



CS 146: Intro to Web Programming and Project Development

Instructor: Patrick Hill

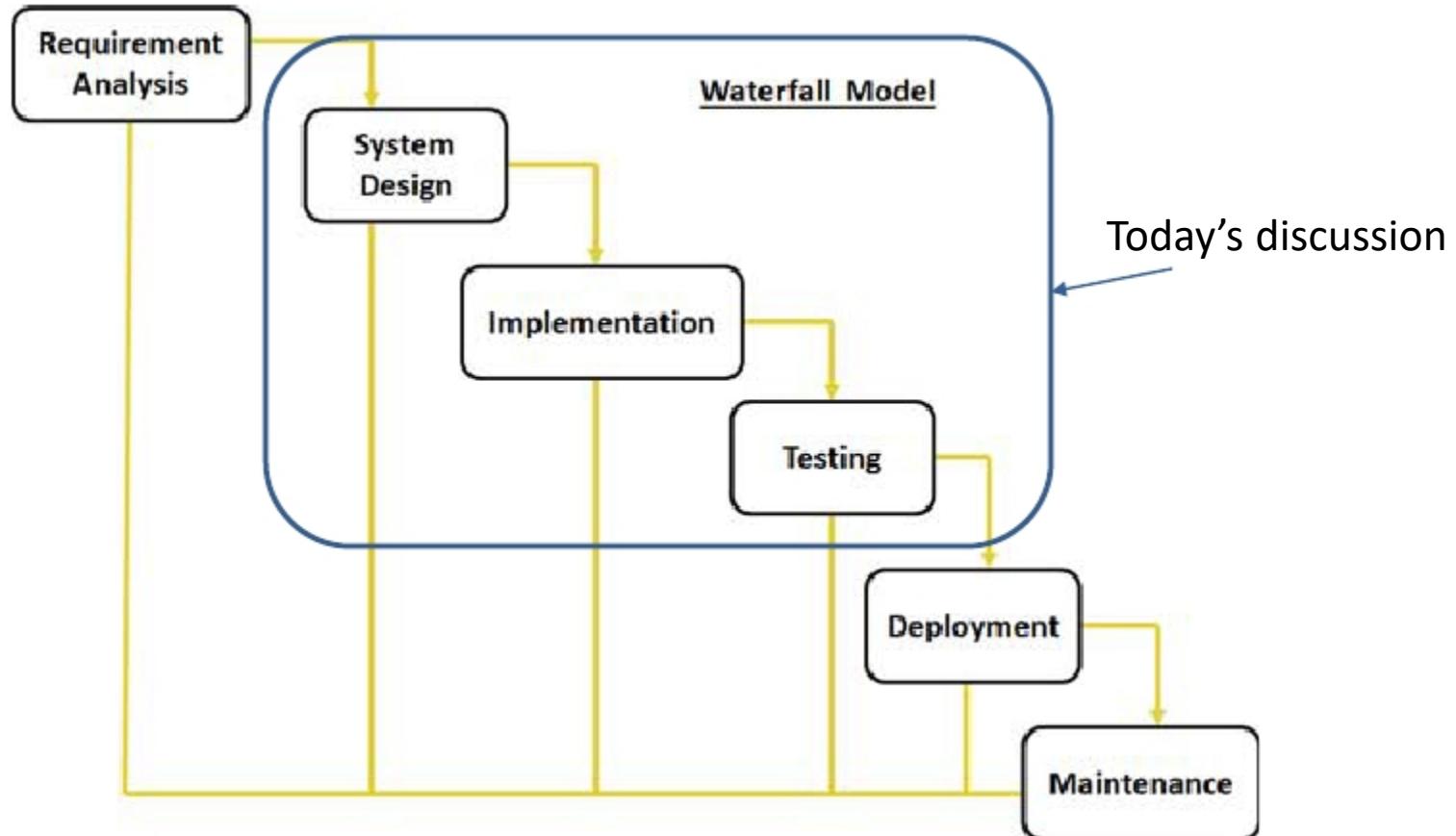
Email: phill@stevens.edu





Introduction to Software Engineering

Waterfall Model Visualized





System Design

- The system design phase is, as it sounds, where you create the design for your solution; the end product is one step away from becoming code
- We concentrate on how things will work, but algorithms are language-independent



Object-Oriented Design

- Based on 3 principles:
 - Encapsulation
 - Inheritance
 - Polymorphism



Encapsulation

- Encapsulation is the grouping of related ideas into one unit, which can thereafter be referred to by a single name
- It is sometimes, though incorrectly, referred to as information hiding, if the definition is very specific, as seen below:
 - Encapsulation: the technique of making the fields in a class private and providing access to the fields via public methods
 - With this definition, encapsulation allows abstracting things away from the user when they don't need to know how things work



Examples

- Is this encapsulation? If so, which definition applies?

```
public class Point  
{  
    public double x;  
    public double y;  
}
```



Examples

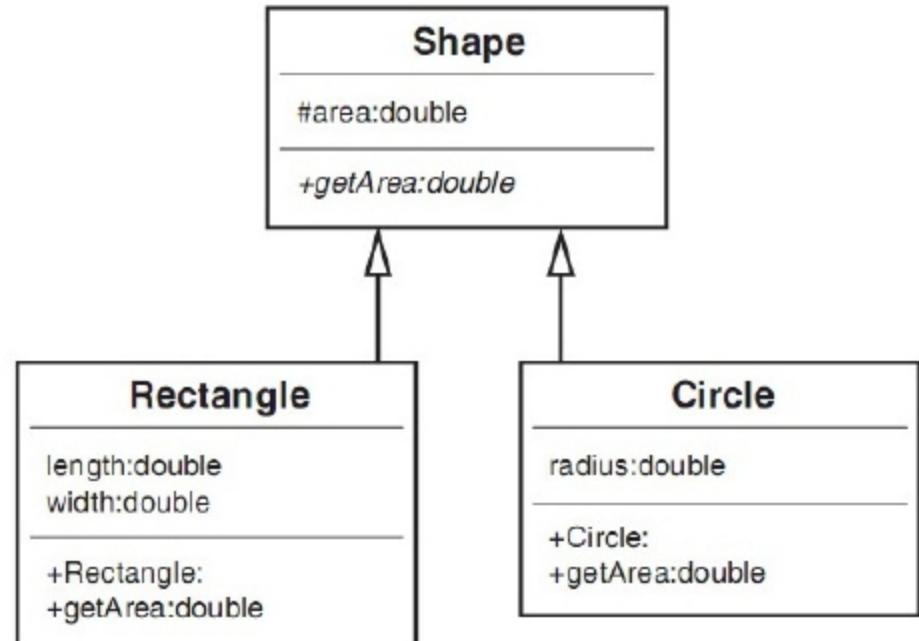
- Is this encapsulation? If so, which definition applies?

```
public interface Point  
{  
    double GetX();  
    double GetY();  
    void SetCartesian(double x, double y);  
    double GetR();  
    double GetTheta();  
    void SetPolar(double r, double theta);  
}
```

For more information, read **Clean Code: A Handbook of Agile Software Craftsmanship**.

Inheritance

- Inheritance: make subclasses that derive from other classes as a refinement of the parent class so you don't have to re-do something that was already done.



Shape UML diagram.

UML is short for *Unified Modeling Language*.

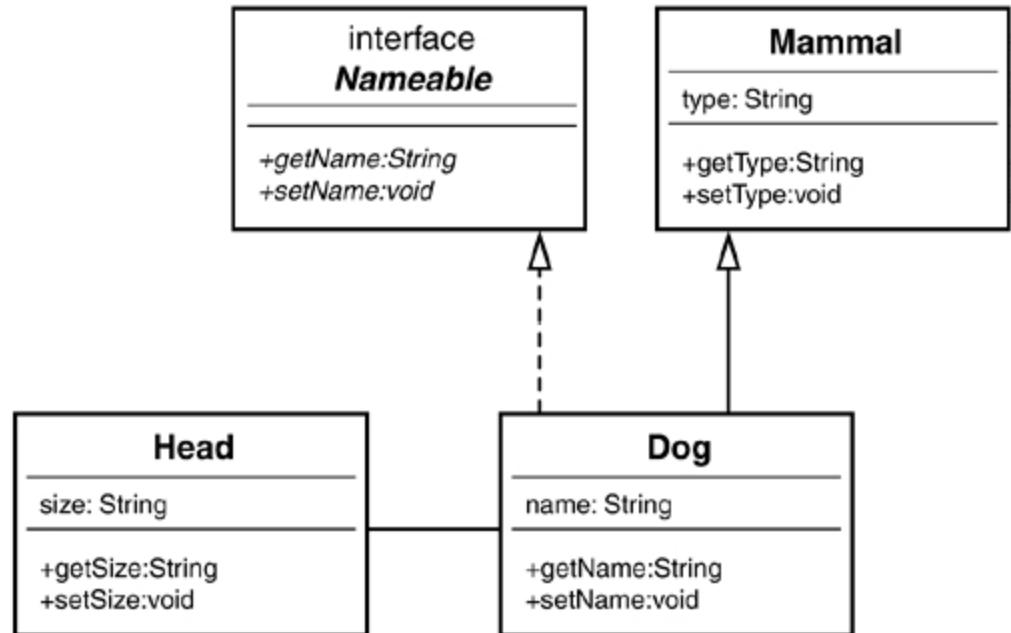


Polymorphism

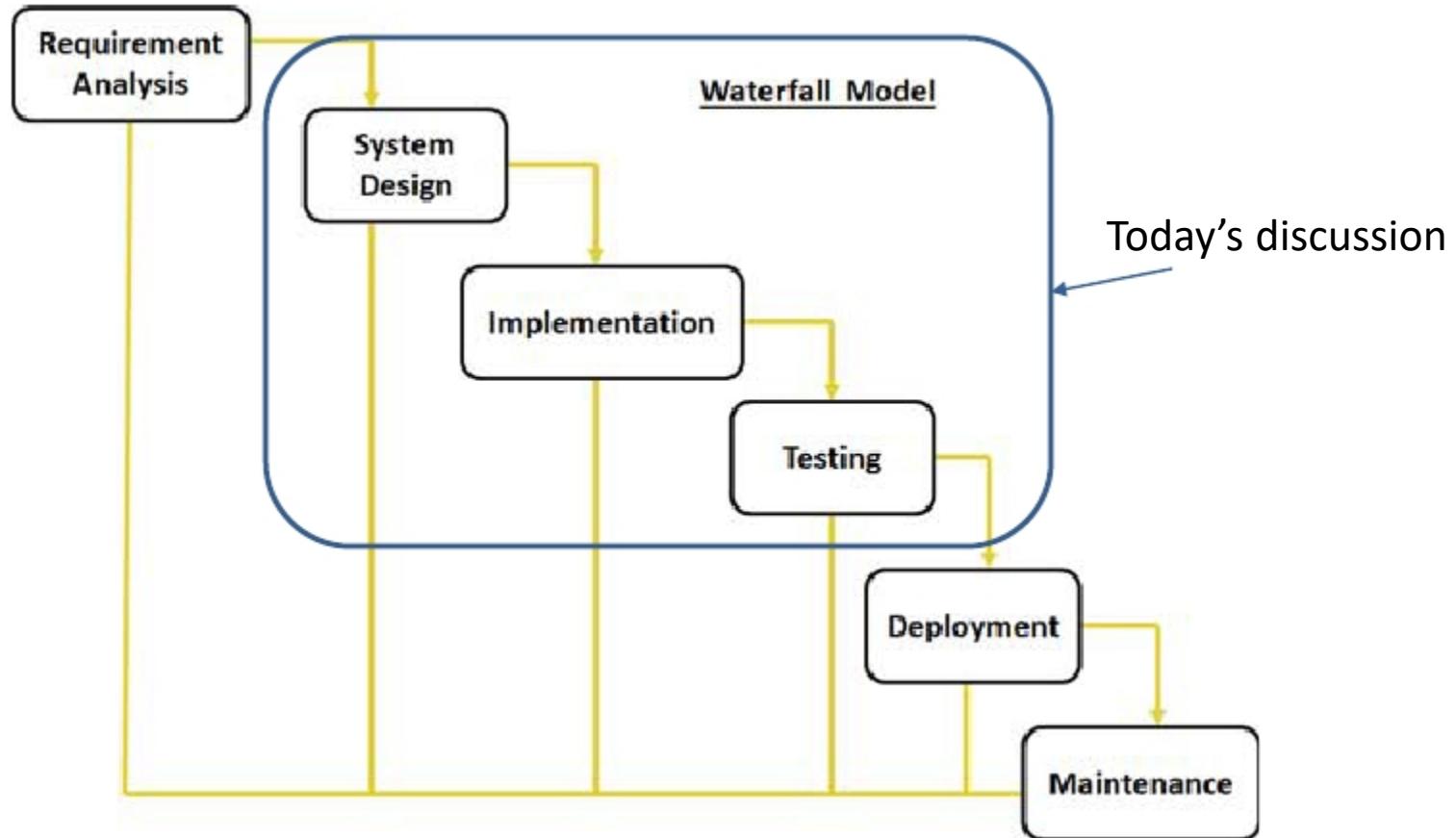
- Polymorphism is the ability of an object to take on many forms
- The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object
- In Java, polymorphism also means the ability to refer to a concrete class object through an interface reference

Examples of Polymorphism

- Nameable creature1 =
new Dog();
- Mammal creature2
new Dog();



Waterfall Model Visualized





Testing

- Testing: A means of determining if software contains bugs
- Debugging: Finding the bug in the code so it can be fixed
- Testing should start during the analysis/design phase, where you come up with a test plan
 - Ask yourself, "How are you going to test your code?"
 - Test cases should be written for all methods before you even write the method; this is sometimes referred to as "test driven programming", which is very common in industry



Three main categories of Errors

- Syntax Errors
- Run-time errors
- Logic errors



Syntax Errors

- You have a typo somewhere or wrote something the compiler didn't understand
- Easy to find because you just need to try to compile your code
- Syntax errors are UNACCEPTABLE, as it shows you never even had the chance to run your code
- The compiler actually tells you what's wrong with your code
- If you are overwhelmed by a multitude of errors, just look at them one at a time (top-most first), fix it, and compile again



Run-time Errors

- Run-time errors: Your program crashes during execution
- Reasonably easy to find with thorough testing, though **much** harder if code is multithreaded and the error is the result of a race condition
- Trace back where it happens to figure out what's wrong with the code; debuggers can be very helpful with this type of error



Logic Errors

- Hardest ones to fix
- Program doesn't crash but produces wrong output; it doesn't do what you intended
- May result in code that leads to a race condition and manifests itself as a run-time error somewhere else
- Hardest part is that the bug usually comes from your thought processes, making you think a wrong line of code is producing good output when it isn't



Purposes of Testing

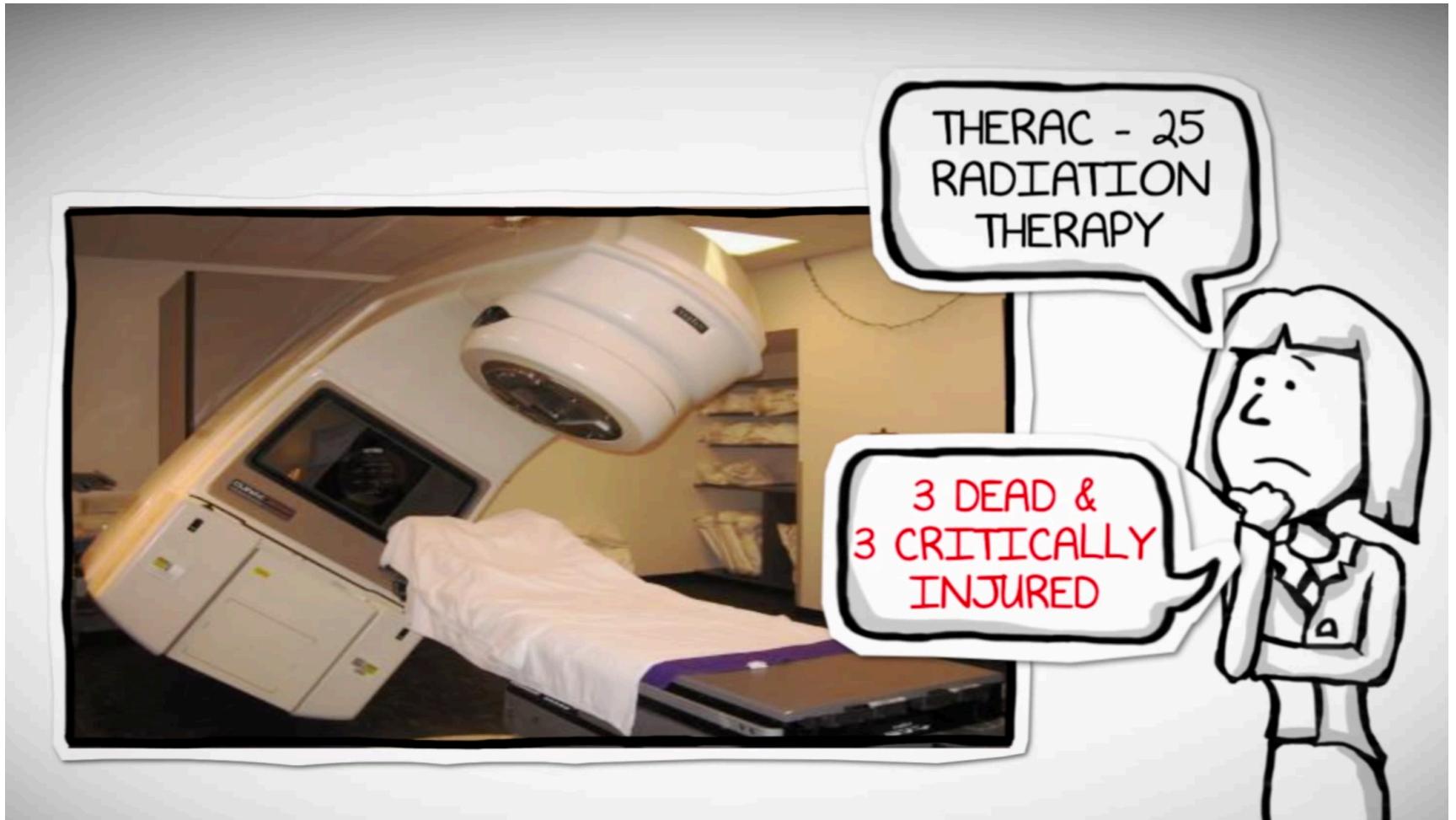
- To make sure the software
 - meets the requirements that guided its design and development,
 - responds correctly to all kinds of inputs,
 - performs its functions within an acceptable time,
 - is sufficiently usable,
 - can be installed and run in its intended environments, and
 - achieves the general result its stakeholders desire

Why is Testing Important?

China Airlines Flight 140, April 26th, 1994



Why is Testing Important?



Why is Testing Important?

April, 1999

Failed Satellite Launch



\$ 1.2 billion lost

Why is Testing Important?

May, 1996

U.S. Bank Accounts



823 Customers paid \$920 million

Paul Ehrlich:
“To err is human, but to really foul things up you need a computer.”



Exercise

- Consider a scenario where you write the code to move a file from folder A to folder B
 - How can you test this?
 - What can go wrong?



Exhaustive Search

- Suppose you have 15 input fields with 5 possible values each
- In order to test all the combinations you'd need
 - $5^{15} = 30,517,578,125$ combinations to test
- If you were to test all the possible combinations, the project execution time and cost would rise exponentially
- Exhaustive Testing is not possible
- Instead we need optimal amount of testing based on the Risk Assessment of the application



Exercise

Which operation is most likely to cause your operating system to fail?

A.

opening
Microsoft
Word

B.

open
Interne
Explorer

C.

opening 10 heavy
graphics
applications all at
the same time



Defect Clustering

- A small number of modules contain most of the defects detected
- With experience, you can identify such risky modules
- However, this can lead to another problem..



Pesticide Paradox

- If the same tests are repeated over and over again, eventually the same test cases will no longer find new bugs
- This is called the Pesticide Paradox
- To overcome this, the test cases needed to be regularly reviewed and revised, adding new and different test cases to help find more defects
- You can never claim that your code is BUG-FREE
- Why?
- Absence of error is a fallacy



7 Testing Principles

- Testing shows presence of defects
- Exhaustive Testing is impossible
- Early Testing
- Defect Clustering
- Pesticide Paradox
- Testing is context dependent
- Absence of errors is a fallacy