



SWE 205: Introduction to Software Engineering

Lecture 17

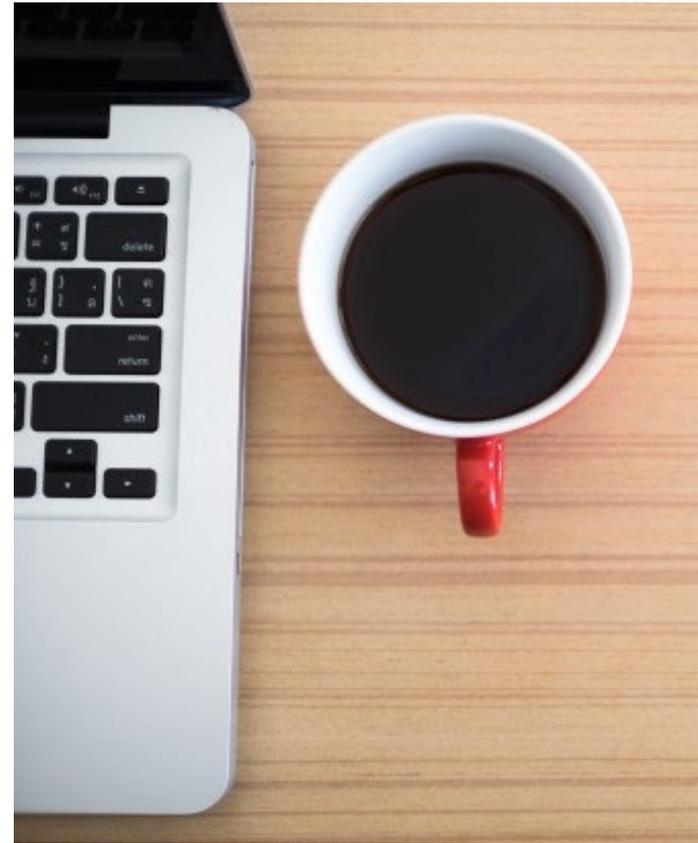
Software Construction

Course Topics

- Introduction
- Software Process Models
- Requirements Engineering
- Modeling
- Software Construction Techniques
- Testing
- Project Management
- Refactoring
- Ethical Issues

Lecture Objectives

- ✓ Software Construction
Fundamentals
 - Creating Understandable Code
 - Source Code Organization
 - Code Documentations



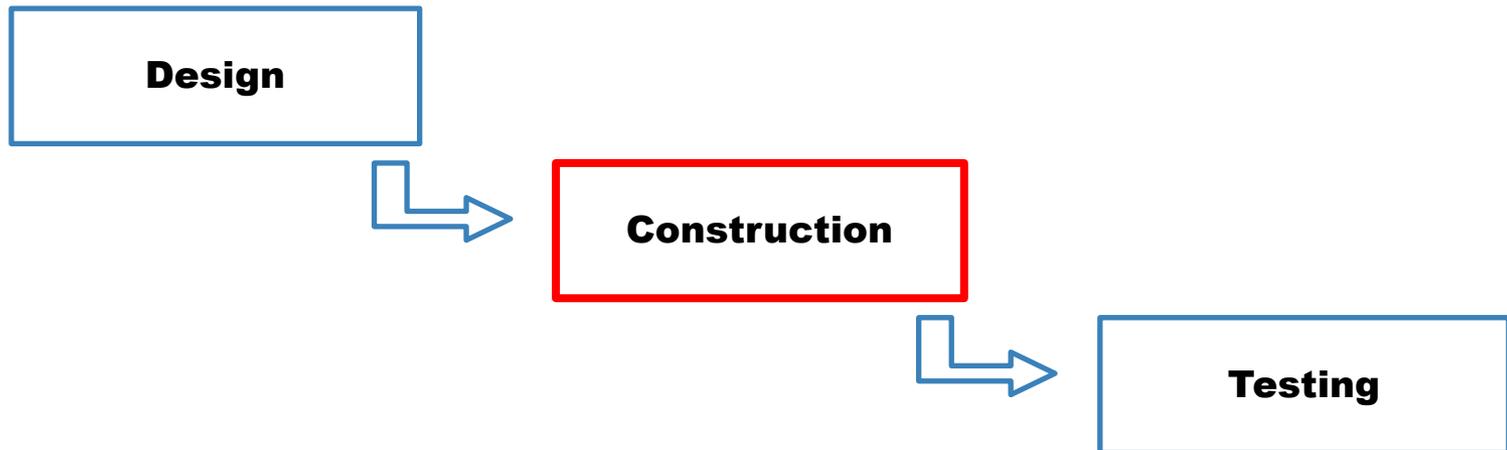
Software Construction



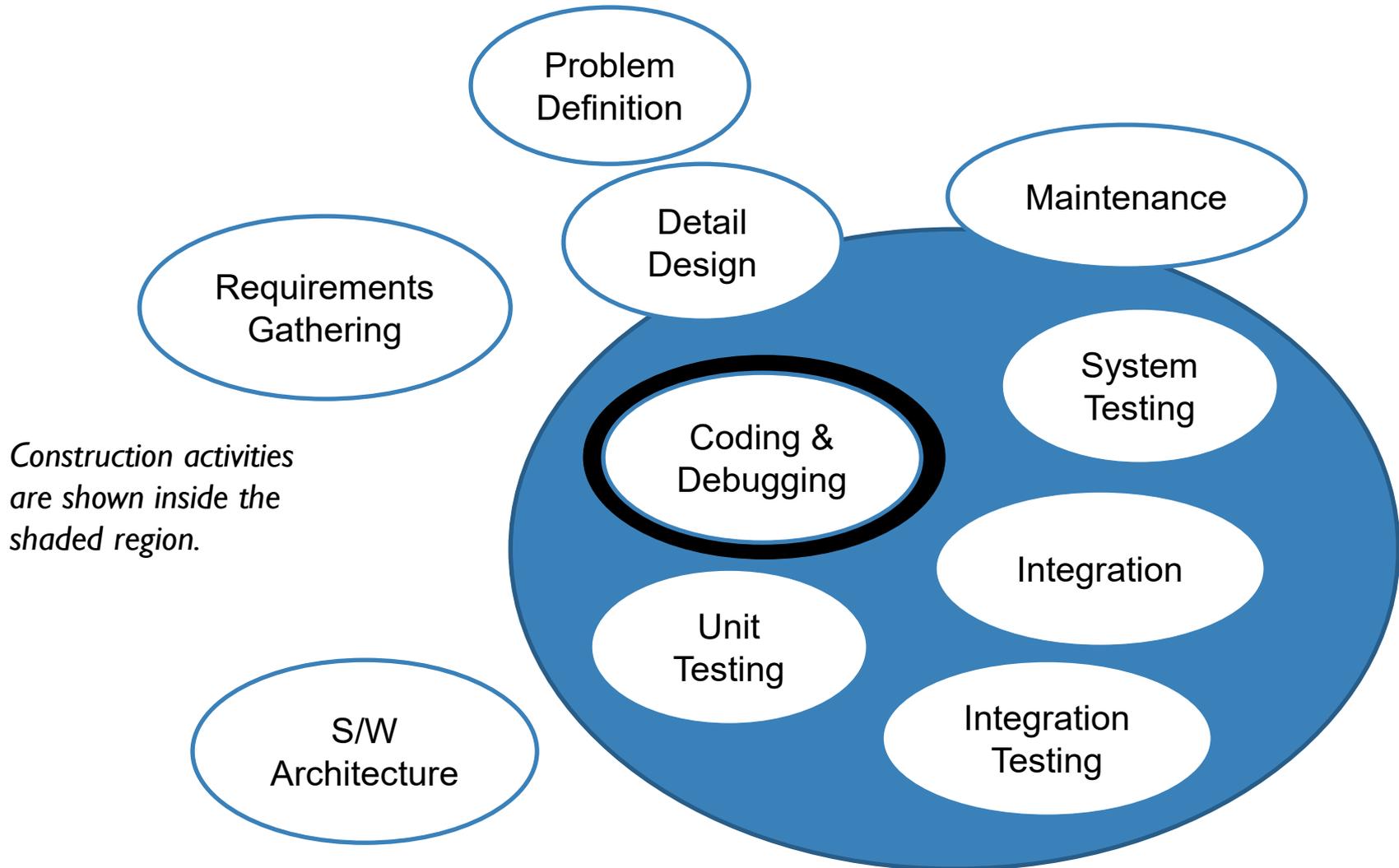
- What is Software Construction?
 - In general “construction” refers to the hands-on part of creating something.
 - Software Construction can be defined as detailed creation of **working, meaningful** software through a combination of coding, verification, unit testing, integration testing, and debugging.
 - Construction is also sometimes known as “coding” or “programming.”

Introduction

- Software construction closely tied to
 - Software design
 - Software testing



Software Construction Activities



Software Construction Fundamentals



- The fundamentals of software construction include:
 - *Minimizing complexity*
 - *Anticipating change*
 - *Constructing for verification*
 - *Standards in construction*

Minimizing Complexity



- Humans are severely limited in our ability to hold complex information in our working memories
 - As a result, minimizing complexity is one the of strongest drivers in software construction
 - Need to reduce complexity throughout the lifecycle
 - As functionality increases, so does complexity
- 

Minimizing Complexity



- Accomplished through use of standards

- Examples:
 - J2EE for complex, distributed Java applications
 - UML for modeling all aspects of complex systems
 - High-level programming languages such as C++ and Java
 - Source code formatting rules to aid readability

Anticipating Change



- Software changes over time

- Anticipation of change affect how software is constructed

- This can effect
 - Use of control structures
 - Handling of errors
 - Source code organization
 - Code documentation
 - Coding standards

Constructing for verification



- Construct software that allows bugs to be easily found and fixed

- Examples:
 - Enforce coding standards
 - Helps support code reviews
 - Unit testing
 - Organizing code to support automated testing
 - Restricted use of complex or hard-to-understand language structures

Reuse



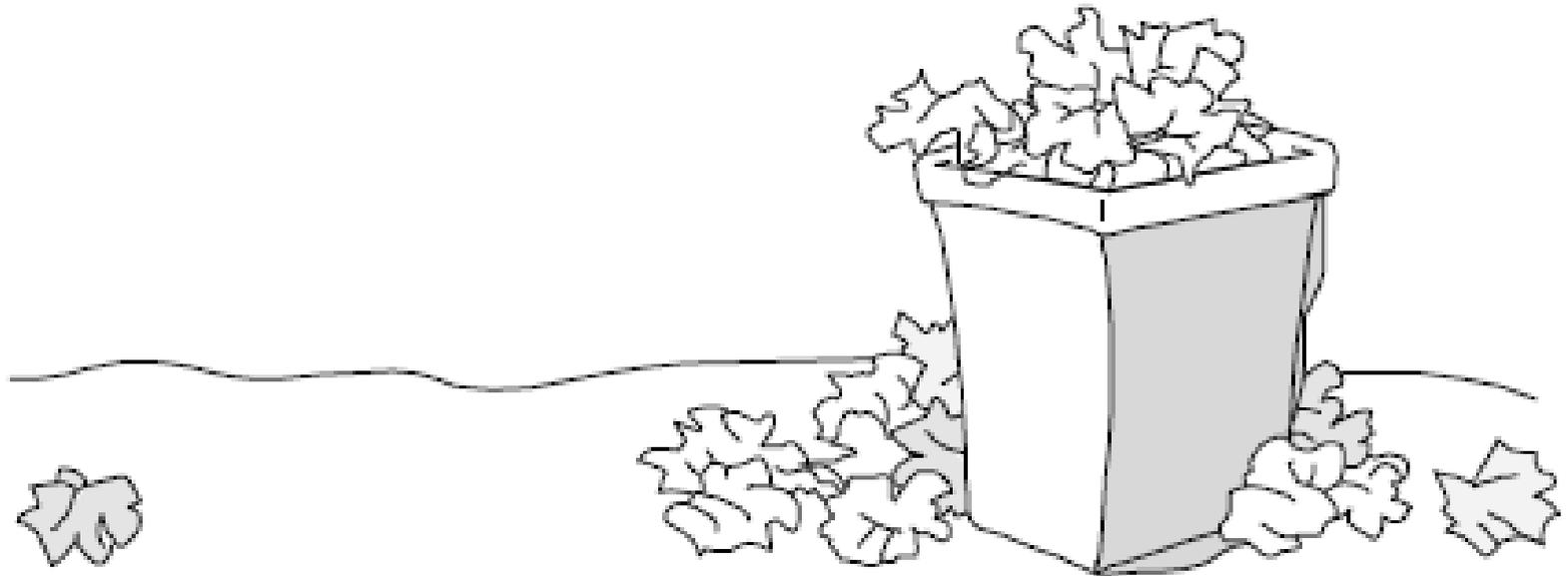
- In software construction, typical assets that are reused include libraries, modules, components, source code, and commercial off-the-shelf (COTS) assets.
 - Reuse is best practiced systematically, according to a well-defined, repeatable process.
 - Systematic reuse can enable significant software productivity, quality, and cost improvements.
- 

Standards in Construction

- Standards which directly affect construction issues include:
 - Programming languages
 - E.g. standards for languages like Java and C++
 - Communication methods
 - E.g. standards for document formats and contents
 - Platforms
 - E.g. programmer interface standards for operating system calls, J2EE
 - Tools
 - E.g. diagrammatic standards for notations like the Unified Modeling Language

Software Construction Metaphor

- *The letter-writing metaphor suggests that the software process relies on expensive trial and error rather than careful planning and design.*



1/3 Creating Understandable Code

- Naming

```
x = x - xx;  
xxx = aretha + SalesTax( aretha );  
x = x + LateFee( x1, x ) + xxx;  
x = x + Interest( x1, x );
```

- Source Code Layout

```
/* Use the insertion sort technique to sort the "data" array in ascending order.  
This routine assumes that data[ firstElement ] is not the first element in data and  
that data[ firstElement-1 ] can be accessed. */ public void InsertionSort( int[]  
data, int firstElement, int lastElement ) { /* Replace element at lower boundary  
with an element guaranteed to be first in a sorted list. */ int lowerBoundary =  
data[ firstElement-1 ]; data[ firstElement-1 ] = SORT_MIN; /* The elements in  
positions firstElement through sortBoundary-1 are always sorted. In each pass  
through the loop, sortBoundary is increased, and the element at the position of the  
new sortBoundary probably isn't in its sorted place in the array, so it's inserted  
into the proper place somewhere between firstElement and sortBoundary. */ for ( int  
sortBoundary = firstElement+1; sortBoundary <= lastElement; sortBoundary++ ) { int  
insertVal = data[ sortBoundary ]; int insertPos = sortBoundary; while ( insertVal <  
data[ insertPos-1 ] ) { data[ insertPos ] = data[ insertPos-1 ]; insertPos =  
insertPos-1; } data[ insertPos ] = insertVal; } /* Replace original lower-boundary  
element */ data[ firstElement-1 ] = lowerBoundary; }
```

Source Code Layout



■ Layout Techniques

- White Space
 - Use whitespacetoenhancereadability
- Grouping
 - Paragraph of code should contain statements that accomplish a single task and that are related to each other
- Blank lines
 - Separate unrelated statements from each other using blank line.
- Indentation
 - Use indentation to show the logical structure of a program.
- Parenthesis
 - Use parentheses to clarify expressions that involve more than two terms to add clarity.

Naming Mechanism

■ Previous Example with Meaningful Names

```
x = x - xx;  
xxx = aretha + SalesTax( aretha );  
x = x + LateFee( x1, x ) + xxx;  
x = x + Interest( x1, x );
```



```
balance = balance - lastPayment;  
monthlyTotal = NewPurchases + SalesTax( newPurchases );  
balance = balance + LateFee( customerID, balance ) + monthlyTotal;  
balance = balance + Interest( customerID, balance );
```

- Naming is a tedious job.

- But Naming Conventions help when
 - Multiple programmers are working on a project
 - You plan to turn a program over to another programmer for modifications and maintenance
 - Your program is so large that you can't hold the whole thing in your mind at once and must think about it in pieces
 - You have a lot of unusual terms that are common on a project and want to have standard terms or abbreviations to use in coding

Naming Mechanism

- Name fully and accurately describe the entity the variable represents.
 - A variable that contains the current interest rate is better named `rate` or `interestRate` than `r` or `x`.

- Follow Language-Specific Conventions
 - For example in Java
 - `i` and `j` are integer indexes
 - Constants are in ALL_CAPS separated by underscores
 - Class and interface names capitalize the first letter of each word, including the first—for example, `ClassOrInterfaceName`.
 - Variable and method names use lowercase for the first word, with the first letter of each following word capitalized—for example, `variableOrRoutineName`.

2/3: Source Code Organization



- Typically organized into statements, methods, classes and packages.
- Important questions is when to create a method, a class or a package.

2/3: Source Code Organization (cont.)



- When to create a Method?
 - Reduce complexity
 - Make a section of code readable
 - Avoid duplicate code
 - Improve performance

- When to create a Class?
 - You can hide implementation details
 - Changes don't affect the whole program
 - You don't have to pass data all over your program
 - You're able to work with real-world entities rather than with low-level implementation structures

3/3 Code Documentation

- Why people don't write comments?
 - They think their code is clearer than it could possibly be.
 - They think that other programmers are far more interested in their code than they really are.
 - They are lazy.
 - They are afraid someone else might figure out how their code works.

```
1 import java.awt.*;
2 import java.applet.*;
3 import java.awt.event.*;
4
5 public class Demo2Image extends Applet implements ActionListener {
6     Button btnMove;
7     private int imageLeft = 10; //stores location of left side of image
8     Image myImage; //Image object which will be drawn
9
10    public void init() {
11        setLayout(null);
12        btnMove = new Button("Move Image");
13        btnMove.setBounds(40,100,300,40);
14        add(btnMove);
15        btnMove.addActionListener(this);
16    }
17
18    public void paint(Graphics g) {
19        myImage = getImage(getCodeBase(), "box.gif" ); //loads the image
20        //box.gif MUST BE IN SAME FOLDER AS THE CLASS FILE (not the .java file)
21        g.drawImage(myImage,imageLeft,10,this);
22        //draws the image at (x = imageLeft, y = 10)
23        g.drawLine(0,0,imageLeft, 10);
24        //draws a line from origin to corner of box image
25    }
26
27    public void actionPerformed(ActionEvent event){
28        if (event.getSource() == btnMove) {
29            imageLeft = imageLeft + 10; //increments left side of image
30            repaint();
31        }
32    }
33 }
```

3/3 Code Documentation (cont.)



- Explain the code's intent or summarize what the code does, rather than just repeating the code
 - Avoid end line comments
 - They make the statement lengthier and intermingle with the code
 - Use end line comments only with data declarations
 - Comments should focus on why rather than how.
 - Avoid redundant, extraneous and self-indulgent comments.
 - Avoid abbreviations in comments
- 

3/3 Code Documentation (cont.)



- Use commenting style that allows comments to be easily modified
- Keep comments clear, correct and up to date.
- Finally – the beginning of a file, class and a routine should always be commented with its purpose.

Example of code comments !!



Code Documentation: javadoc

- javadoc.exe: documentation tools can be very handy

```
1  /**
2   * Returns an Image object that can then be painted on the screen.
3   * The url argument must specify an absolute {@link URL}. The name
4   * argument is a specifier that is relative to the url argument.
5   * <p>
6   * This method always returns immediately, whether or not the
7   * image exists. When this applet attempts to draw the image on
8   * the screen, the data will be loaded. The graphics primitives
9   * that draw the image will incrementally paint on the screen.
10
11  @param url  an absolute URL giving the base location of the image
12
13  @param name the location of the image, relative to the url argument
14
15  @return    the image at the specified URL
16
17  @see      Image
18  */
19  public Image getImage(URL url, String name) {
20      try {
21          return getImage(new URL(url, name));
22      } catch (MalformedURLException e) {
23          return null;
24      }
25  }
```



About 4,720,000 results (0.08 seconds)

[Advanced search](#)

Code Documentation: javadoc output

Image.htm

Class Image

```
java.lang.Object  
└─ Image
```

```
public class Image  
extends java.lang.Object
```

Method Detail

getImage

```
public Image getImage(URL url,  
                       java.lang.String name)
```

Returns an Image object that can then be painted on the screen. The url argument must specify an absolute URL. The name argument is a specifier that is relative to the url argument.

This method always returns immediately, whether or not the image exists. When this applet attempts to draw the image on the screen, the data will be loaded. The graphics primitives that draw the image will incrementally paint on the screen.

Parameters:

url - an absolute URL giving the base location of the image
name - the location of the image, relative to the url argument

Returns:

the image at the specified URL

See Also:

[Image](#)

Key Points



- Software Construction Techniques
 - Creating Understandable Code
 - Source Code Layout
 - Naming Mechanism
 - Source Code Organization
 - Methods
 - Classes
 - Code Documentation

References



- Ian Sommerville, “Software Engineering”, 10th Edition, Addison-Wesley, 2015.
- R. S. Pressman, Software Engineering: A Practitioner’s Approach, 10th Edition, McGraw-Hill, 2005.

Course Topics

- ~~Introduction~~
- ~~Software Process Models~~
- ~~Requirements Engineering~~
- ~~Modeling~~
- Software Construction Techniques
- Testing
- Project Management
- Refactoring
- Ethical Issues