9/18/2018

**Introduction to Software Engineering**

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

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.

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

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

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