ECE25100:
OBJECT ORIENTED PROGRAMMING

Note 1 _ Software Design and Programming Concepts

Instructor: Xiaoli Yang

What is Software?

Software is:

(1) instructions (computer programs) that when executed provide desired features, function, and performance;

(2) data structures that enable the programs to adequately manipulate information and

(3) documentation that describes the operation and use of the programs.

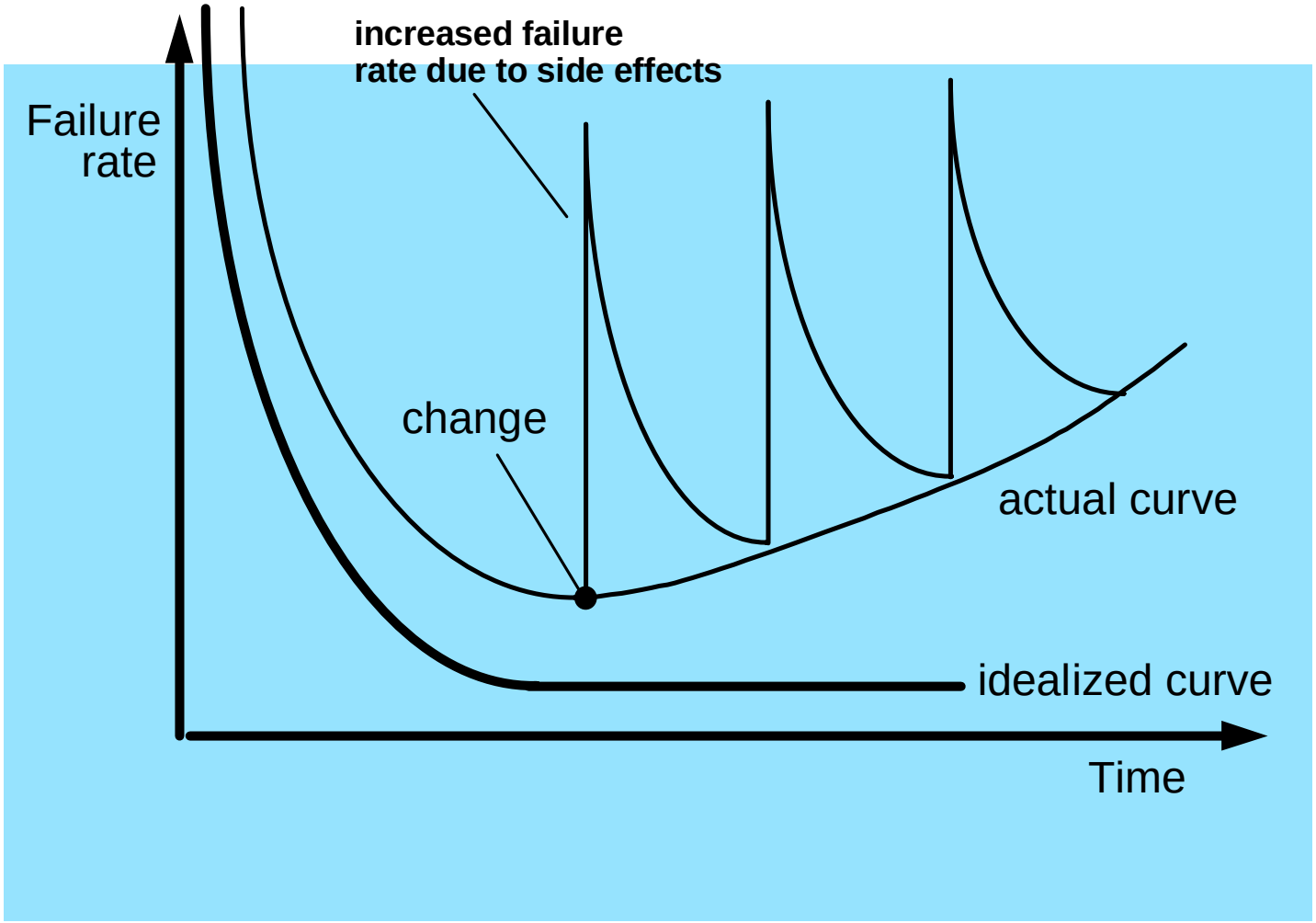
The Nature of Software

- Software is intangible
  - Hard to understand development effort
- Software is easy to reproduce
  - Cost is in its development
    - in other engineering products, manufacturing is the costly stage
- The industry is labor-intensive
  - Hard to automate
- Untrained people can hack something together
  - Quality problems are hard to notice
- Software is easy to modify
  - People make changes without fully understanding it

The Nature of Software

- Software is developed or engineered, it is not manufactured in the classical sense.
- Software doesn't "wear out."
- Although the industry is moving toward component-based construction, most software continues to be custom-built.

Wear vs. Deterioration



Software Applications

- System software
- Application software
- Engineering/Scientific software
- Embedded software
- Product-line software
- Web/Mobile applications)
- AI software (robotics, neural nets, game playing)

Legacy Software

Why must it change?

- software must be adapted to meet the needs of new computing environments or technology.
- software must be enhanced to implement new business requirements.
- software must be extended to make it interoperable with other more modern systems or databases.
- software must be re-architected to make it viable within a network environment.

WebApps

- Modern WebApps are much more than hypertext files with a few pictures
- WebApps are augmented with tools like XML and Java to allow Web engineers including interactive computing capability
- WebApps may standalone capability to end users or may be integrated with corporate databases and business applications

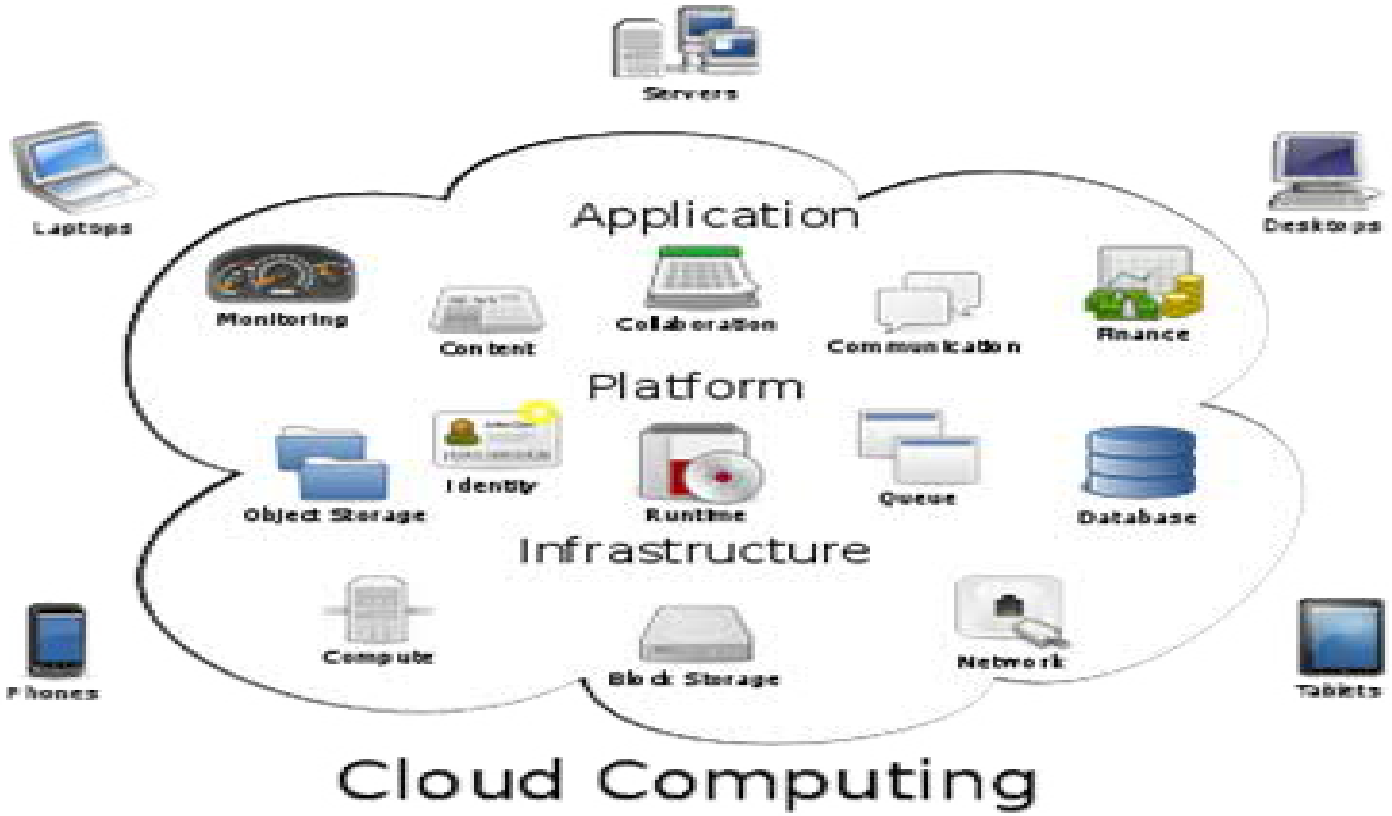
WebApps

- Semantic web technologies (Web 3.0) have evolved into sophisticated corporate and consumer applications that encompass semantic databases that require web linking, flexible data representation, and application programmer interfaces (API's) for access
- The aesthetic nature of the content remains an important determinant of the quality of a WebApp.

Mobile Apps

- Reside on mobile platforms such as cell phones or tablets
- Contain user interfaces that take both device characteristics and location attributes
- Often provide access to a combination of web-based resources and local device processing and storage capabilities
- Provide persistent storage capabilities within the platform
- A mobile web application allows a mobile device to access to web-based content using a browser designed to accommodate the strengths and weaknesses of the mobile platform
- A mobile app can gain direct access to the hardware found on the device to provide local processing and storage capabilities
- As time passes these differences will become blurred

Cloud Computing



Cloud Computing

- Cloud computing provides distributed data storage and processing resources to networked computing devices
- Computing resources reside outside the cloud and have access to a variety of resources inside the cloud
- Cloud computing requires developing an architecture containing both frontend and backend services
- Frontend services include the client devices and application software to allow access
- Backend services include servers, data storage, and server-resident applications
- Cloud architectures can be segmented to restrict access to private data

Product Line Software

- Product line software is a set of software-intensive systems that share a common set of features and satisfy the needs of a particular market
- These software products are developed using the same application and data architectures using a common core of reusable software components
- A software product line shares a set of assets that include requirements, architecture, design patterns, reusable components, test cases, and other work products
- A software product line allow in the development of many products that are engineered by capitalizing on the commonality among all products with in the product line

Software Engineering

- Some realities:
  - a concerted effort should be made to understand the problem before a software solution is developed
  - design becomes a pivotal activity
  - software should exhibit high quality
  - software should be maintainable
- The seminal definition:
  - [Software engineering is] the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines.

Software Engineering

- The IEEE definition:
  - Software Engineering:
  - (1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.
  - (2) The study of approaches as in (1).

Software Engineering and the Engineering Profession

- The term Software Engineering was coined in 1968
  - People began to realize that the principles of engineering should be applied to software development
- Engineering is a licensed profession
  - In order to protect the public
  - Engineers design artifacts following well accepted practices which involve the application of science, mathematics and economics
  - Ethical practice is also a key tenet of the profession
- In many countries, much software engineering does not require an engineering license, but is still engineering

Software Engineering Projects

Most projects are evolutionary or maintenance projects, involving work on legacy systems

- Corrective projects: fixing defects
- Adaptive projects: changing the system in response to changes in
  - Operating system
  - Database
  - Rules and regulations
- Enhancement projects: adding new features for users
- Reengineering or perfective projects: changing the system internally so it is more maintainable
- 'Green field' projects
  - New development
  - The minority of projects

Software Engineering Projects (Cont'd)

- Projects that involve building on a framework or a set of existing components.
  - A framework is an application that is missing some important details.
    - E.g. Specific rules of this organization.
  - Such projects:
    - Involve plugging together components that are:
      - Already developed.
      - Provide significant functionality.
    - Benefit from reusing reliable software.
    - Provide much of the same freedom to innovate found in green field development.

Software Design

- What is design?
  - Noun: mental plan, preliminary sketch or outline
  - Verb: to conceive in the mind; to invent
- What is software design?
  - As a product
    - Output of design process
  - As a process
    - Approach to doing design

Nature of Design

Design

- Form of problem solving

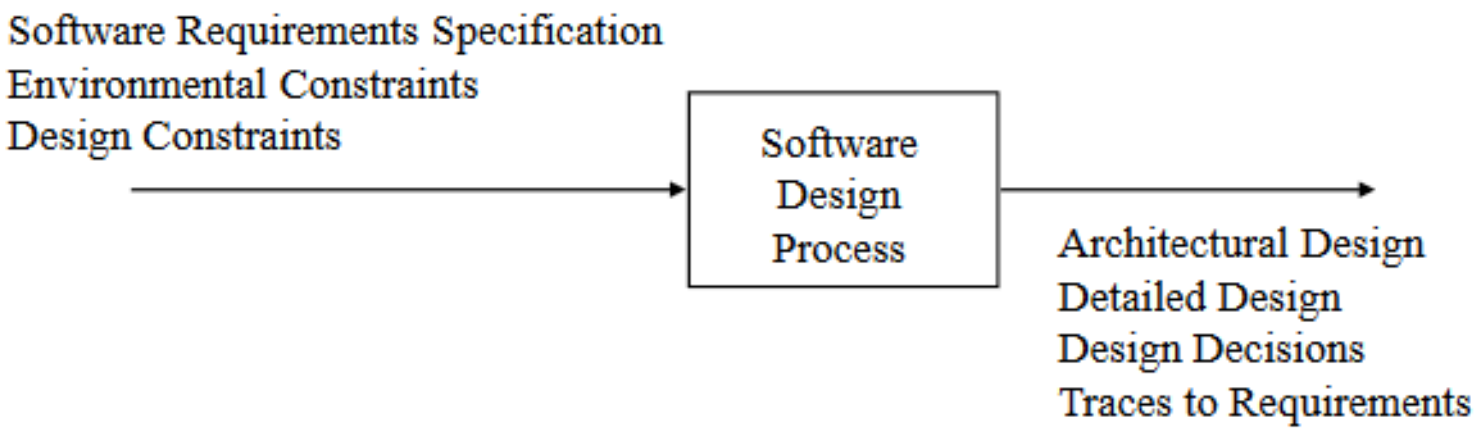
Design as “wicked problem”

- Unlike an algorithm
  - There is no one “correct ” solution
- Tradeoffs in design
  - E.g., Structure vs performance
  - Centralized vs. distributed
  - Sequential vs. concurrent

Software Design Activities

- Architectural Design
  - Structure system into components
  - Define the interfaces between components
- Detailed Design
  - Define internal logic
  - Define internal data structures

Context of Software Design



Inputs to Software Design

- Software requirements specification
  - Describes WHAT system shall do not HOW
  - External view of system to be developed
- Environmental constraints
  - Hardware, language, system usage
- Design constraints
  - Design method
  - Design notation

Outputs From Software Design

- Architectural Design
  - Overall description of software structure
    - Textual and Graphical
  - Specification of software components and their interfaces
    - Modules, classes
- Detailed Design of each component
  - Internal logic
  - Internal data structures
- Design decisions made
  - Design rationale
- Traces to requirements

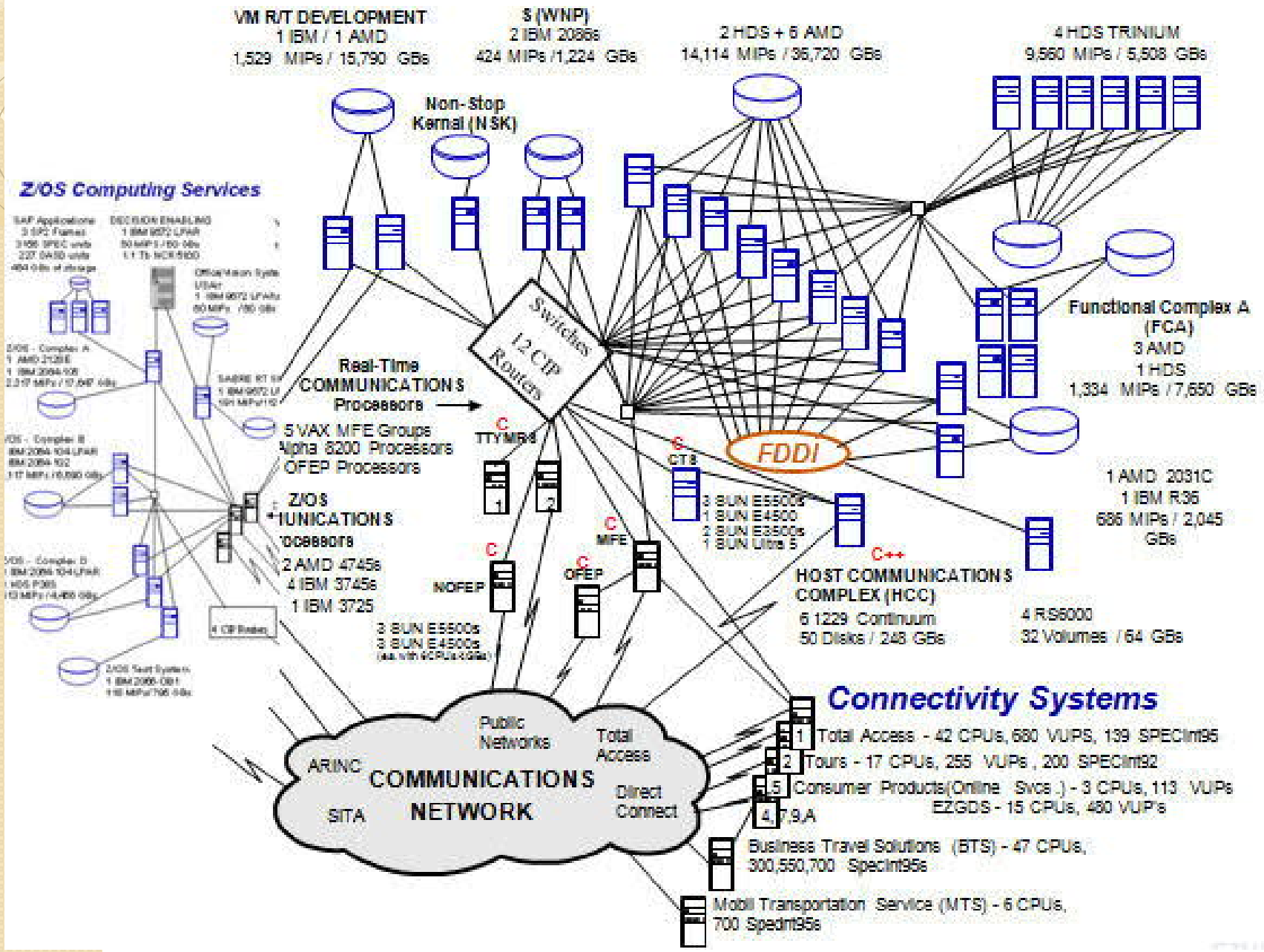
Software Design Process

- Software life cycle (a.k.a. process)
  - Phased approach to software development
- Software life cycle (a.ka. process) models
  - Waterfall – limitations of Waterfall Model
  - Incremental - evolutionary prototyping
  - Exploratory - throwaway prototyping
  - Spiral model – risk driven process model

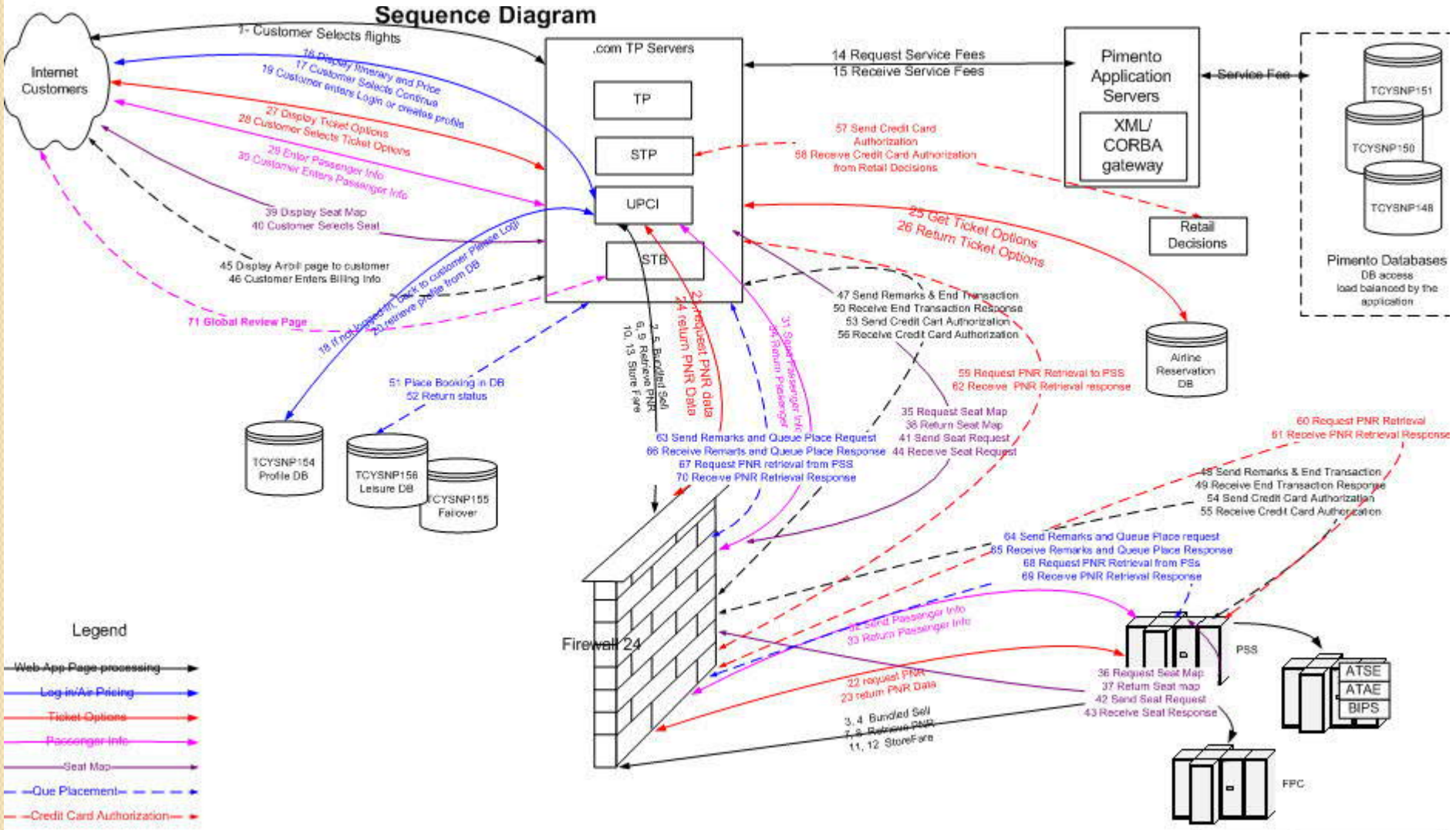
Difficulties and Risks in Software Design

- Complexity and large numbers of details
- Uncertainty about technology
- Uncertainty about requirements
- Uncertainty about software engineering skills
- Constant change
- Deterioration of software design
- Political risks

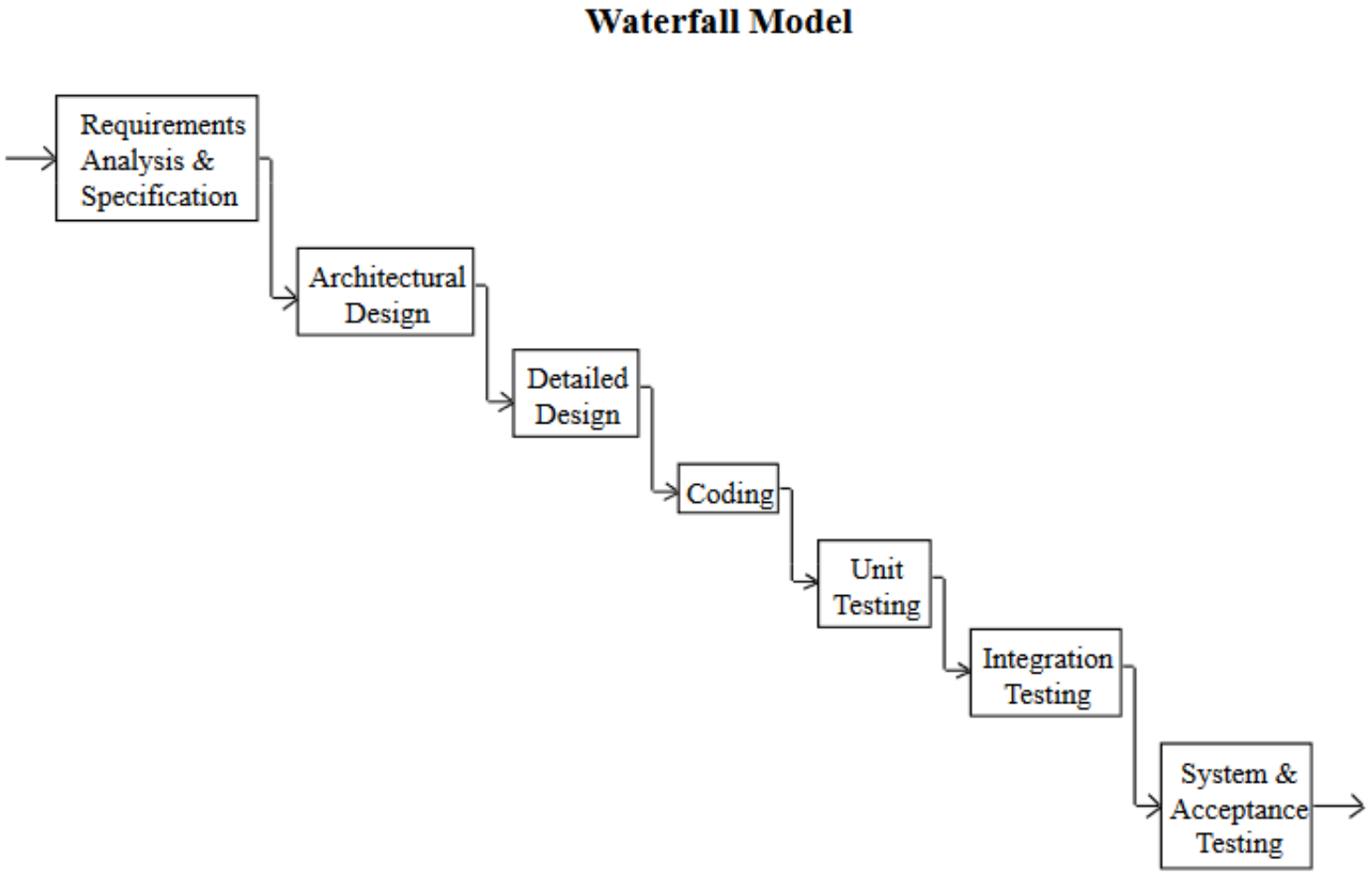
Complex Server Connections



Complex Message Flow



The Waterfall Model



Software Life Cycle Model: Software Definition

- Requirements Analysis and Specification
  - Analysis of user's problem
  - Specification of "what" system shall provide user
- Architectural Design
  - Specification of "how" system shall be structured into components
  - Specification of interfaces between components.

Software Life Cycle Model: Software Construction

- Detailed Design
  - Internal design of individual components
    - Design of logic and data structures
- Coding
  - Map component design to code
- Unit Testing
  - Test individual components

Software Life Cycle Model: Software Integration and Test

- Integration Testing
  - Gradually combine components and test combinations
- System Testing
  - Test of entire system against software requirements
- Acceptance Testing
  - Test of entire system by user prior to acceptance

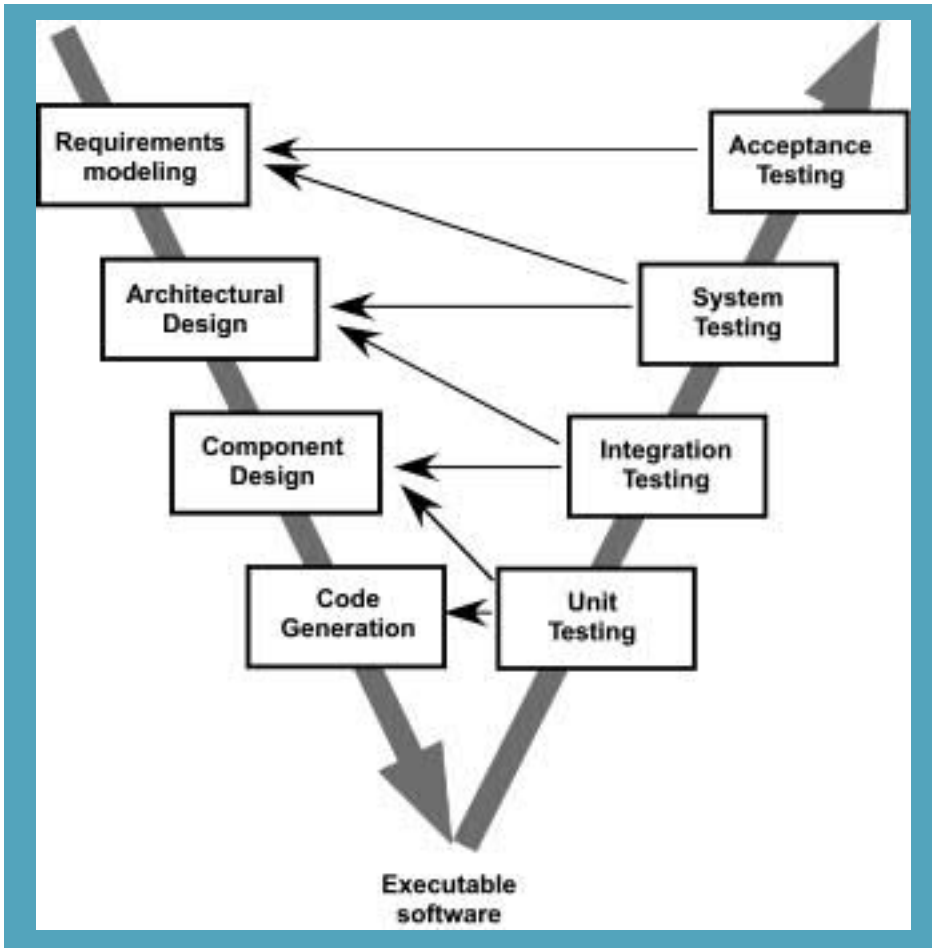
Software Life Cycle Model: Software Maintenance

- Modification of software system after installation and acceptance
  - Fix software errors
  - Improve performance
  - Address changes in user requirements
- Often implies significant software redesign

Limitations of Waterfall Model

- Does not show iteration in software life cycle
- Does not show overlap between phases
- Software requirements are tested late in life cycle
- Operational system available late in life cycle.

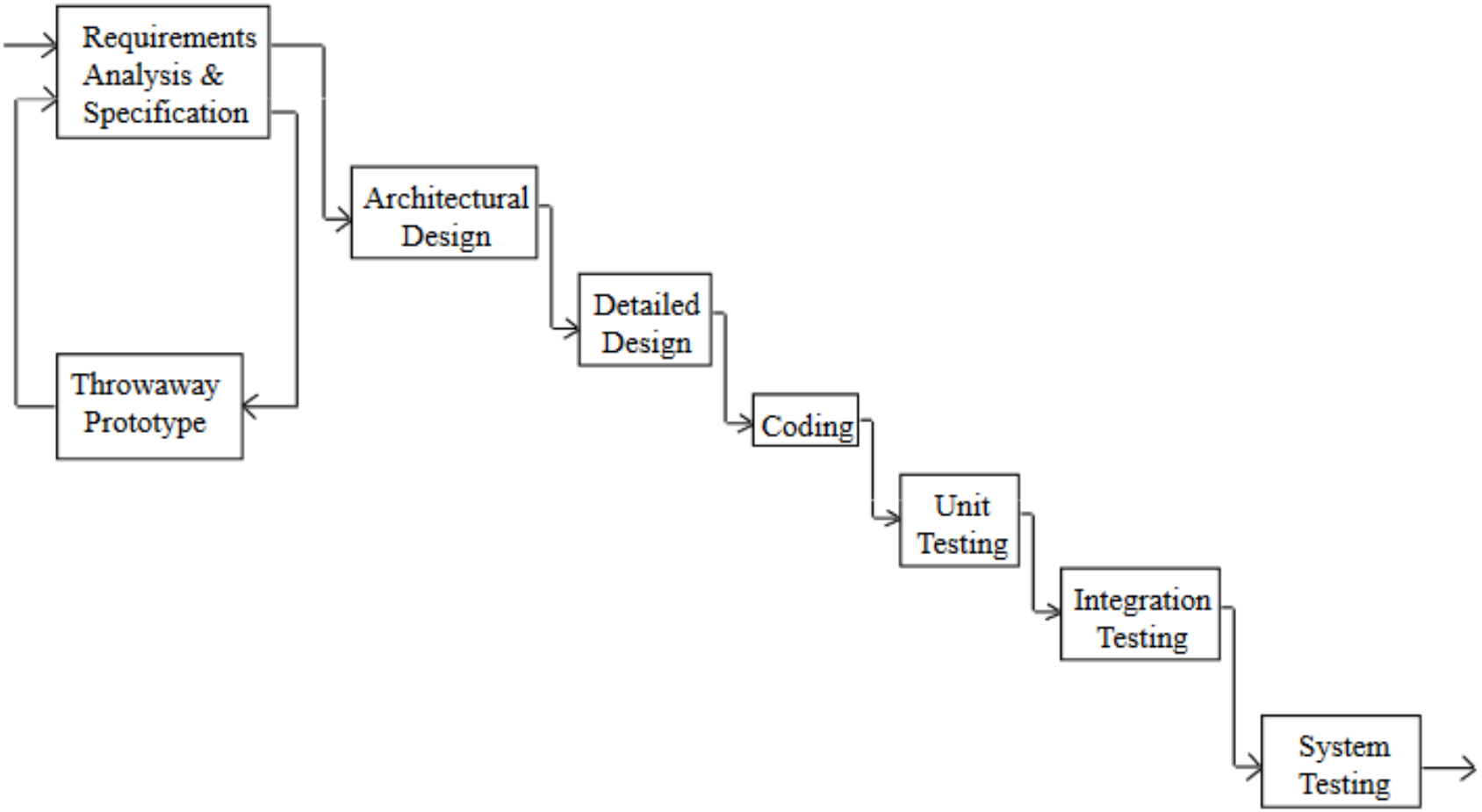
The V-Model


Requirements modeling
Acceptance Testing
Architectural Design
System Testing
Component Design
Integration Testing
Code Generation
Unit Testing
Executable software

Prototyping During Requirements Phase

- **Problem**
  - Software requirements are tested late in life cycle
- Solution
  - Use throw-away prototyping
- Help ensure requirements are understood
- Also first attempt at designing system
  - Design of key file and data structures
  - Design of user interface
  - Early design tradeoffs

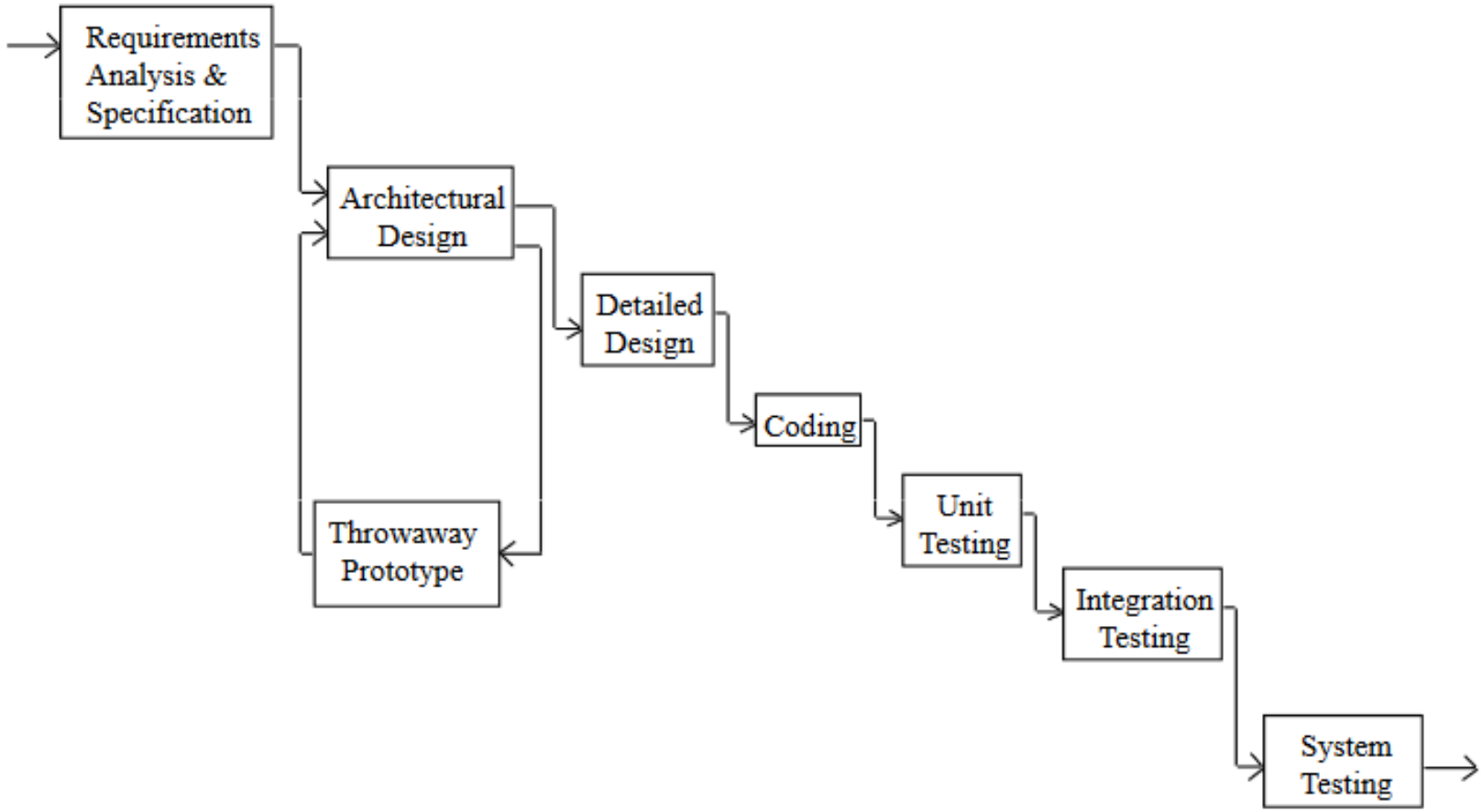
Impact of Throwaway Prototyping on Software Life Cycle



Throw-away Prototyping in Design

- **Objectives**
  - Test design early
  - Experiment with alternative design decisions
- Examples of prototyping in design
  - Algorithm design
    - Experiment with – speed, accuracy
  - Early Performance analysis
    - Measure timing parameters
  - User interface

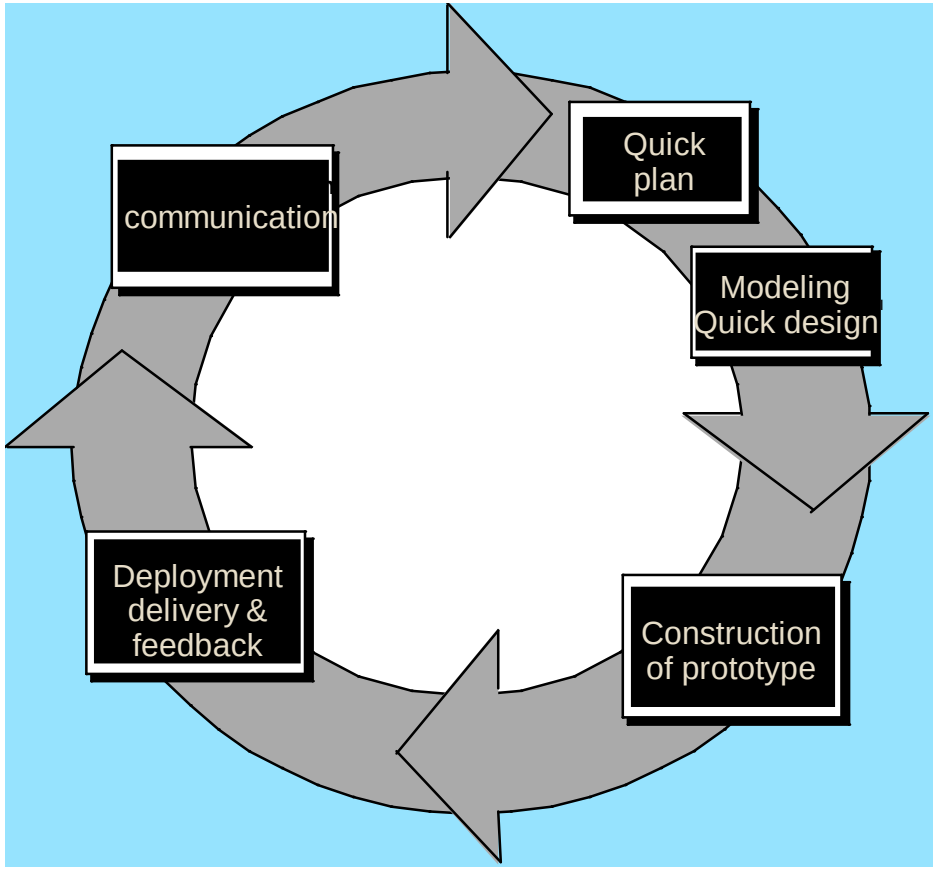
Impact of Throwaway Prototyping on Architectural Design Phase



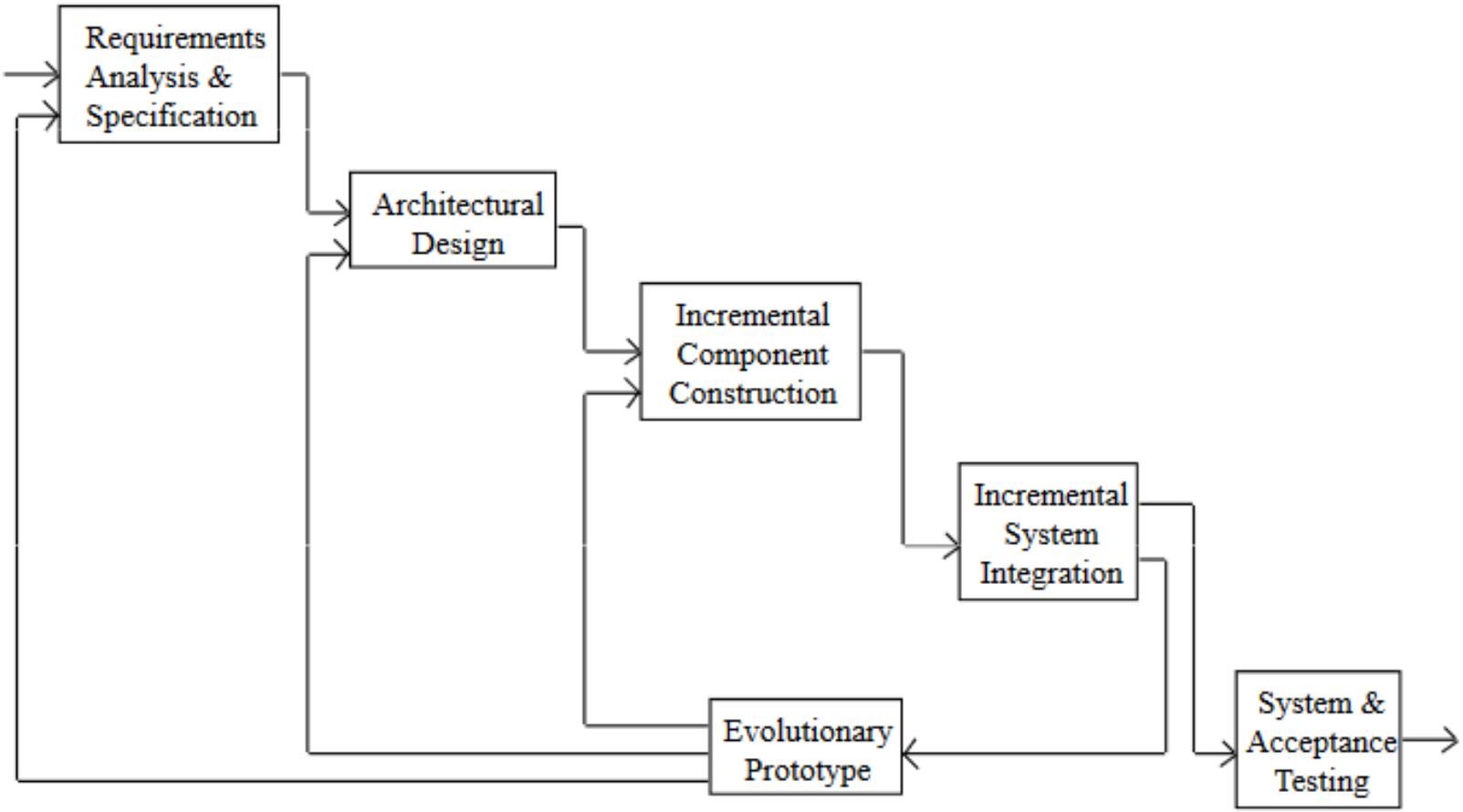
Incremental Development

- Problem
  - Operational system available late in life cycle
- Solution
  - Use incremental development
  - Also known as evolutionary prototyping
- Objective
  - Subset of system working early
  - Gradually build on
  - Prototype evolves into production system

Evolutionary Models: Prototyping



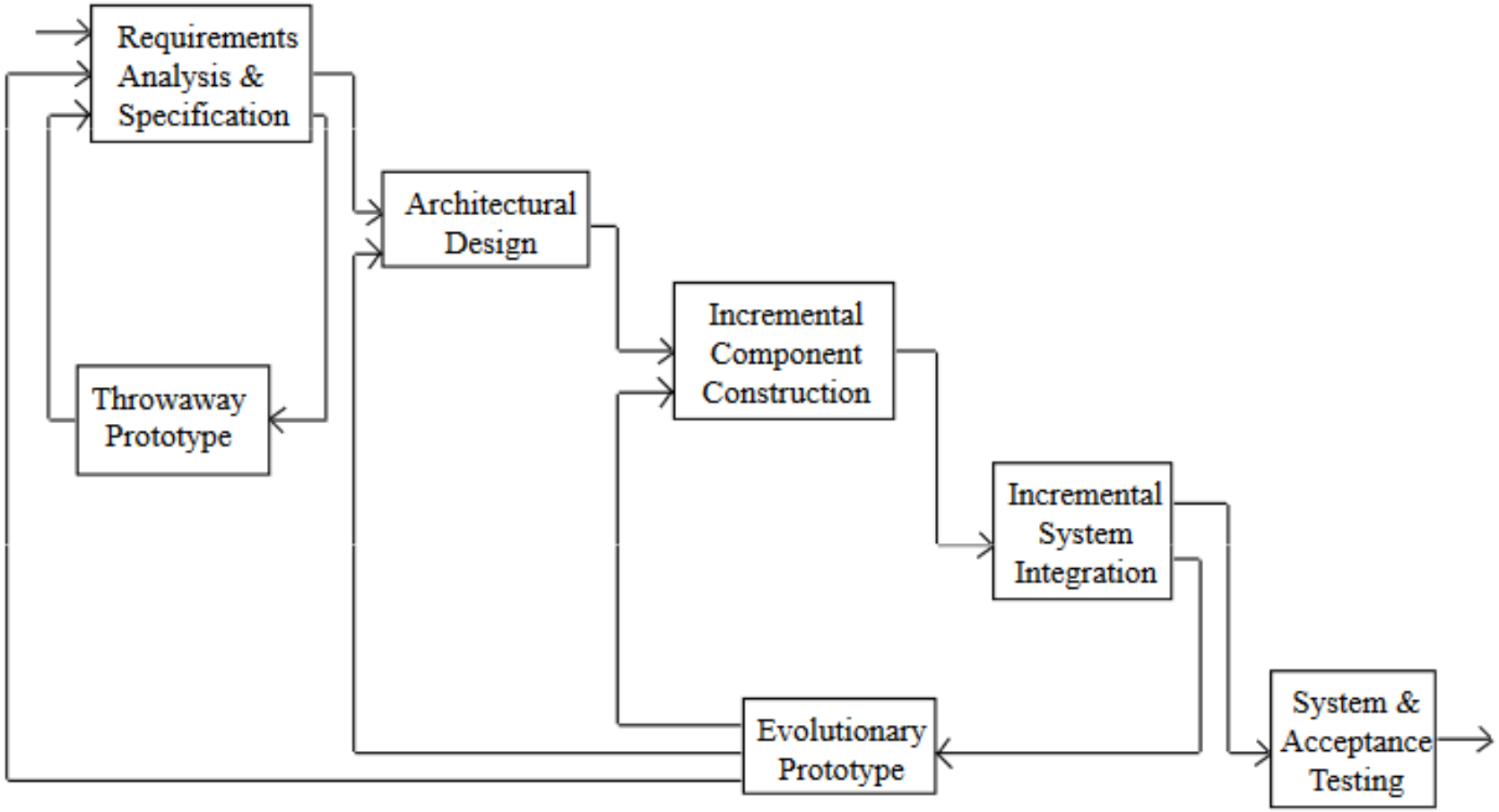
Incremental Development Software Life Cycle



Should Prototype Evolve into Production System

- Tradeoff
  - Rapid development
  - Quality of product
- Throw-away prototype
  - Speed, not quality is goal
  - Must not evolve into production system
- Evolutionary prototype
  - Must emphasize quality
  - Maintainability is key issue

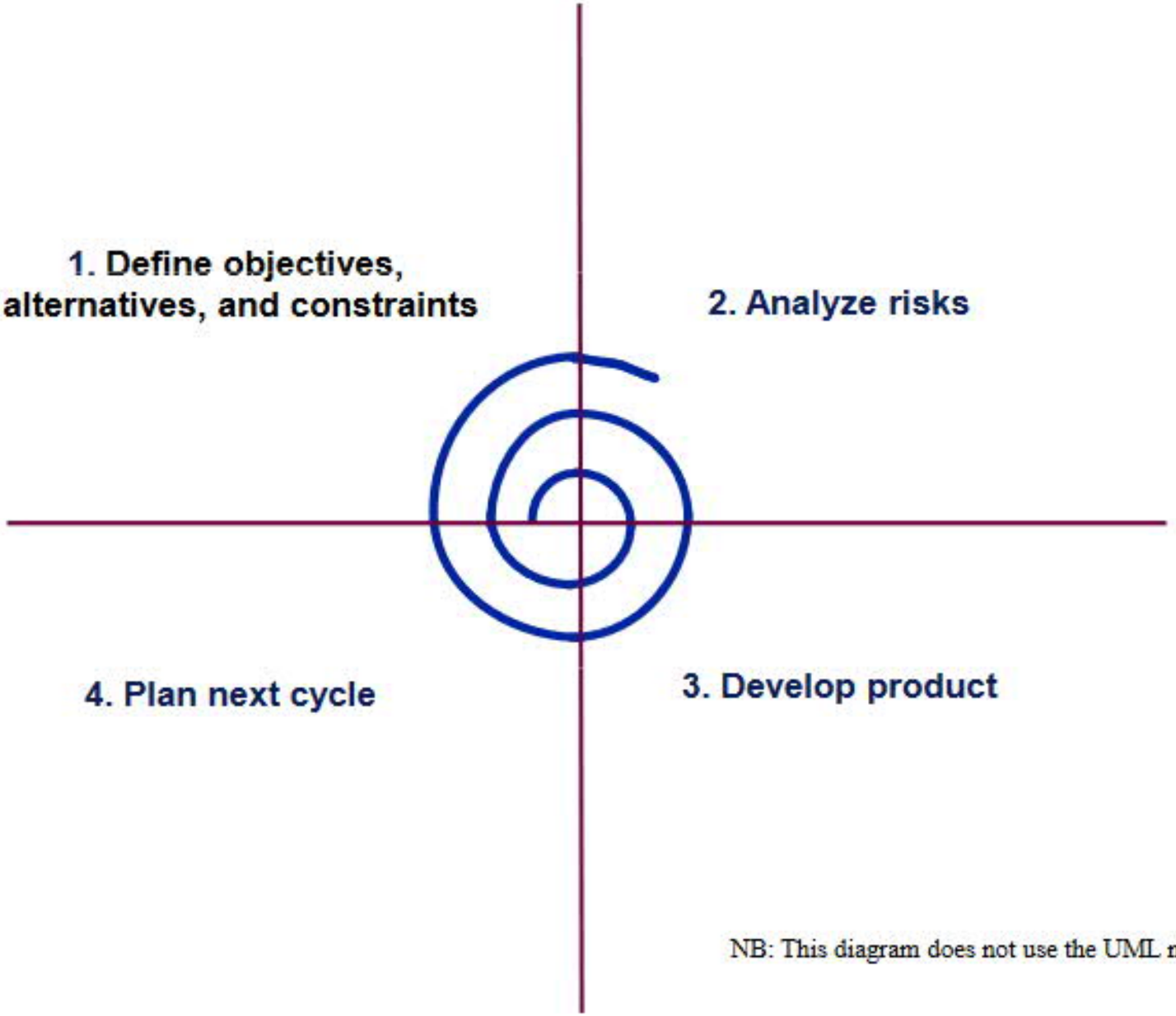
Combined Throwaway Prototyping / Incremental Development Software Life Cycle Model



Spiral Process Model (SPM)

- SPM consists of four main activities that are repeated for each cycle (Fig. 5.6)
  - Defining objectives, alternatives and constraints
  - Analyzing risks
  - Developing and verifying product
  - Spiral planning
- Number of cycles is project specific
- Risk driven process
  - Analyze risks in second quadrant

Figure 5.6 The spiral process model



NB: This diagram does not use the UML notation

Software Quality

- Usability
  - Users can learn it fast and get their job done easily
- Efficiency
  - It doesn't waste resources such as CPU time and memory
- Reliability
  - It does what it is required to do without failing
- Maintainability
  - It can be easily changed
- Reusability
  - Its parts can be used in other projects, so reprogramming is not needed

Software Quality and the Stakeholders

Customer:
solves problems at
an acceptable cost in
terms of money paid and
resources used

User:
easy to learn;
efficient to use;
helps get work done

Developer:
easy to *design*;
easy to *maintain*;
easy to *reuse* its parts

Development manager:
sells more and
pleases customers
while costing less
to develop and maintain

Software Quality: Conflicts and Objectives

- The different qualities can conflict
  - Increasing efficiency can reduce maintainability or reusability
  - Increasing usability can reduce efficiency
- Setting objectives for quality is a key engineering activity
  - You then design to meet the objectives
  - Avoids 'over-engineering' which wastes money
- Optimizing is also sometimes necessary
  - E.g. obtain the highest possible reliability using a fixed budget

Short Term Vs. Long Term Quality

- Short term:
  - Does the software meet the customer's immediate needs?
  - Is it sufficiently efficient for the volume of data we have today?
- Long term:
  - Maintainability
  - Customer's future needs
  - Scalability: Can the software handle larger volumes of data?

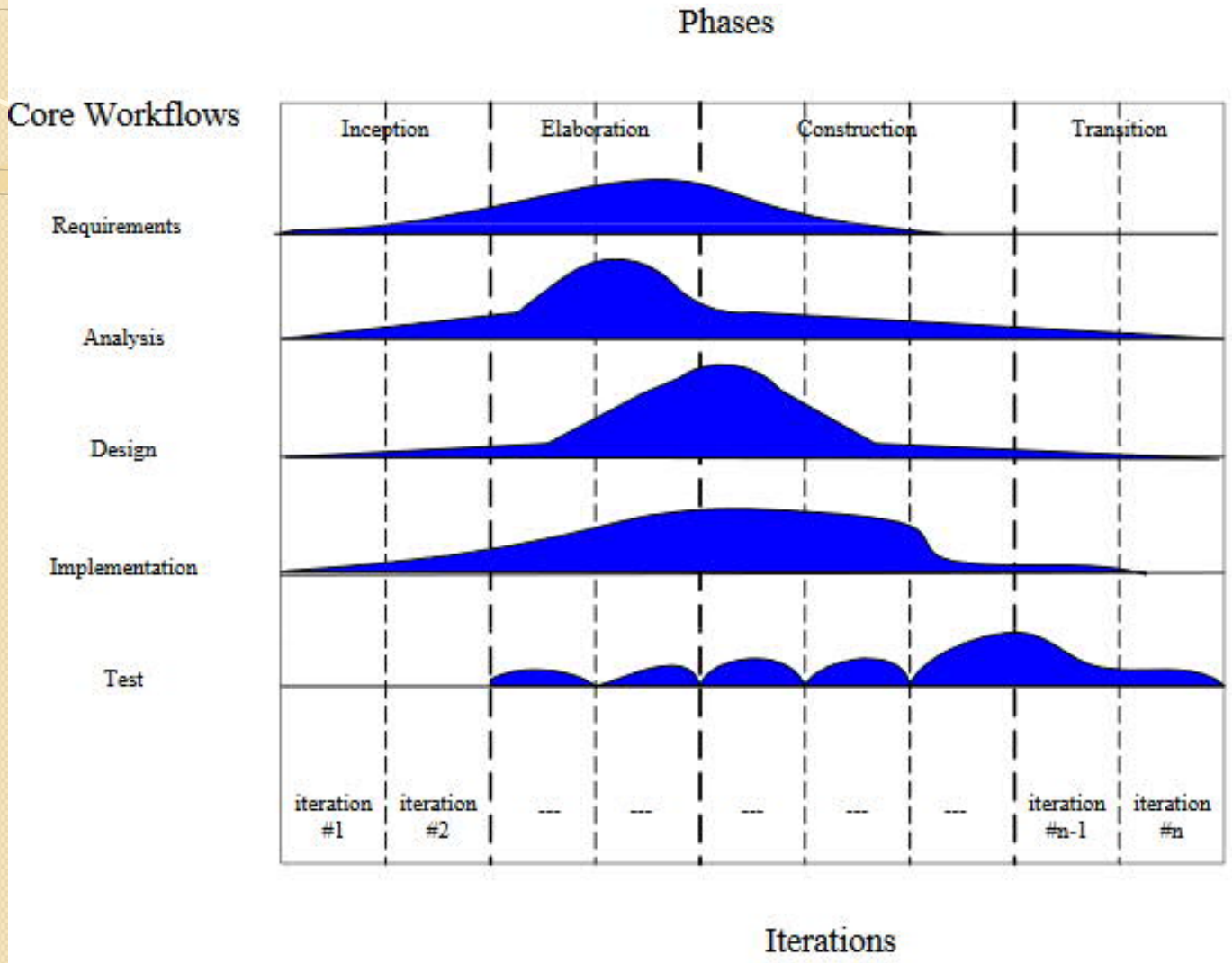
Unified Software Development Process

- Risk driven iterative process
  - Also known as Rational Unified Process
- Workflow
  - Sequence of activities that produces a result of observable value
- Workflows in Unified Process
  - Requirements
    - Product: Use case model.
  - Analysis
    - Product: Analysis model.
  - Design
    - Products: design model and deployment model
  - Implementation
    - Product: software implementation
  - Test
    - Products: Test cases and test results

Unified Software Development Process

- Phase
  - Time between two major milestones
- Phases in Unified Process
  - Inception
    - Seed idea is developed
  - Elaboration
    - Software architecture is defined
  - Construction
    - Software is built to the point at which it is ready for release
  - Transition
    - Software is turned over to the user community

Figure 3.5: Unified Software Development Process



Software Design Concepts

- Objects and Classes
- Information Hiding
- Inheritance
- Concurrency
- Finite State Machines

What is a programming language ?

- A programming language essentially represents a set of words, rules and tools that are used to explain (or define) your program. There are many different programming languages just as there are many different "spoken" languages.

Programming Languages

Machine Language **Assembly Language** **High-Level Language**

- Machine language is a set of primitive instructions built into every computer.
- The instructions are in the form of binary code, so you have to enter binary codes for various instructions.
- Program with native machine language is a tedious process.
- For example, to add two numbers, you might write an instruction in binary like this:

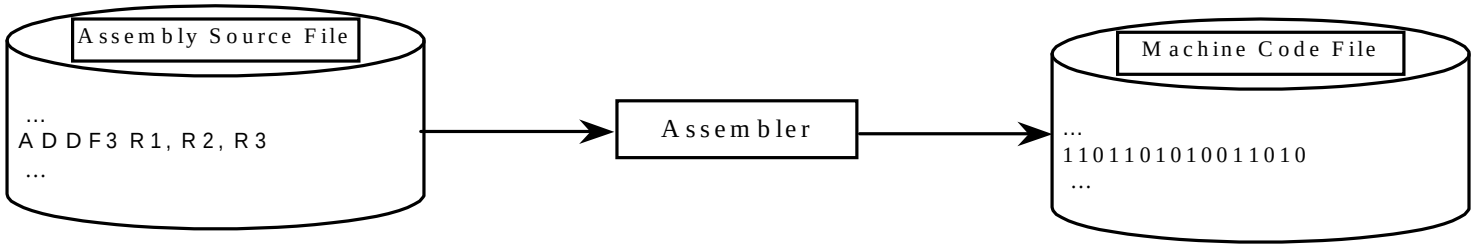
```
1101101010011010
```

Programming Languages

Machine Language **Assembly Language** **High-Level Language**

- Assembly languages were developed to make programming easy.
- Since the computer cannot understand assembly language, however, a program called assembler is used to convert assembly language programs into machine code.
- For example, to add two numbers, you might write an instruction in assembly code like this:

  ADDF3 R1, R2, R3



Programming Languages

Machine Language **Assembly Language** **High-Level Language**

The high-level languages are English-like and easy to learn and program. For example, to add two numbers, you might write an instruction in a high-level language code like this :

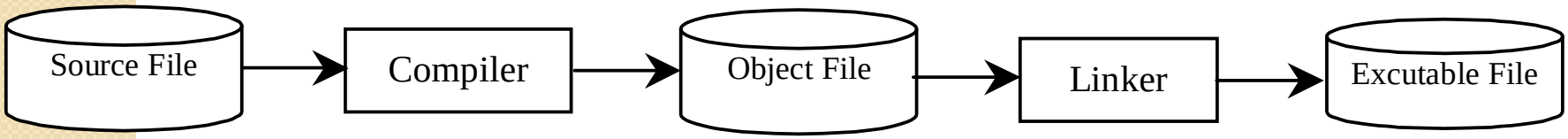
```
int a = b + c;
```

Popular High-Level Languages

- COBOL (COmmon Business Oriented Language)
- FORTRAN (FORmula TRANslation)
- BASIC (Beginner All-purpose Symbolic Instructional Code)
- Pascal (named for Blaise Pascal)
- Ada (named for Ada Lovelace)
- C (whose developer designed B first)
- Visual Basic (Basic-like visual language developed by Microsoft)
- Delphi (Pascal-like visual language developed by Borland)
- C++ (an object-oriented language, based on C)
- Java (We use it in the class)

Compiling Source Code

- A program written in a high-level language is called a source program.
- Since a computer cannot understand a source program. Program called a compiler is used to translate the source program into a machine language program called an object program.
- The object program is often then linked with other supporting library code before the object can be executed on the machine.

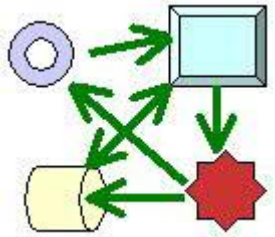


What is a procedural programming language ?

- Procedural Programming languages allow programmers to write code:
  - according to a set of steps for solving a problem
  - in a top-down manner
  - with a chronological ordering in mind
- They often follow these steps:
  - Get some data input from the user
  - Perform some calculations and make some decisions
  - Display some data output on the screen

What is a Object-Oriented programming ?

- For Object-Oriented Programming, instead of thinking of "steps" to solve a problem, we need to think about and identify the objects involved in the problem and how they'll interact.
  - involves identifying objects and their relationships
  - involves first defining object state and behaviors
  - and then using them with one another



- OO programming is all about knowing:
  - which objects to use
  - the kind of information we need to know about the objects
  - how objects behave and
  - how to use them with each other

Main powerful concepts in OOP

Inheritance

- promotes code sharing and re-usability
- intuitive hierarchical code organization

Encapsulation

- provides notion of security for objects
- reduces maintenance headaches
- more robust code

Polymorphism

- simplifies code understanding
- standardizes method naming

OO Programming Languages

- Through these powerful concepts, Object-Oriented code is typically:
  - easier to understand (relates to real world objects)
  - better organized and hence easier to work with
  - simpler and smaller in size
  - more modular
  - better quality

OO Programming Languages (Cont.)

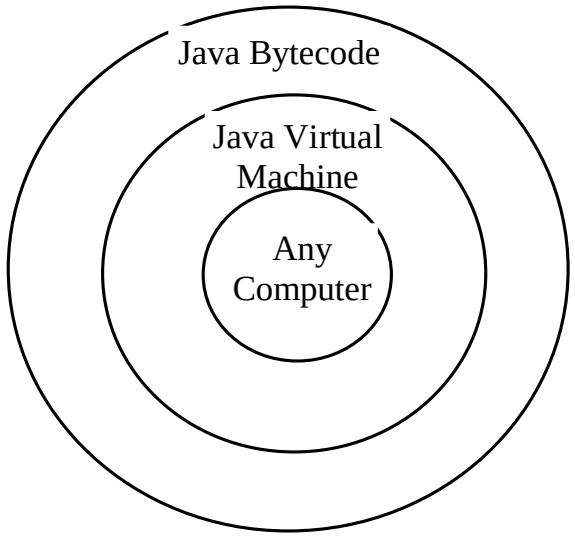
- This leads to:
  - high productivity and a shorter delivery cycle
  - less manpower required
  - reduced costs for maintenance
  - more reliable and robust software
  - pluggable systems (updated UI's, less legacy code)

The JAVA Programming Language

- Java is an object-oriented programming language originally from SUN Microsystems.
- Java is an interpreted language:
  - runs using a Java Virtual Machine (JVM)
    - is REQUIRED to run any JAVA program
    - is machine-independent.
    - also exists within internet browsers (e.g., Netscape, Internet Explorer)
  - does not have an executable file.

Compiling Java Source Code

- Nowadays computers are networked to work together. Java was designed to run object programs on any platform.
- With Java, you write the program once, and compile the source program into a special type of object code, known as bytecode.
- The bytecode can then run on any computer with a Java Virtual Machine, as shown in Figure Java Virtual Machine is a software that interprets Java bytecode.



The JAVA Programming Language

- First version released in 1995
- Four major versions released since then
  - JDK 1.02 (1996) JDBC, Distributed Objects
  - JDK 1.1 (1997) New Event Model
  - J2SE 1.2 (1998) Swing
  - J2SE 1.3 (2000) Cleanup
  - J2SE 1.4 (2002) printing and long term persistence for JavaBeans components etc.
  - J2SE 5.0 (2004) More interesting features
  - Java SE 6 (2006) released
  - Java SE 7 (2011) released
  - Java SE Development Kit 8u111
  - Java Platform, Standard Edition 9.0.1

    Java SE 9.0.1 is the latest update to the Java Platform. This release includes important bug fixes. Oracle strongly recommends that all Java SE 9 users upgrade to this release

The JAVA Programming Language

- JAVA has become a basis for new technologies:
  - Java Beans
  - Enterprise Java Beans (EJB's)
  - Servlets and Java Server Pages (JSPs)
- In addition, many packages have been added which extend the language to provide special features:
  - Java Media Framework (for video streaming, webcams, MP3 files, etc)
  - Java 3D (for 3D graphics)
  - J2ME (for wireless communications such as PDAs and cellphones)

Java Platforms

- **Java Standard Edition (Java SE)**
  - **Java SE** can be used to develop client-side standalone applications or applets.
- **Java Enterprise Edition (Java EE)**
  - **Java EE** can be used to develop server-side applications such as Java servlets and Java ServerPages.
- **Java Micro Edition (Java ME)**
  - **Java ME** can be used to develop programs for mobile wireless information devices such as cellular phones and personal digital assistants (PDAs).

Reasons to use JAVA

- Architecture independence
  - ideal for internet applications
  - code written once, runs anywhere
  - reduces cost $$$
- Distributed and multi-threaded
  - useful for internet applications
  - programs can communicate over network (e.g., web)
  - uses RMI (Remote Method Invocation) API
- Dynamic
  - code loaded only when needed
- Memory managed
  - automatic memory allocation / de-allocation
  - garbage collector releases memory for unused objects
  - simpler code & less debugging
- Robust
  - no pointers
  - strongly typed

Useful Websites

- General Java information:
http://www.oracle.com/technetwork/java/index.html
- Java Platform (JDK) 9
http://www.oracle.com/technetwork/java/javase/downloads/index.html
- JDK 9 API Specification:
https://docs.oracle.com/javase/9/docs/api/index.html?overview-summary.html
- Eclipse: http://www.eclipse.org/
- Jcreator: http://jcreator.software.informer.com
- TextPad: https://www.textpad.com/download/

Writing the first program

- To keep things simple our first few program will all be written as just a single main program.
- In java, the file in which your program resides must contain a .java extension (e.g. example 1: MyFirstProgram.java).
- Then, the program must be wrapped in a class definition which is the same as the file basename (MyFirstProgram). Careful, Java is case-sensitive.

```
class MyFirstProgram { … }
```

Writing the first program

```
public class MyFirstProgram {
    public static void main(String[ ] args){
        //print out the words Hello Everyone to the console
        System.out.println("Hello Everyone");
    }
}
```

- main is defined similar to C, but with a few more modifiers. These are all required. No shortcuts.
- Just as in C, main(..) is the principle entry point into the program. When you say

  java Program

  Java looks in Program for a procedure named main. This is where the program starts.
- To print to stdout in java use

Compiling/running first java program

- Create source code file (MyFirstProgram.java).
- To compile:

  prompt >> javac MyFirstProgram.java
- This produces byte code file named

  MyFirstProgram.class
- To run:

  prompt >> java MyFirstProgram

Observations

- Java's compiler: .class file is not machine code. It is intermediate form called Java Byte code. Can run on any platform as long as platform has a Java Virtual Machine (JVM).
- Java's Interpreter: The second step on previous slide invokes the JVM to interpret the byte code on the given platform.

Observations (Cont.)

- In theory, byte code can be moved to another platform and be run there without recompiling – this is the magic of applets.
- Leave off the .class part when invoking the JVM.
- This is an old-fashioned command-line program. Java also supports GUI applications and web-browser hosted programs called applets.

Objects and Classes in JAVA

- An object simply represents any object that you may think of in the "real world“:
  - Tangible
  - Non tangible objects
- Objects are defined by creating a class. In fact, we organize ALL of our code into classes, each class being a separate definition of an object; each kind of object as its own class and in its own file (called a .java file).
- Before you can USE an object in your program, you MUST define it by creating its class. In fact, ALL objects MUST belong to a class of some kind.

Objects and Classes in JAVA (Cont.)

- To create a class, we simply make a new file and use the class keyword in JAVA followed by the classname and brace characters { }. Here are two examples for defining a Car and a BankAccount object:

```
public class Car {
        // write your code here
}

public class BankAccount {
        // write your code here
}
```

Objects and Classes in JAVA (Cont.)

- These two examples of classes would be saved individually into files called Car.java and BankAccount.java, respectively.
- Classes must have a name (i.e., a classname) which should:
  - be unique from other classes
  - start with an uppercase letter
  - represent a singular object, NOT plurality (i.e., Car...not Houses, Customer...not Customers)

Objects and Classes in JAVA (Cont.)

- To complete the definition of the class, we need to decide on two "things" that actually describe or define the Object:
- STATE (a.k.a. attributes or fields)
  - this is the information we want to keep for the object

    (e.g., color, size, amount, name, phoneNumber, etc...)
- BEHAVIOR (a.k.a. methods)
  - this is what we'd like to be able to do with the object (i.e., its capabilities)

    (e.g., walk, run, drive, sell, buy, transfer, computeTotal, open, close, etc...)
- So objects can be thought of simply as bundles of information as well as an instruction manual showing us how to use the object:

Objects and Classes in JAVA (Cont.)

- Here are some examples of objects and some of their possible state and behavior:

Object	State	Behavior
Person	name , age, sex, phone number	**get** the address, **change** phone number
Customer	name, address, purchase history	**make** purchase, **list** items bought, **return** item
BankAccount	owner, balance, account number	**withdraw**, **deposit**, **transfer**, **get** balance

States and Behaviors of a Class

- State is written usually right at the top of the class definition. Each part of the state (each attribute) has a type as well as its own name.
- Each behavior is written one after another just beneath the state.

```
public class ClassName {
  // These represent the state of the object
   type1    name1;
   type2    name2;
   ...
  // These represent the behaviour of the object
   type1 nameOfBehaviour1() {
      // code defining behaviour 1
   }
   type2 nameOfBehaviour2() {
      // code defining behaviour 2
   }
   ...
}
```

Why to define classes ?

- A class definition represents a place where all objects belonging to that class obtain their state and behavior.
  - this keeps us organized
  - code becomes modular and more pluggable
- A class is often used as a code sharing mechanism. So...objects can be shared between different programs.
  - the amount of code written is reduced

How to write an OO program ?

- Identify the Objects (i.e., Classes) that represent our problem which is to be solved (or simulated)
  - also identify the object characteristics (i.e., state and behavior)
  - you may want to start writing some basic class definitions (i.e., code in separate class files)
- Identify the inter-relationships between the Objects (i.e., how they interact)
  - you'll likely add some more behavior to your previously-defined objects
- Write testing code to create some objects and make them interact together.

animation

Trace a Program Execution_Example 2

```
public class ComputeAreaTest {
  /** Main method */
  public static void main(String[] args) {
    ComputeArea computeArea = new ComputeArea();
    computeArea.compute();
  }
}
```

Create a new object of the class ComputeArea

animation

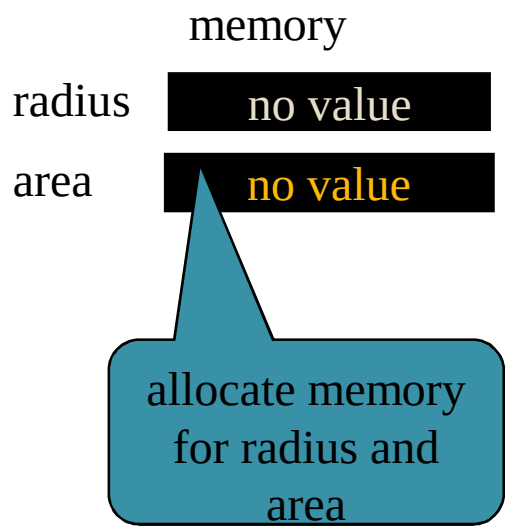
Trace a Program Execution_Example 2

```
public class ComputeArea {
  double radius;
  double area;

   public ComputeArea(){
    // Assign a radius
                radius = 20.0;
  }

   public void compute(){
   // Compute area
   area = radius * radius * 3.14159;

   // Display results
   System.out.println("The area for the circle of radius " +
radius + " is " + area);
   }
}
```



animation

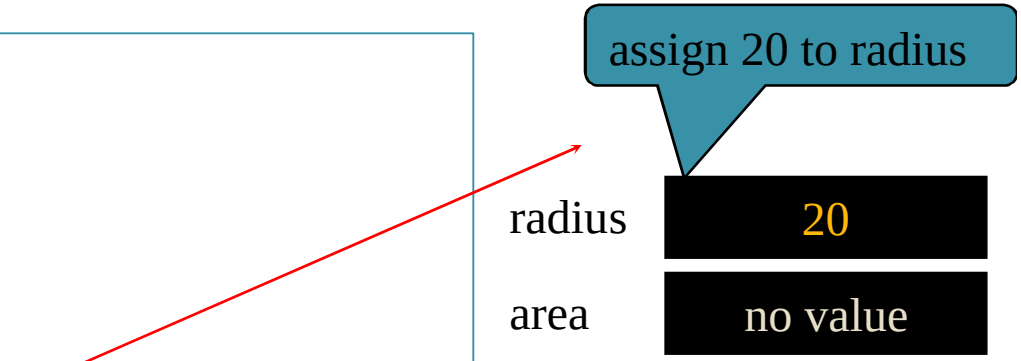
Trace a Program Execution_Example 2

```
public class ComputeArea {
  double radius;
   double area;

   public ComputeArea(){
    // Assign a radius
       radius = 20.0;
  }

   public void compute(){
   // Compute area
   area = radius * radius * 3.14159;

   // Display results
   System.out.println("The area for the circle of radius " +
     radius + " is " + area);
   }
}
```



animation

Trace a Program Execution_Example 2

```
public class ComputeAreaTest {
  /** Main method */
  public static void main(String[] args) {
    ComputeArea computeArea = new ComputeArea();
    computeArea.compute();
  }
}
```

animation

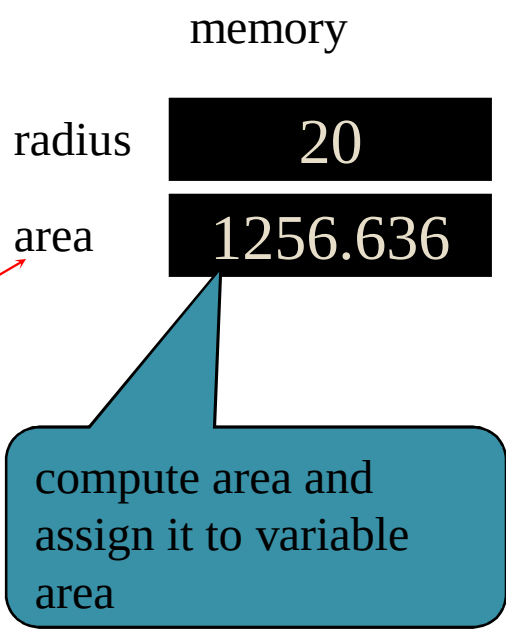
Trace a Program Execution_Example 2

```
public class ComputeArea {
   double radius;
    double area;

    public ComputeArea(){
     // Assign a radius
        radius = 20.0;
   }

    public void compute(){
    // Compute area
    area = radius * radius * 3.14159;

    // Display results
    System.out.println("The area for the circle of radius " +
      radius + " is " + area);
    }
}
```



animation

Trace a Program Execution_Example 2

```
public class ComputeArea {
   double radius;
    double area;

    public ComputeArea(){
     // Assign a radius
         radius = 20.0;
   }

    public void compute(){
    // Compute area
    area = radius * radius * 3.14159;

    // Display results
    System.out.println("The area for the circle of radius " +
      radius + " is " + area);
    }
}
```

memory

radius	20
area	1256.636

print a message to the console

animation

Trace a Program Execution_Example 2

```
public class ComputeAreaTest {
  /** Main method */
  public static void main(String[] args) {
    ComputeArea computeArea = new ComputeArea();
    computeArea.compute();
  }
}
```

How to use JAVA class library

- Why Using the JAVA class libraries ?
  - the classes are carefully written and are efficient.
  - it would be silly to write code that is already available to you.
- A all of JAVA's class libraries are arranged in packages.
- A package:
  - represents a collection of classes which logically fit together.
  - has a one-to-one correspondence with a directory (or file folder in windows).
  - allows developers to share groups of related classes.

Getting Input Using Scanner

- Create a Scanner object

```
Scanner scanner = new Scanner(System.in);
```

- Use the methods next(), nextByte(), nextShort(), nextInt(), nextLong(), nextFloat(), nextDouble(), or nextBoolean() to obtain to a string, byte, short, int, long, float, double, or boolean value.
- For example:

```
System.out.print("Enter a double value: ");
Scanner scanner = new Scanner(System.in);
double d = scanner.nextDouble();
```

How to use JAVA class library

- There are MANY standard packages in JAVA, each with many classes. For example:

java.lang	Basic classes and interfaces required by many JAVA programs. It is **automatically imported** into all programs.
java.util	Utility classes and interfaces such as Date/Time manipulations, random numbers, string manipulation, collections ...
java.io	Classes that enable programs to input and output data.
Java.text	Classes and interfaces for manipulating numbers, dates, characters and strings. Provides internationalization capabilities as well.

How to use JAVA class library (Cont.)

- Use the import keyword to make use of the standard java packages:
  - import <packageName>.*; This will allow us to use all code within the specified package.
  - Basically, the import statement is used to tell the compiler which package (i.e., directory) the class files are sitting in. You can always replace the * by a class name (where the class name is in the package).
  - Keep in mind though that the import statement does not load any classes, it merely instructs the compiler where to find them when you run your code.

Examples

- Example 1:

  HelloCalculatorTest.java

Programming Errors

- Syntax Errors
  - Detected by the compiler
- Runtime Errors
  - Causes the program to abort
- Logic Errors
  - Produces incorrect result

Syntax Errors

```
public class SyntaxError {
  public static void main(String[] args)
  {
    i = 30;
    System.out.println(i + 4);
  }
}
```

Runtime Errors

```
public class RuntimeError {
  public static void main(String[] args)
  {
    int i = 1 / 0;
  }
}
```

Logic Errors

```
public class LogicError {
  public static void main(String[] args)
  {
     int i;
     for (i=0; i<10; i++);
     {
          System.out.println("i is " + i);
     }
  }
}
```

Debugging

- Logic errors are called bugs. The process of finding and correcting errors is called debugging.
- A common approach to debugging is to use a combination of methods to narrow down to the part of the program where the bug is located.
  - You can hand-trace the program (i.e., catch errors by reading the program),
  - you can insert print statements in order to show the values of the variables or the execution flow of the program.
- This approach might work for a short, simple program. But for a large, complex program, the most effective approach for debugging is to use a debugger utility.

References

- Carleton University COMP 1005 and 1006:
  http://www.scs.carleton.ca/~lanthier/teaching/COMP1405/Notes/
  http://www.scs.carleton.ca/~lanthier/teaching/COMP1406/Notes/
- Armstrong Atlantic State University
  http://www.cs.armstrong.edu/liang/intro10e/
- Oracle Java tutorial:
  http://docs.oracle.com/javase/tutorial/
- MIT opencourseware
  https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-092-introduction-to-programming-in-java-january-iap-2010/