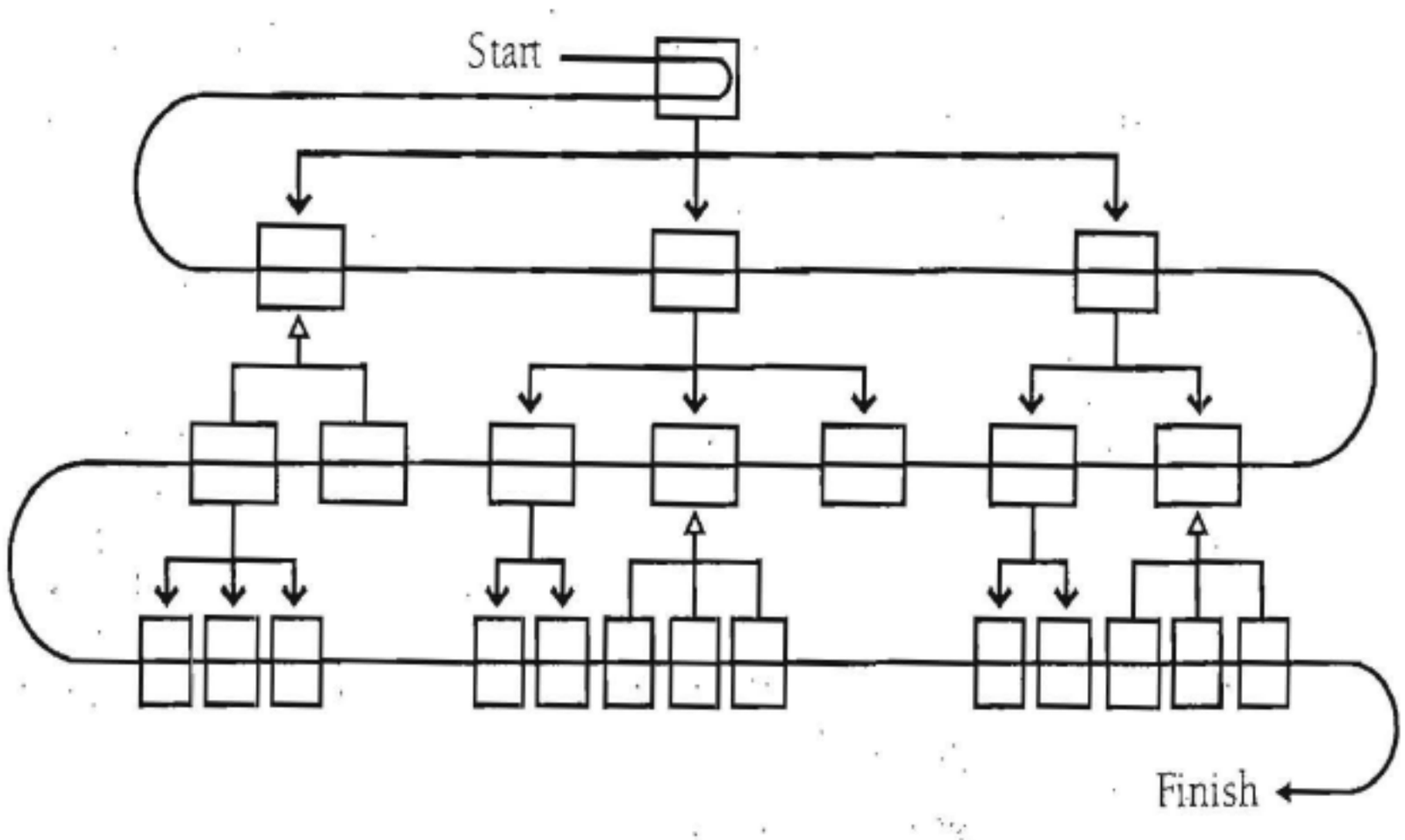


Figure 29-5 In top-down integration, you add classes at the top first, at the bottom last.

Software Integration - Approaches

- Top-Down Integration – Advantages
  - The application/system control logic, overall conceptual design issues, tested early
  - Partially working system completed early, e.g. user interface parts; hence, a morale booster
  - Coding can begin before low level design details are completed

Software Integration - Approaches

- Top-Down Integration – Disadvantages
  - Interfaces between classes must be carefully specified
  - Late discovery of system interface problems cause high-level changes
  - Large number of stubs needed introducing more errors from the new code
  - Not applicable if the collection of classes has no top

Software Integration - Approaches

- Top-Down Integration – Hybrid: Slides

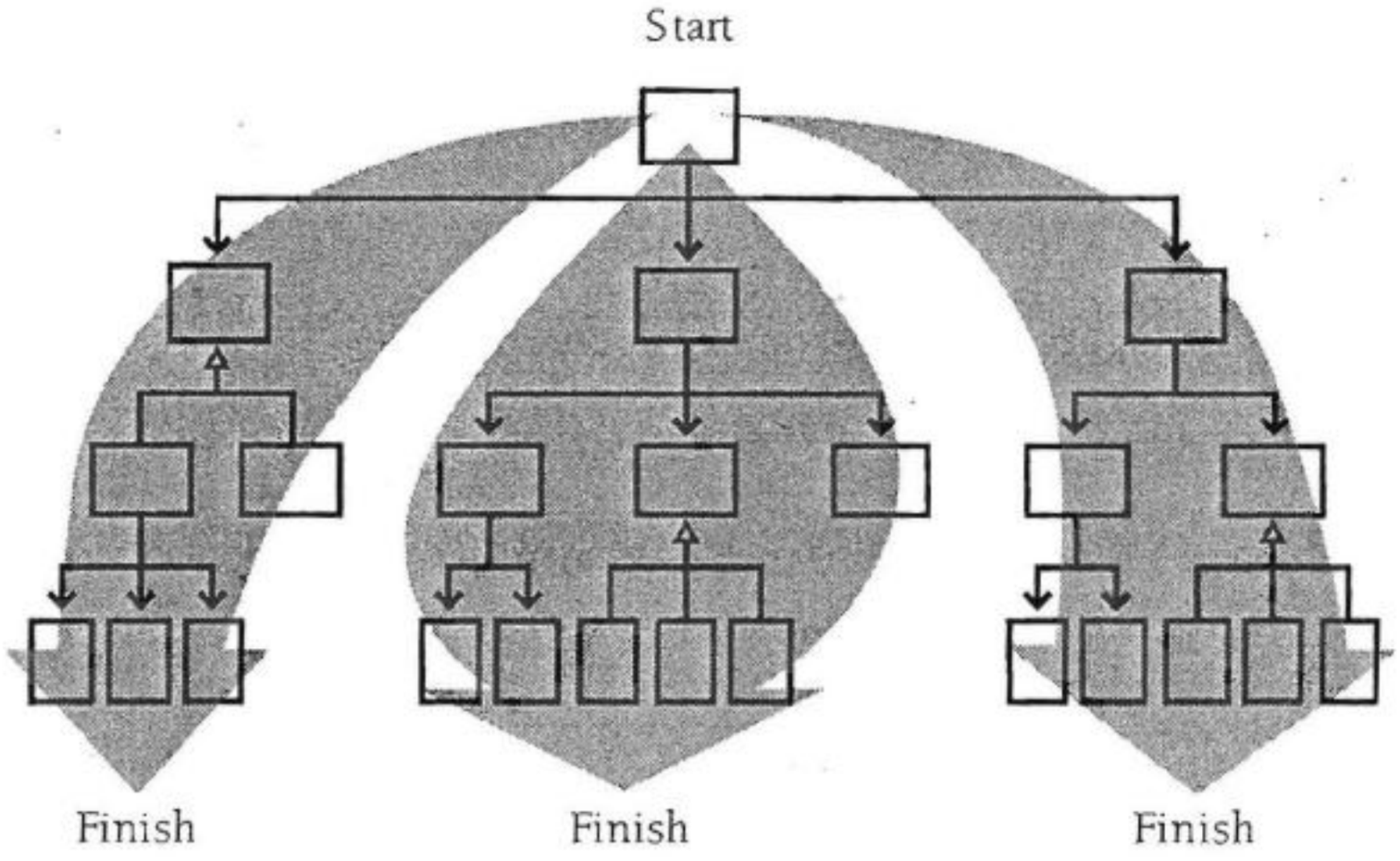


Figure 29-6 As an alternative to proceeding strictly top to bottom, you can integrate from the top down in vertical slices.

Software Integration - Approaches

- Bottom-Up Integration – Using Drivers

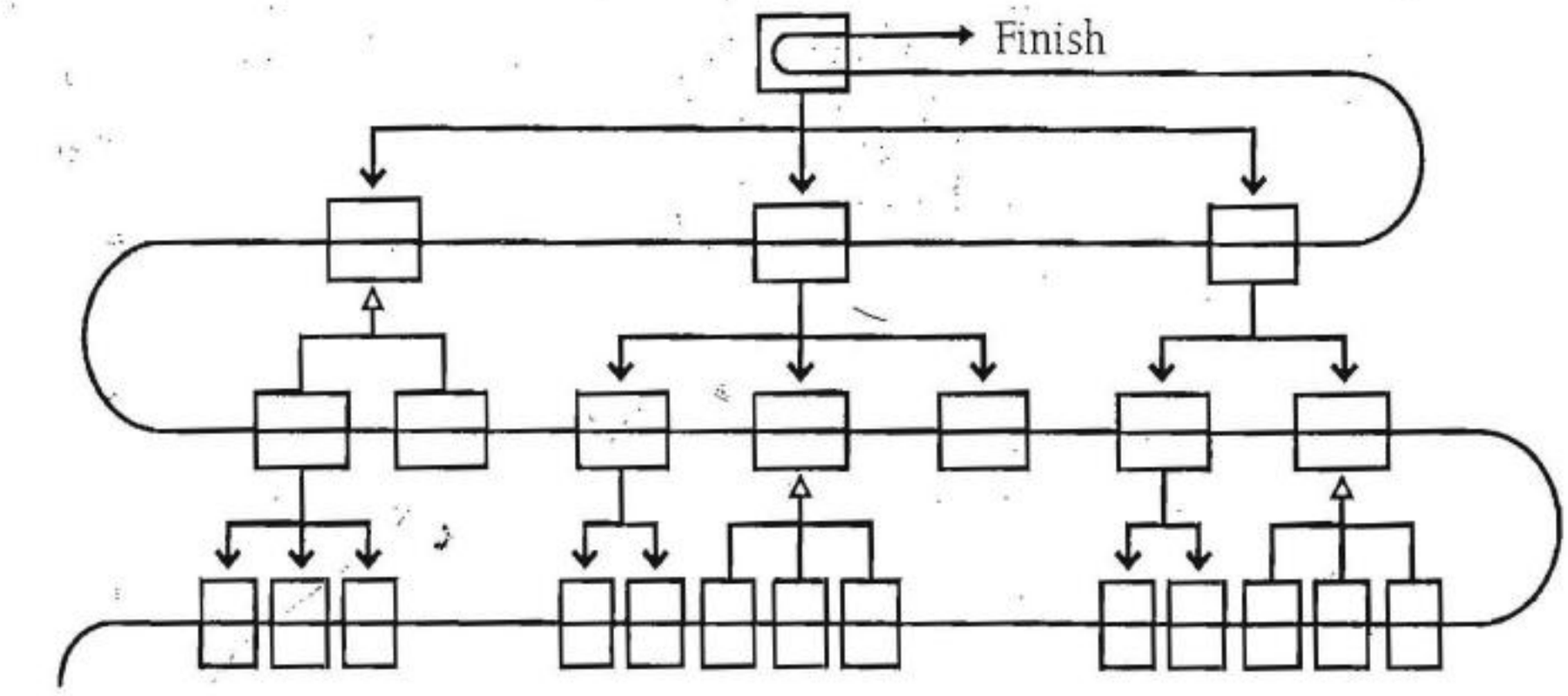


Figure 29-7 In bottom-up integration, you integrate classes at the bottom first, at the top last.

Software Integration - Approaches

- Bottom-Up Integration – Advantages
  - Possible sources of error restricted to the single class being integrated
  - Potentially troublesome system interfaces exercised early
- Bottom-Up Integration – Disadvantages
  - High-level conceptual design problems found too late that all low level works may be wasted
  - Design of the whole system must be completed before integration can begin

Software Integration - Approaches

- Bottom-Up Integration – Hybrid: Sections

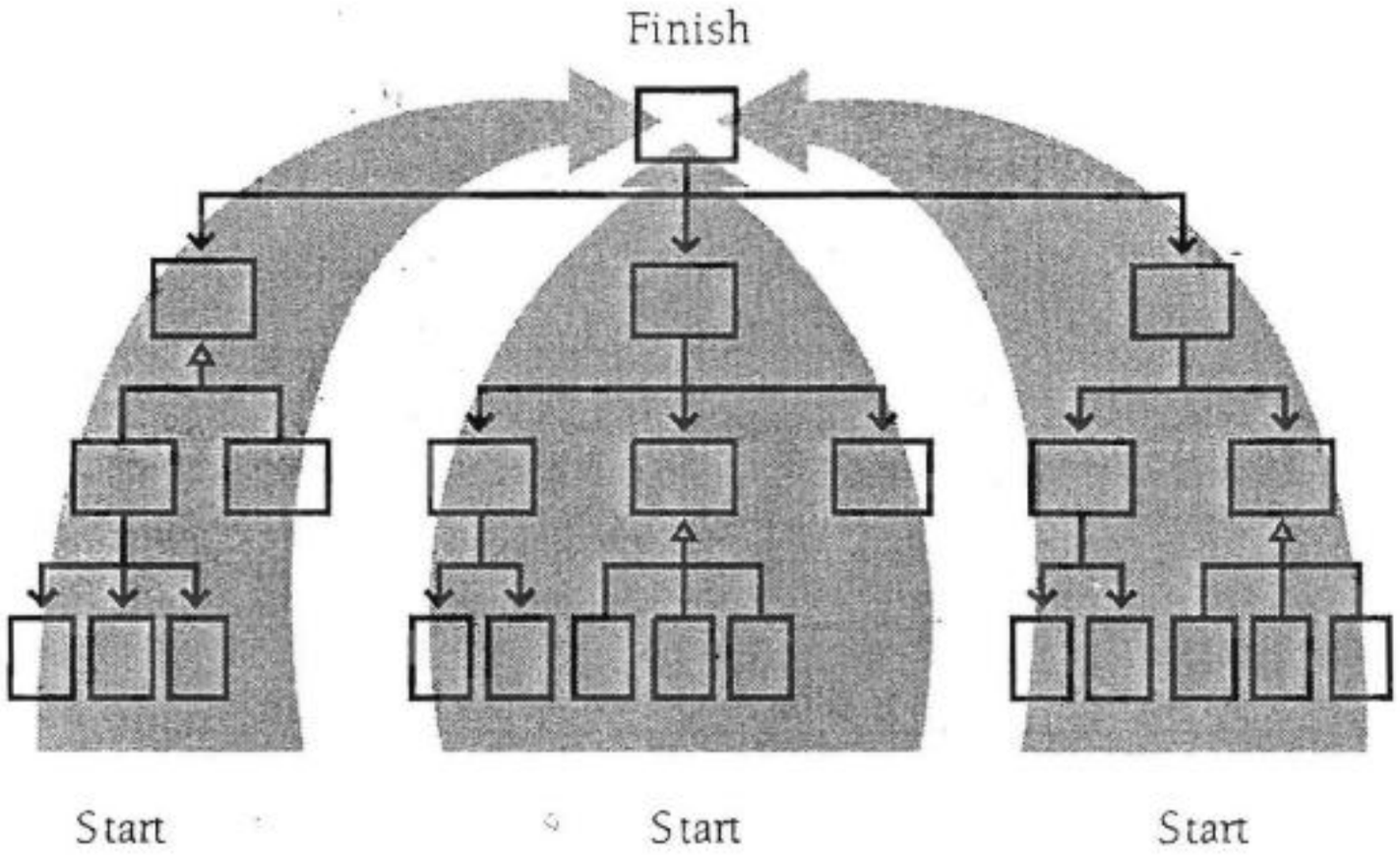


Figure 29-8 As an alternative to proceeding purely bottom to top, you can integrate from the bottom up in sections. This blurs the line between bottom-up integration and feature-oriented integration, which is described later in this chapter.

Software Integration - Approaches

- Sandwich Integration

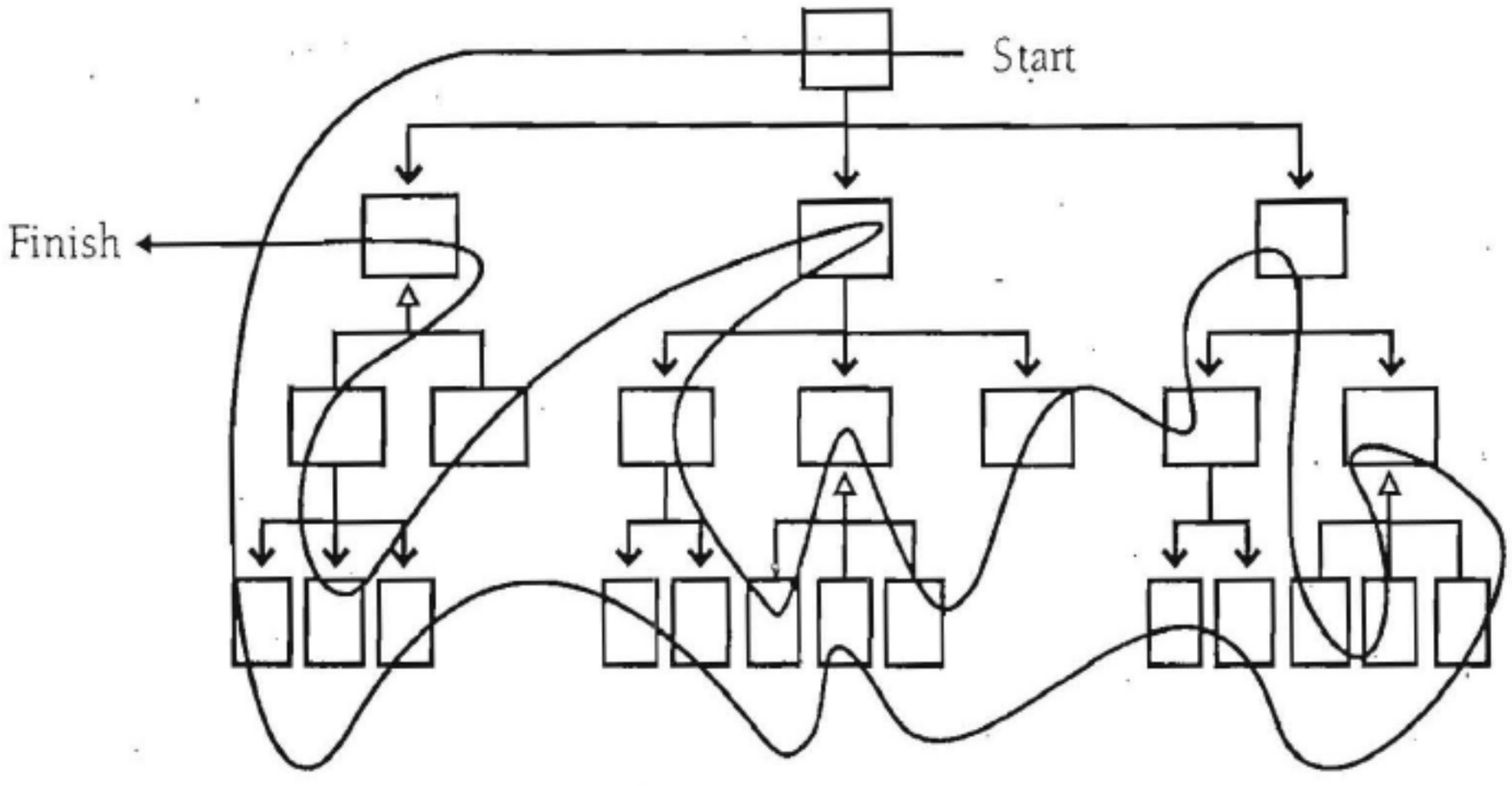


Figure 29-9 In sandwich integration, you integrate top-level and widely used bottom-level classes first and you save middle-level classes for last.

Software Integration - Approaches

- Sandwich Integration – Advantages
  - Rigidity of pure top-down and bottom-up integration is avoided
  - Amount of scaffolding is potentially minimized
  - A realistic, practical approach

Software Integration - Approaches

- Risk-Oriented Integration – *"Hard Part First"*

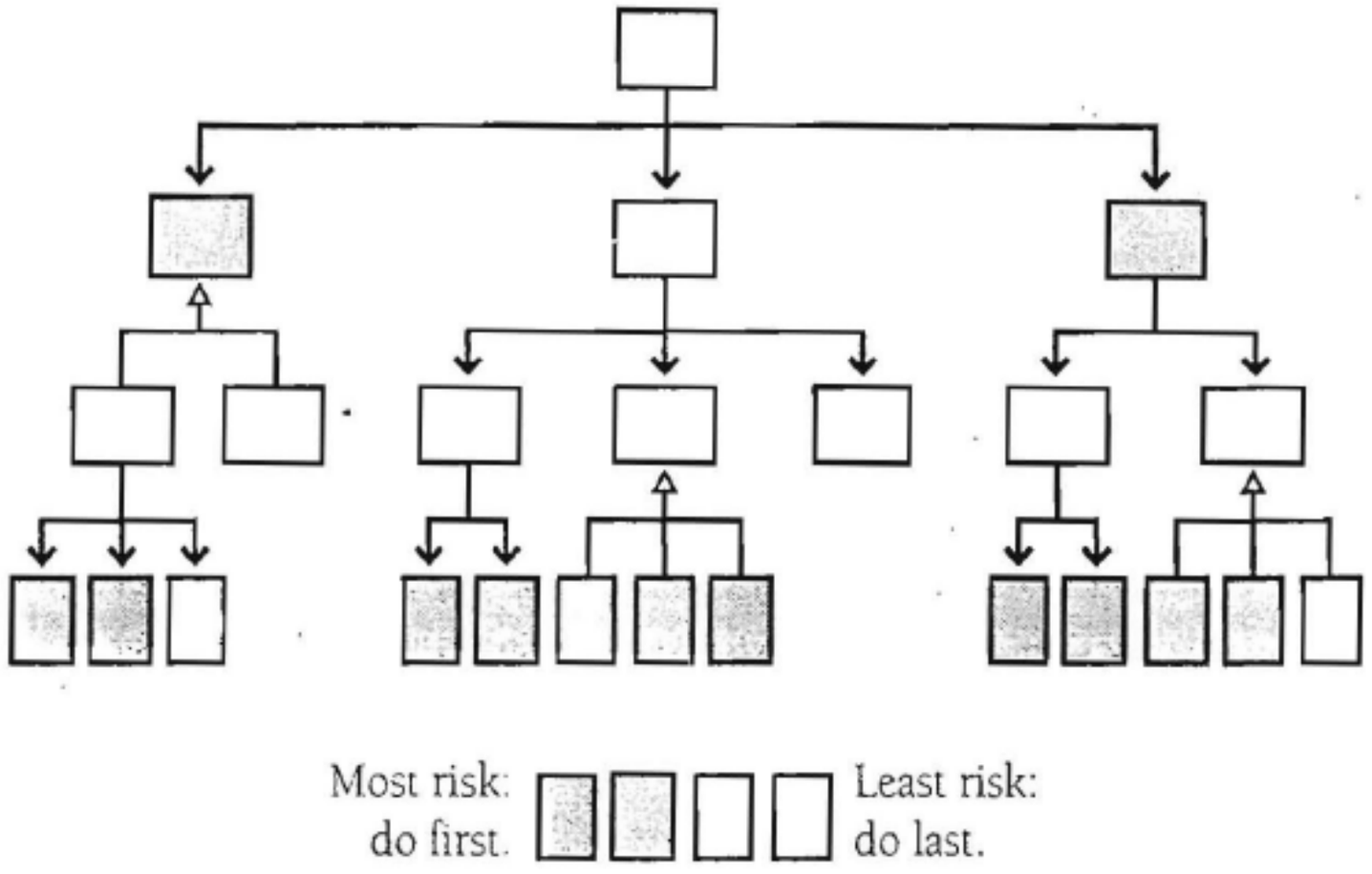


Figure 29-10 In risk-oriented integration, you integrate classes that you expect to be most troublesome first; you implement easier classes later.

Software Integration - Approaches

Feature-Oriented Integration

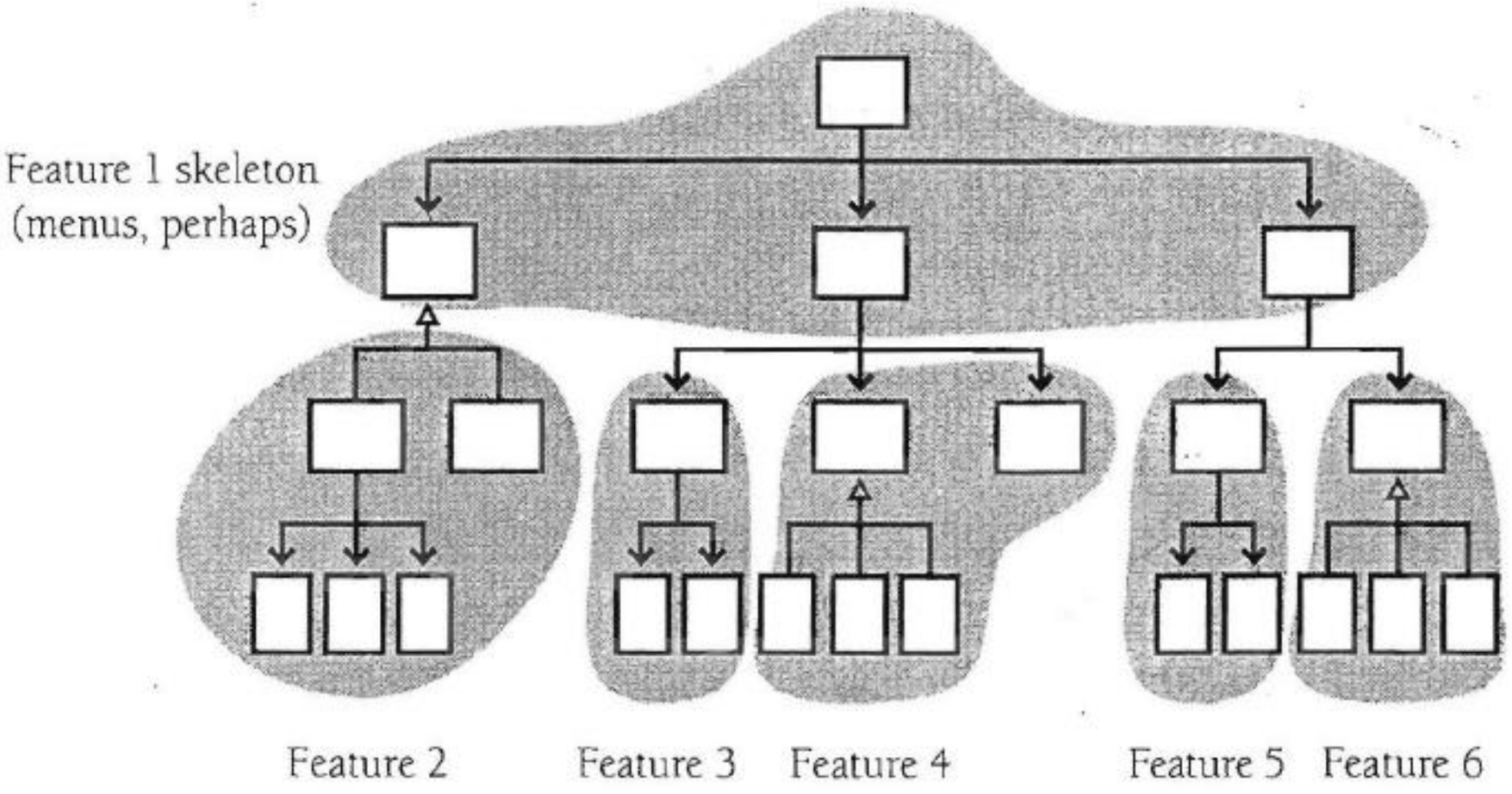


Figure 29-11 In feature-oriented integration, you integrate classes in groups that make up identifiable features—usually, but not always, multiple classes at a time.

Software Integration - Approaches

- Feature-Oriented Integration – Advantages
  - Most scaffolding eliminated
  - Functional software for early release
  - Works well with object-oriented design as objects tend to map well to features

Software Integration - Approaches

- T-Shaped Integration

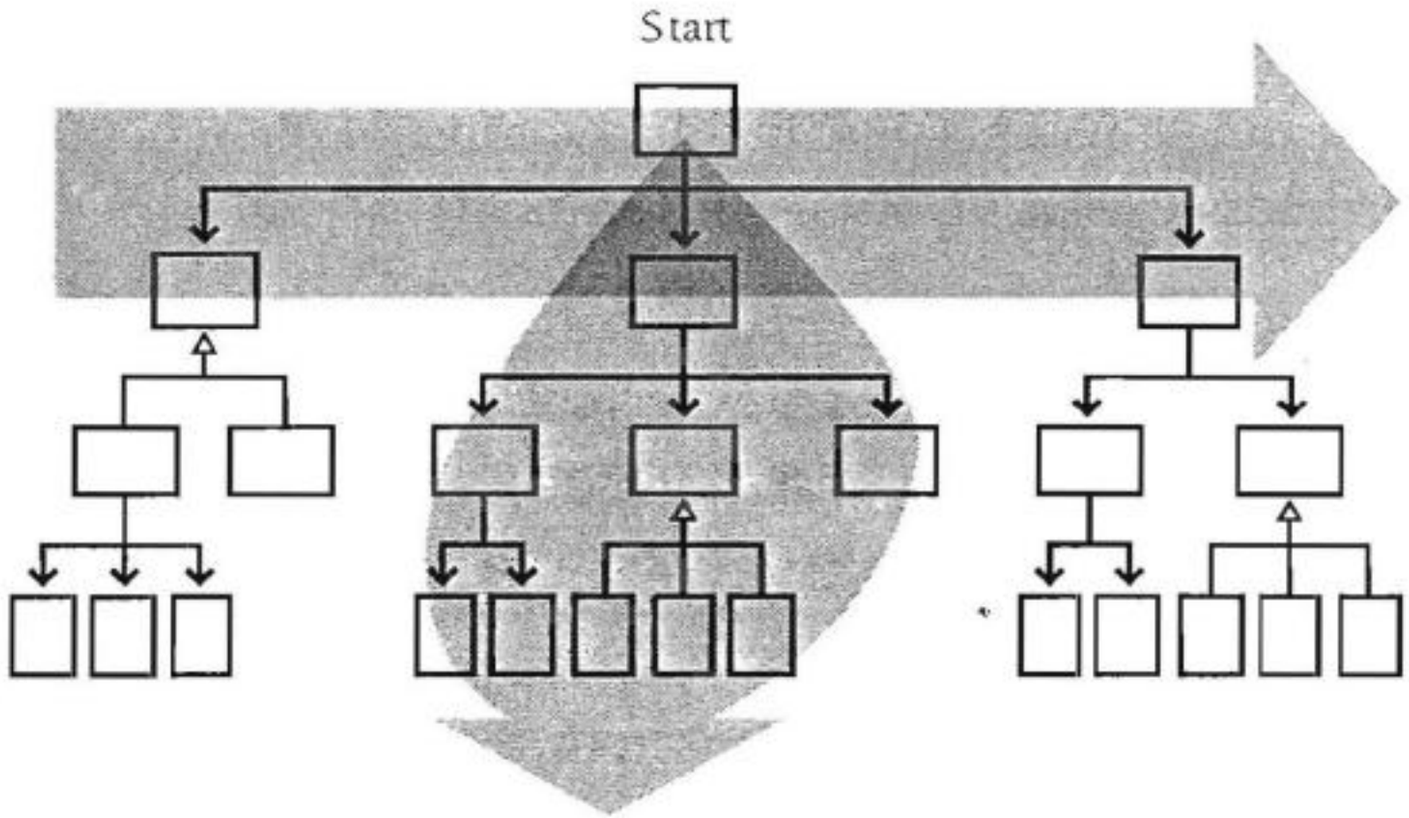


Figure 29-12 In T-shaped integration, you build and integrate a deep slice of the system to verify architectural assumptions, and then you build and integrate the breadth of the system to provide a framework for developing the remaining functionality.

Software Integration - Approaches

Object Oriented Integration

- Can be done top down or bottom up depending on the program structure
- Top-Down integration used for controlling classes
- Bottom-Up integration used for operational classes
- Typical integration sequence:
  - Objects with no messages to other objects are integrated first
  - Then objects with messages to the above objects
  - And so on until the whole system is tested

Software Integration - Approaches

Object Oriented Integration

- Intra-method testing
  - Testing of individual methods in isolation
- Inter-method testing:
  - Pairs of methods within the same class
- Intra-class testing
  - A single entire class usually by a sequence of class method calls
  - Usually as sequences of calls to methods within the class
- Inter-class testing
  - More than one class is tested at the same time

3.1 Software Construction Activities

~~3.1.1 Coding & Unit Testing~~

~~3.1.2 Integration & Integration Test~~

3.1.3 Configuration Management

3.1.3 Software Construction Activities – Configuration Management

Software Configuration Management (SCM)

- What is SCM?
- Purpose of SCM
- SCM Process & Activities
- SCM Tools and Case Study

3.1.3 Configuration Management - Definition

- What is SCM? - Definition
  - Process of identifying, tracking and storing all the artifacts on a project
  - The practice of identifying project artifacts and handling changes systematically so that a system can maintain its integrity over time
  - Each artifact referred to as a *Configuration Item (CI)*

3.1.3 Configuration Management - Terminology

- What is SCM ? - Terminology
  - Version
  - Revision
  - Variation
  - Configuration
  - Configuration Item
  - Baseline
  - Build

3.1.3 Configuration Management - Terminology

- Configuration Item (CI)
  - An artifacts produced in the course of developing a software product, e.g. code and documents, which are subject to SCM process.
- Configuration
  - A unique combination of all related CI's which makes up a specific version of a unified system/product. A software configuration maintains the information about which version of all the related CI's correspond to a particular version/release of the product.

3.1.3 Configuration Management - Terminology

- Version
  - Each unique form of a software module (artifact) existed over time during the course of software development. Usually each module has at least 2 versions: old & new.
- Revision
  - A new version of the module (artifact) intended to replace the old version of the same module
- Variation
  - Different versions of the same module (artifact) intended to co-exist for different purposes

3.1.3 Configuration Management - Terminology

- Build (Product Build)
  - To create the final (executable) form of the software system/product by integrating all the CI's as per the SCM configuration specification.
- Release
  - The final product version built from a particular CI configuration for the purpose of public distribution.

3.1.3 Configuration Management - Terminology

- Baseline
  - A set of one or more configuration items whose content and status have been reviewed technically and officially accepted at some step in the product life cycle.
- Generic types of baselines
  - Functional baseline
  - Allocated baseline
  - Developmental baseline
  - Product baseline

Software Construction – Configuration Management

Terms Clarity

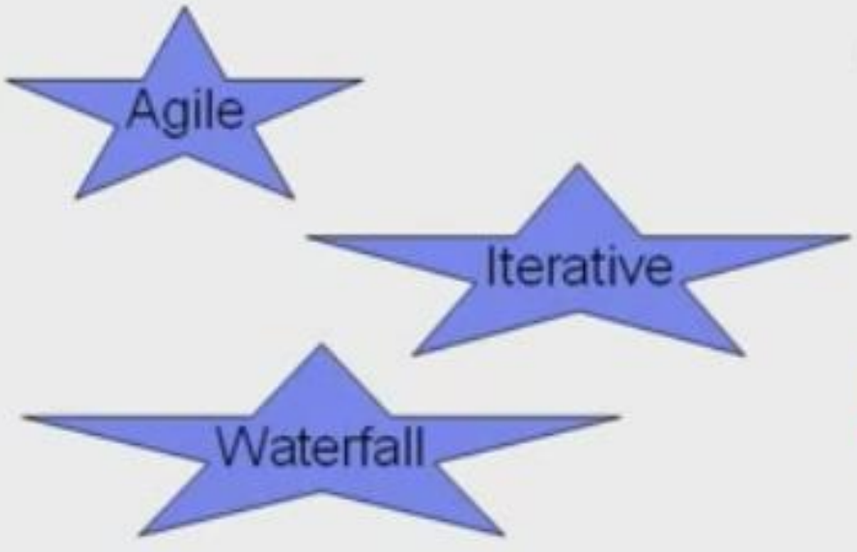
SCM – Software Configuration Management
SCCM – Software Change and Configuration Management
SCRM – Software Configuration and Release Management

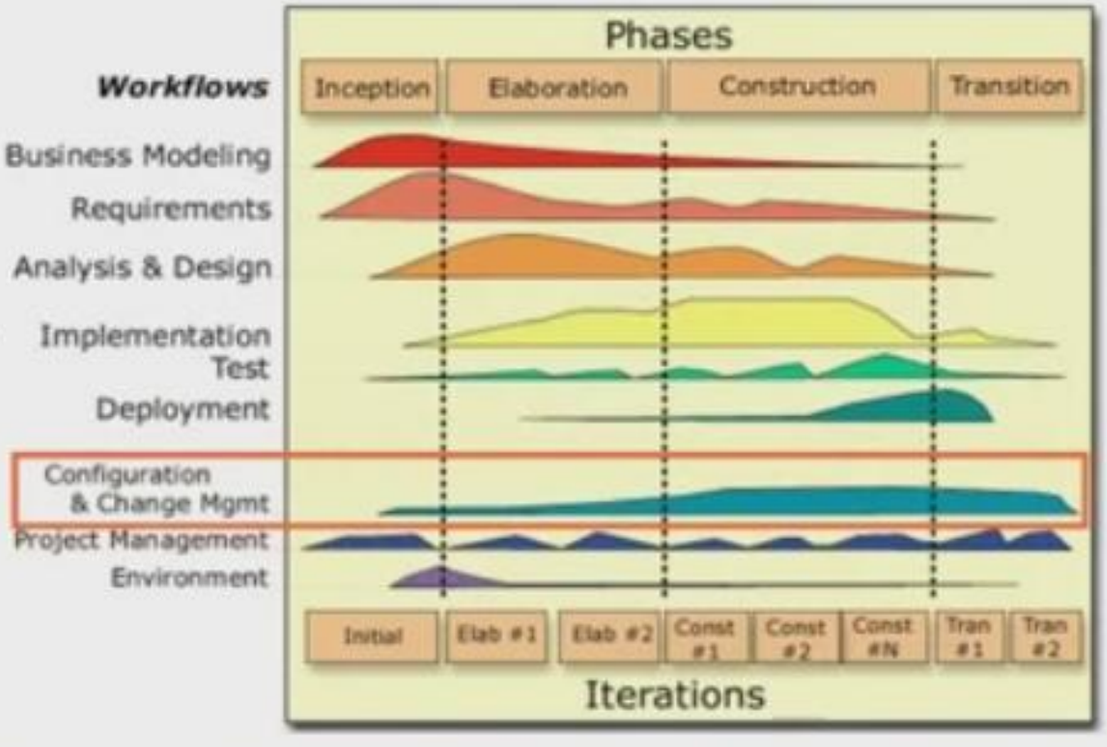
Terms are used interchangeably in the industry and is a huge topic for separate discussion

Software Construction – Configuration Management

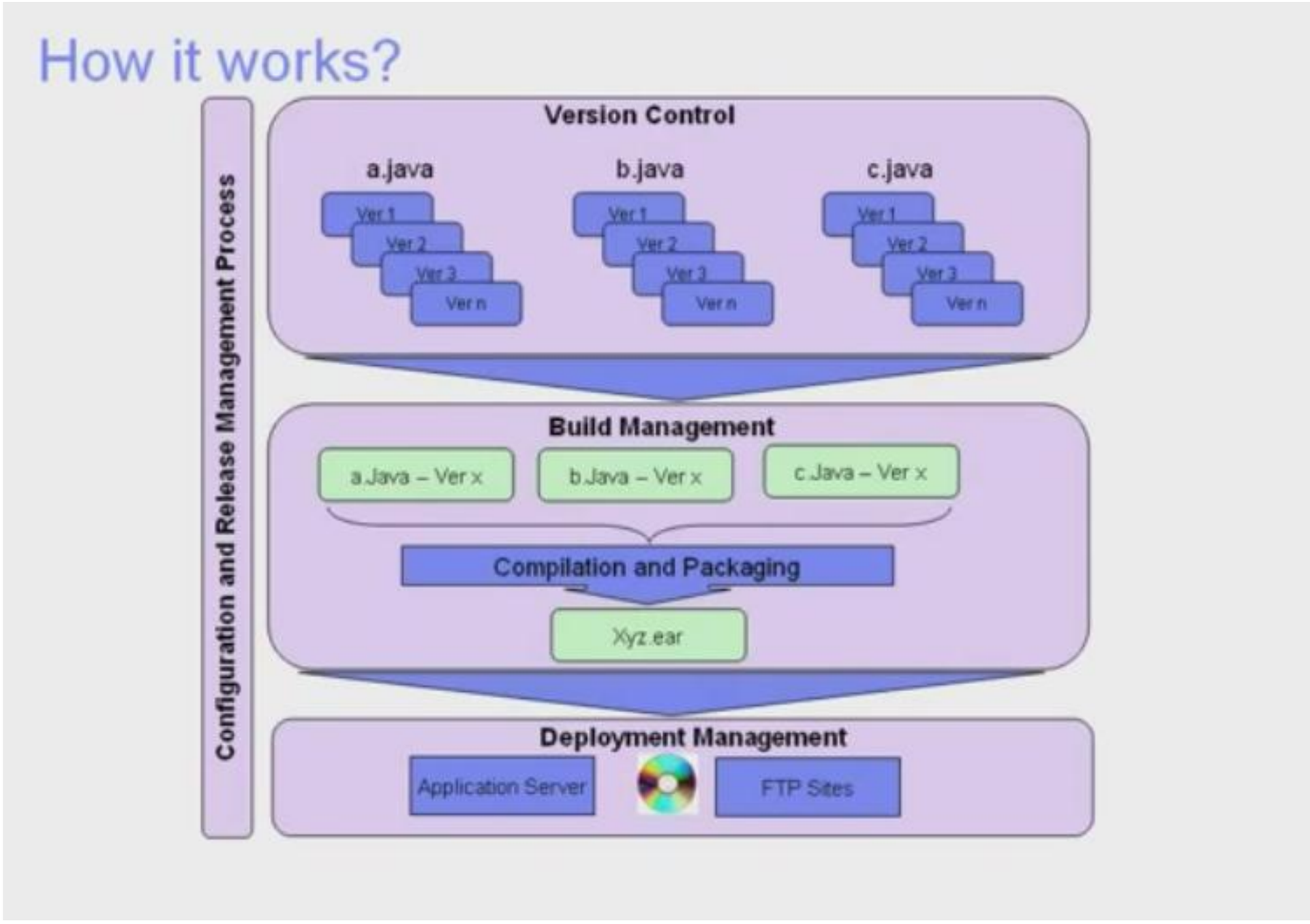
Brief on SCRM

- SCRM is the way to manage change in the development life cycle by controlling the configuration of your configurable items (items that tend to change Eg: source code, executable, documentation etc.,) with the help of some set process, tools and standards
- It is a very important aspect that cuts across all the phases of Development Lifecycle
- Its an integral part of the software development irrespective of the development models being practiced





Software Construction – Configuration Management



3.1.3 Software Construction Activities – Configuration Management

Software Configuration Management (SCM)

- ~~What is SCM?~~
- Purpose of SCM
- ~~SCM Process & Aetivities~~
- ~~SCM Tools and Case Study~~

3.1.3 Configuration Management - Purposes

- Purposes of SCM
  - Right version of code is included in the system
  - Same code can't be changed by different people at the same time
  - Different software versions can be built and distributed as and when needed
  - Changes can be traced and impacted code rolled back
  - Any changes can be group focused/coordinated
  - Different product releases can be worked in parallel

3.1.3 Configuration Management - Purposes

- SCM Goals
  - Baseline Safety
    Ensure that new or changed CI's are safely stored in a repository and can be retrieved when necessary
  - Overwrite Safety
    Ensure that engineer's changes to the same CI's are applied correctly
  - Revision
    Ensure ability to revert to earlier version
  - Disaster Recovery
    Retain backup copy in case of disaster

3.1.3 Software Construction Activities – Configuration Management

Software Configuration Management (SCM)

- ~~What is SCM?~~
- ~~Purpose of SCM~~
- SCM Process & Activities
- ~~SCM Tools and Case Study~~

Software Construction – Configuration Management

Various Activities in SCRM

- **SCRM at a broader level have below activities**
  - Version Control
    - Using a central repository with access control, managing versions of individual files to be able to control and track who, when, why and how of changes to the files
  - Change Control
    - Controlling changes (defects or new enhancements) to the software product using a work flow management system which helps track aspects like who approved what changes to what release and who has implemented the changes
  - Build Management
    - Compile and package source code into binaries that will be delivered to internal or external customers
  - Release Management
    - Plan and manage releases as per the time lines
  - Deployment Management
    - Deploy product or application to various environments
  - Process Management
    - Define, manage and implement configuration management process

3.1.3 Configuration Management - Activities

- SCM Process & Activities
  1. Configuration identification
  2. Baselines
  3. Change control
  4. Version control
  5. Configuration audits
  6. Configuration status reporting
  7. Release management and delivery

3.1.3 Configuration Management - Activities

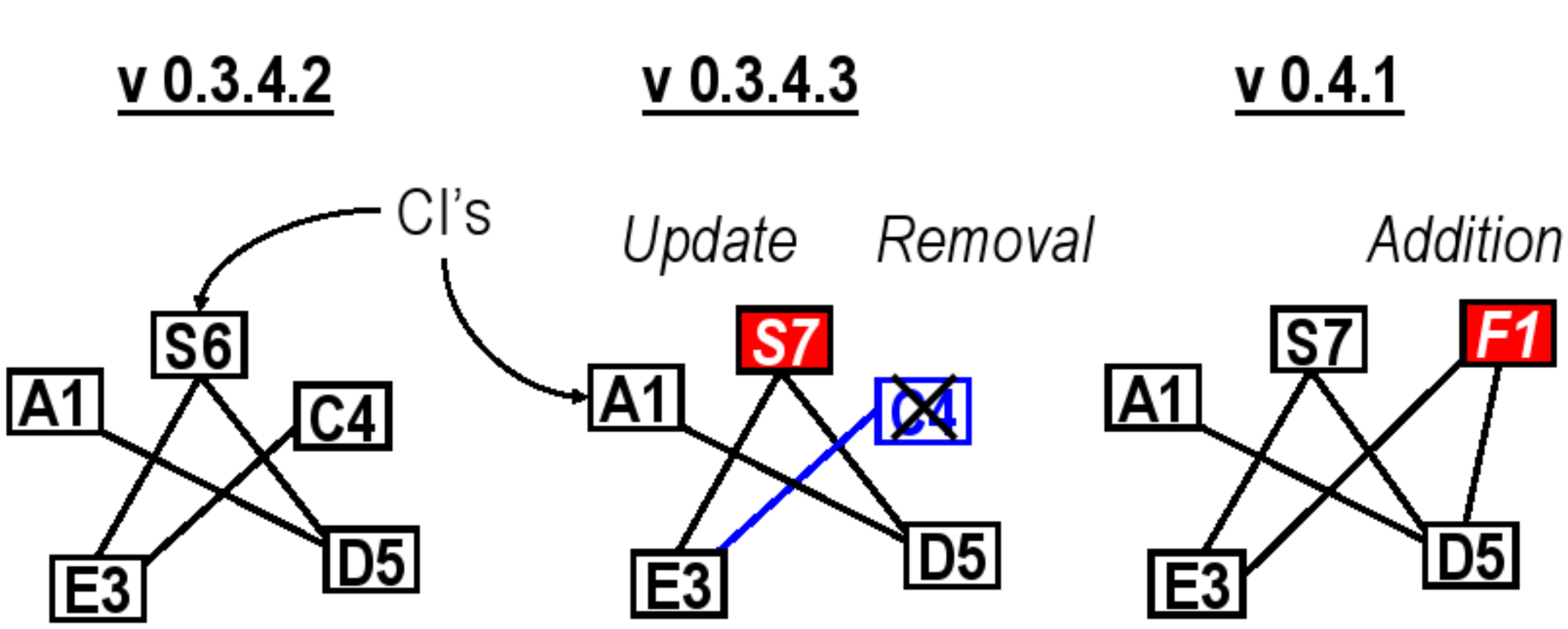
1. Configuration Identification

- First step in the SCM process
- Candidate CI's include:
  - source and object code
  - project specifications
  - user documentation
  - test plans and data
  - supporting software such as compilers, editors
  - any artifact that will undergo modification or need to be retrieved at some time after its creation

3.1.3 Configuration Management – Activities

2. Baseline

- Is an individual or group of CI's labeled at a key project milestone
- Each version below is a baseline



3.1.3 Configuration Management – Activities

3. Change Control

- Set of activities to request, evaluate, approve or disapprove, and implement changes to baselined CI's
- Steps:
  - identification and documentation
  - analysis and evaluation
  - approval or disapproval
  - verification, implementation and release
- Also called *Change Management System*

3.1.3 Configuration Management – Activities

3. Change Control – Step 1

- Identification and Documentation
  - Name of requester
  - Description and extent of the change
  - Reason for the change (e.g. defects fixed)
  - Urgency
  - Amount of time required to complete change
  - Impact on other CI's or impact on the system

3.1.3 Configuration Management – Activities

3. Change Control – Step 2

- Analysis and Evaluation
  - Reason for the change (e.g. bug fix, performance improvement, cosmetic)
  - Number of lines of code changed
  - Complexity
  - Other source files affected
  - Amount of independent testing required to validate the change

3.1.3 Configuration Management – Activities

3. Change Control – Step 3

- Approval or Disapproval

  A management decision by the *change control board* or *change management committee*

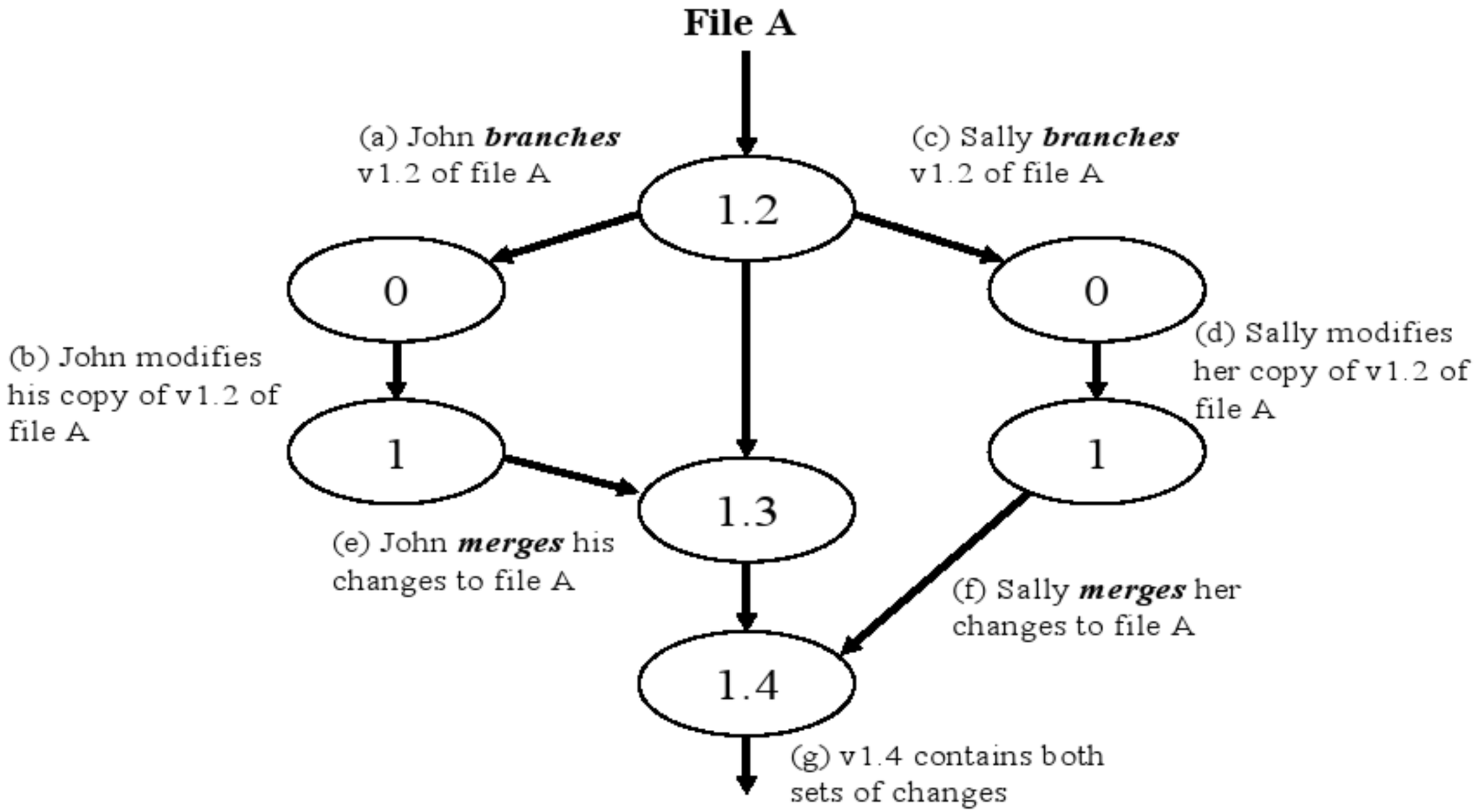
  - Establish a change control procedure
  - Establish a product baseline, versions, subversions
  - Handle change requests in groups
  - Estimate the cost of each change
  - Cost & benefit analysis based on the previous impact analysis
  - Good balance between bureaucracy and effective change control

3.1.3 Configuration Management – Activities

4. Version Control

- System that supports the management and storage of CIs as they are created and modified throughout the SDLC.
- Typical capabilities include:
  - Repository
  - Checkin/Checkout
  - Branching and merging
  - Builds
  - Version labeling

3.1.3 Configuration Management – Activities



3.1.3 Configuration Management – Activities

4. Version Control – Source Code Control

- Automatic version numbering
- Atomic operations on source files--accessed via strict check-in and check-out procedures
- File locking against concurrent access
- Version merging for multiple check-in's
- Easy updates of all files to the current versions
- Files can be backtracked to any versions
- History of all changes can be listed

3.1.3 Configuration Management – Activities

5. Configuration Audits

- Verify proper procedures are followed, such as formal technical reviews
- Verify SCM policies, such as those defined by change control, are followed
- Determine whether a software baseline is comprised of the correct configuration items
  - Are there extra items included? Are there items missing? Are the versions of individual items correct?
- Typically conducted by quality assurance group

3.1.3 Configuration Management – Activities

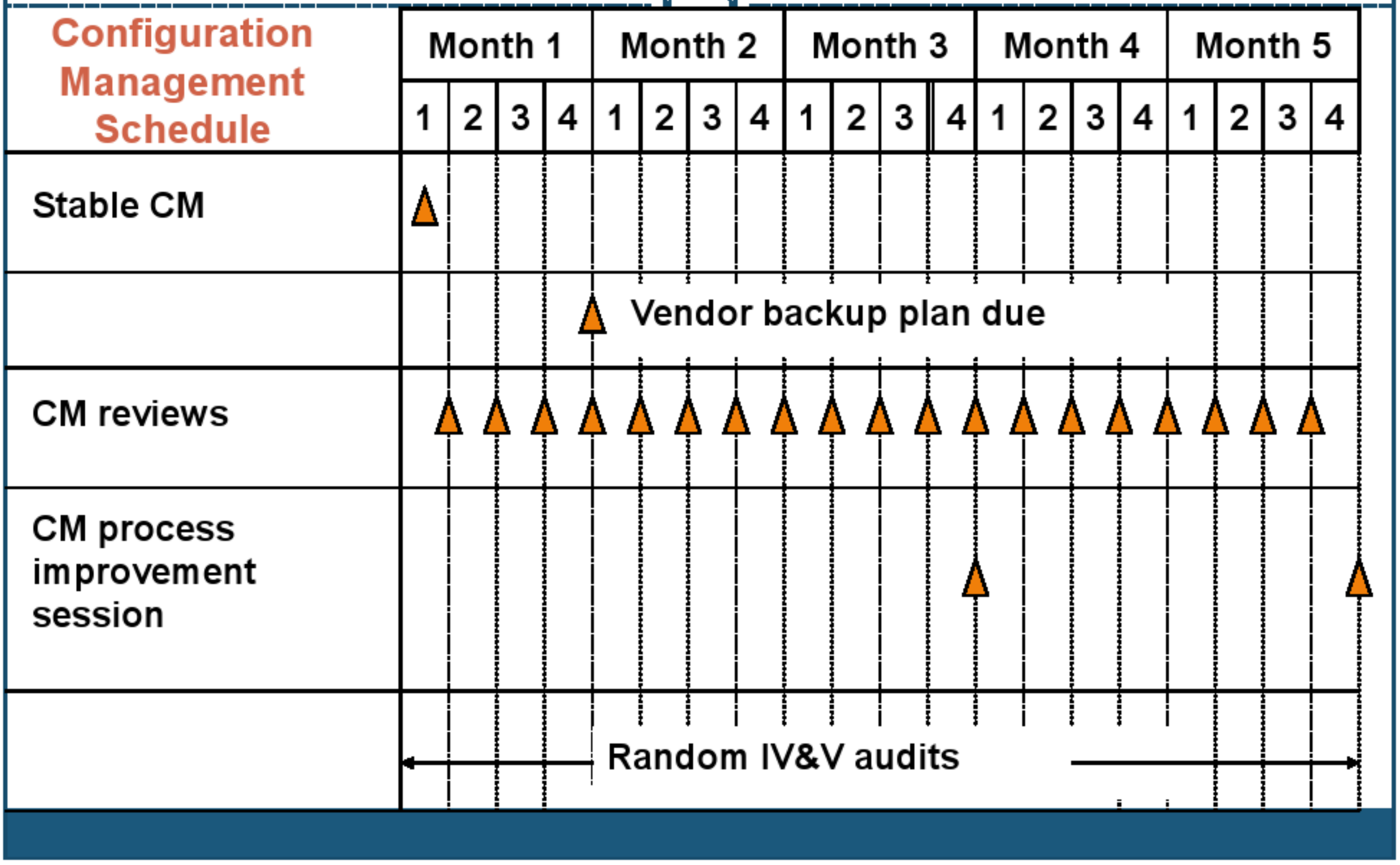
6. Configuration Status Reporting

- Extraction, arrangement and formation of configuration reports
  - Name and version of CI's
  - Approval history of changed CI's
  - Software release contents and comparison between releases
  - Number of changes per CI
  - Average time taken to change a CI

3.1.3 Configuration Management – Planning

1. Roughly sketch out your SCMP
   Determine procedures for making changes
   See the case study for an example
2. Specify what you need from a CM tool
   For class use, maybe only *locking* and *backup*
3. Evaluate affordable tools against needs and budget
   Commercial tools are in wide use
   For class use, try free document storage web sites; simple method of checking out, e.g. renaming, can be too simple
5. Finalize your SCMP

Software Construction – Configuration Management



3.1.3 Configuration Management - Documentation

IEEE 828-2005 SCMP Table of Contents

3.1 Introduction
3.2 SCM **management**
  3.2.1 Organization
  3.2.2 SCM responsibilities
  3.2.3 Applicable policies, directives & procedures
  3.2.4 Management of the SCM process
3.3 SCM **activities**
  3.3.1 Configuration identification
    3.3.1.1 Identifying configuration items
    3.3.1.2 Naming configuration items
    3.3.1.3 Acquiring configuration items
  3.3.2 Configuration control
    3.3.2.1 Requesting changes
    3.3.2.2 Evaluating changes
    3.3.2.3 Approving or disapproving changes
    3.3.2.4 Implementing changes
  3.3.3 Configuration status accounting
  3.3.4 Configuration evaluation & reviews
  3.3.5 Interface control
  3.3.6 Subcontractor / vendor control
  3.3.7 Release management and delivery
3.4. SCM **schedules**
3.5. SCM **resources**
3.6. SCM **plan maintenance**

3.1.3 Software Construction Activities – Configuration Management

Software Configuration Management (SCM)

- ~~What is SCM?~~
- ~~Purpose of SCM~~
- ~~SCM Process & Aetivities~~
- SCM Tools and Case Study

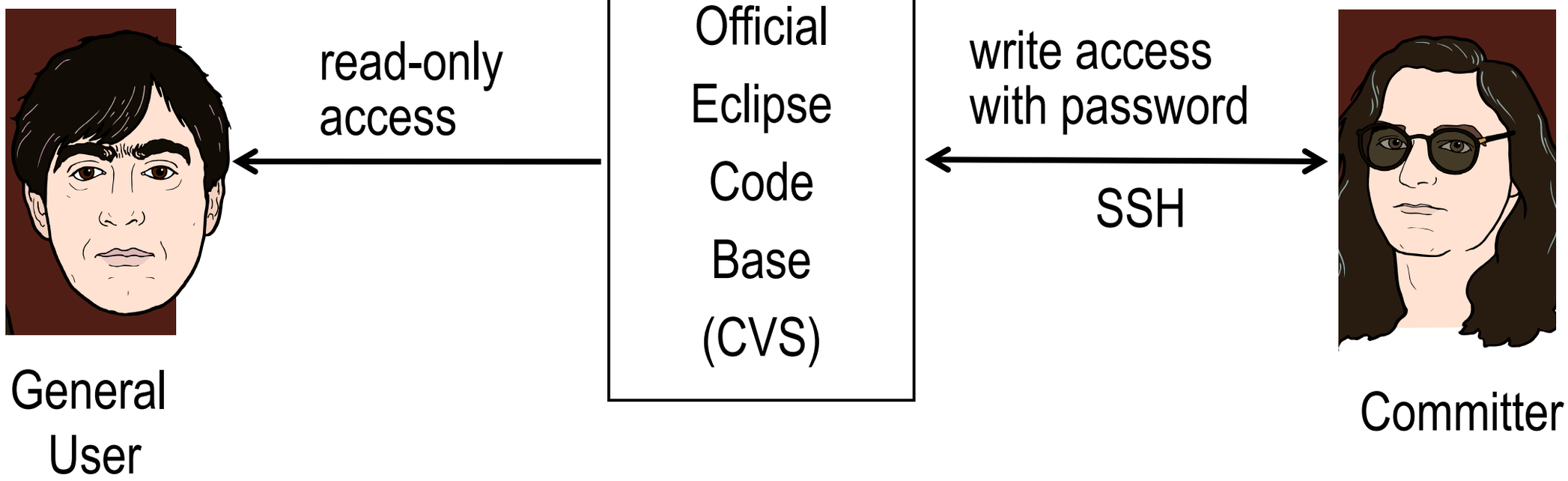
3.1.3 Configuration Management – Tools

- Examples of SCM systems:
  - Subversion
  - CVS
  - Rational Clearcase
  - Microsoft's *SourceSafeTM*
  - Perforce

3.1.3 Configuration Management – Case Study

- CVS - Possible file status:
  - ***Up-to-date***
    identical with the latest revision in the repository.
  - ***Locally Modified***
    File has been edited but has not replaced latest revision
  - ***Needing Patch***
    Someone else has committed newer revision to the repository.
  - ***Needs Merge***
    You have modified the file but someone else has committed a newer revision to the repository, and.
  - ***Unknown***
    A temporary file or never added

3.1.3 Configuration Management – Case Study



Eclipse Configuration Management

3.1.3 Configuration Management – Case Study

Eclipse - Plug-in Version Numbering Scheme

In Eclipse, the format of version numbers are composed of 4 segments:

major.minor.service.qualifier

Each segment captures a different intent:

- the major segment (integer) indicates breakage in the API
- the minor segment (integer) indicates "externally visible" changes
- the service segment (integer) indicates bug fixes and the change of development stream
- the qualifier segment (string) indicates a particular build

Example *1.0.2.x, 1.4.0, 2.1, 5.100.9999.abc*

http://wiki.eclipse.org/Version_Numbering

3.1.3 Configuration Management – Case Study

Eclipse - Plug-in Version Numbering

First development stream
- 1.0.0

Second development stream
- 1.0.100 (indicates a bug fix)
- 1.1.0 (a new API has been introduced)

The plug-in ships as 1.1.0

Third development stream
- 1.1.100 (indicates a bug fix)
- 2.0.0 (indicates a breaking change)

The plug-in ships as 2.0.0

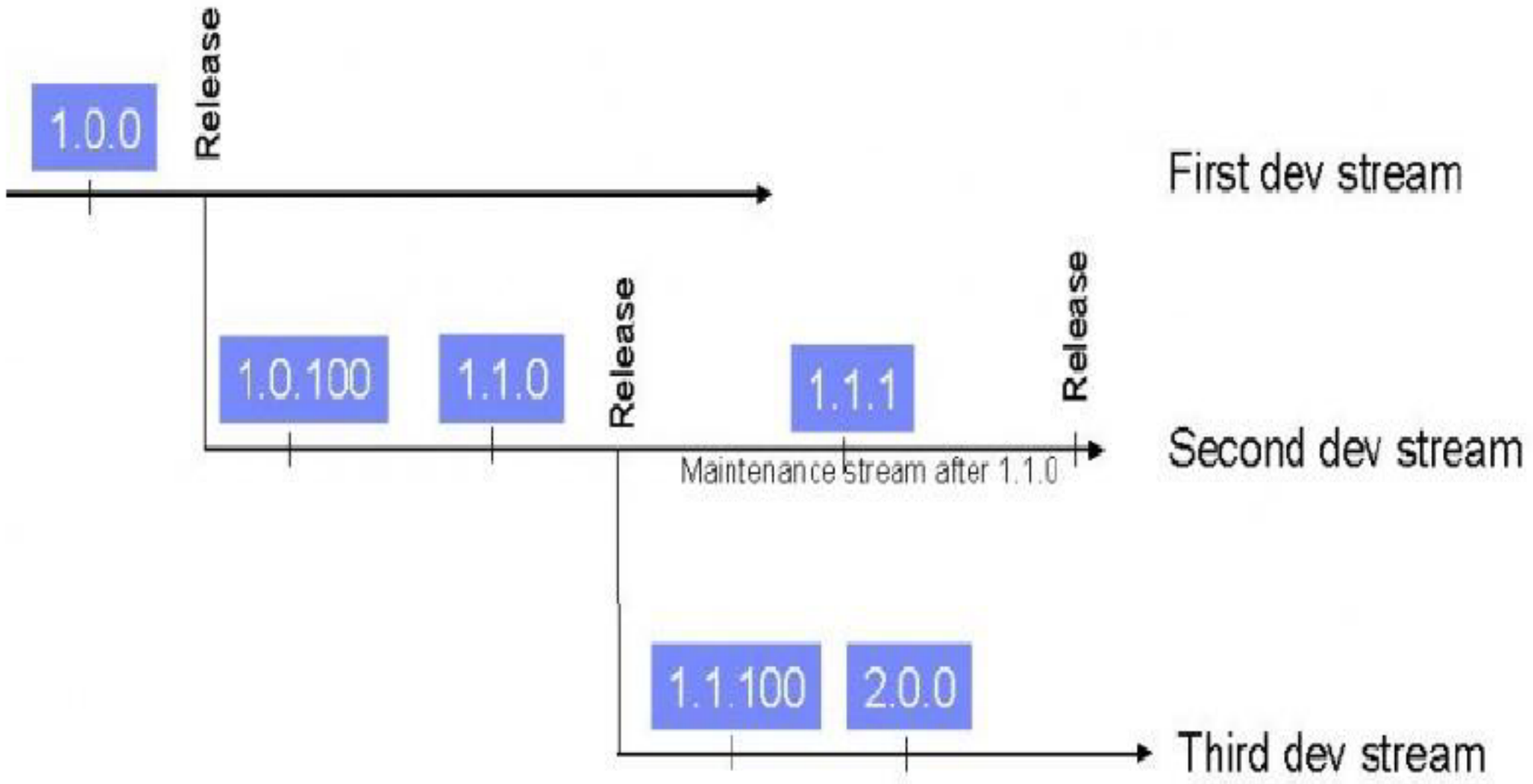
Maintenance stream after 1.1.0
- 1.1.1

The plug-in ships as 1.1.1

http://wiki.eclipse.org/Version_Numbering

3.1.3 Configuration Management – Case Study

- Eclipse - Plug-in Version Numbering



Software Construction -Reference

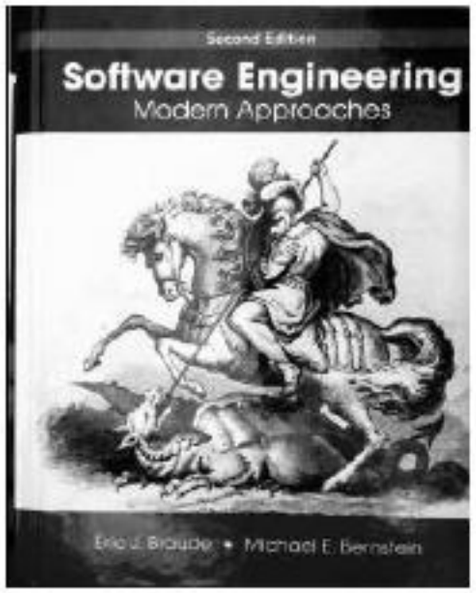
Reading Assignment

Textbook: *"Code Complete"*, 2nd edition

- Chapter 29: Integration
- Chapter 27: Managing Construction

Software Construction -Reference :

REFERENCE



Textbook: *"Software Engineering, Modern Approaches"*, 2nd edition

- Chapter 6: Software Configuration Management