Introduction to Software Engineering

1: What is this about?

Joe Vanderwaart
What is Software Engineering?

• For our purposes: Useful Programming.
• Important considerations:
  – Scale (useful programs are big).
    • Working with other programmers.
    • Dividing programs into manageable pieces.
    • Building re-usable components.
  – Correctness (must work to be useful).
    • Robustness (tolerate errors).
    • Maintainability.
    • Security (sometimes).
Matters of Scale

CS1/CS2 Development Process.

- Clear, thorough spec.
- Thoughts not written down.
- Casual testing.
- Repeat until program works or deadline reached. Then, done forever.

How does it differ from real life?

- Read lab handout
- Think
- Code
- Test
Matters of Scale

CS1/CS2 Development Process.

• Clear, thorough spec.
• “Thoughts” not written down.
• Casual testing.
• Repeat until program works or deadline reached. Then, done forever.
More Realistic Process*

Real-world software engineers:
• Interact with customer to learn what’s needed.
  – Write it down: Requirements specification.
• Make plan for system.
  – Write it down: Design.
• Implement it.
  – Testing is key to quality.
• When customer’s satisfied, done…
• …until problem discovered or new feature needed.
  Then, start over.

*based on Fig. 11.1b, p. 257 of course text.
What This Course Covers

• Part 1: Implementing Program Modules
  – Methods, Classes.
  – Specifications and Program Correctness.
  – Proper use of exceptions, iterators, type hierarchies, generics.
  – Principled approaches to testing.

• Part 2: Building Systems
  – Design and Evaluation of Design.
I Assume You Know…

• Basics of Java.
  (And where to look up what you don’t know.)*
• What all these words mean:
  Object, class, instance variable, method, static, message, interface, inherit, implement, override, public, private, protected, package, type, subtype, statement, expression, variable, value, array, linked list, hash table, tree, library.

I also assume you have a CS account in good standing and can use the lab machines.

*Hint: http://java.sun.com/j2se/1.5.0/docs/api/
What I Will Ask of You

• Lecture attendance and readings.
• Weekly problem sets.
• “Midterm” “project”.
  Implement a program to my specification.
  Assigned “soon”, due April 23.
• “Final” project.
  Build a system of your own design.
  – Work in groups of 2-4 persons.
  – No dropping course once project groups formed (April 30).
  – Oral presentations last 1-2 days of class.

• (No exams)
Required Text

• Program Development in Java by Liskov.

• Source for required readings, exercises.

• Caveats:
  – Written for older versions of Java. (No generics, no Iterator-based for loop.)
  – Offers some advice I find misguided. (I’ll point it out when it comes up so we can discuss.)
Course Staff

• Instructor (me) (joev@cs)
  – Available in my office by appointment.
  – Regular hours TBA on my home page.
  www.cs.caltech.edu/~joev/

• TA: Henna Kermani (henna@its)
  – Office hours in Jorgensen lab TBA.
Web Page

www.cs.caltech.edu/courses/cs3/

• Reading and Homework Assignments
• Links to Supplemental Readings and other resources.
• Announcements
Overview
Implementing Program Modules

• Big systems must be built of smaller parts.
• Each programmer works on one at a time.
  – With many programmers, a distributed process.
  – Less time spent on communication, better.
  – Less sensitivity to each other’s schedules, better.
  – Method: Agree what each module must do; go separate ways to work out how.
• Parts should be re-usable when it makes sense.
  – In different projects. Hard, takes extra work.
  – In new releases of same project.
    In other words, should tolerate change.
My Slogan:

It’s not enough that your program (class, method,…) works now; what matters is how well it keeps working when conditions change.

I’ll repeat this again and again, with variations.
Abstraction

• Rules of language give meaning to code.
  – Know language + read code ⇒ know what code means.
  – This meaning is concrete.

• Program modules should also have intent, or abstract meaning.
  – Example 1:
    • Concrete: Function returns 0.15 times its argument.
    • Abstract: Given check total, function returns tip amount.
  – Example 2:
    • Concrete: This object consists of an array of objects and an integer.
    • Abstract: This object represents a stack.

“Tip” and “Stack” are abstract ideas, called abstractions.

Corresponding function or class implements the abstraction.
Abstraction

• Rules of language give meaning to code.
  – Know language + read code ⇒ know what code means.
  – This meaning is concrete.

• Program modules should also have intent, or abstract meaning.
  – Example 1:
    • Concrete: Function returns 0.15 times its argument.
    • Abstract: Given check total, function returns tip amount.

  – Example 2:
    • Concrete: This object consists of an array of objects and an integer.
    • Abstract: This object represents a stack.

“Tip” and “Stack” are abstract ideas, called abstractions.

Corresponding function or class implements the abstraction.
Abstraction

• Rules of language give meaning to code.
  – Know language + read code ⇒ know what code means.
  – This meaning is concrete.
• Program modules should also have intent, or abstract meaning.
  – Example 1:
    • Concrete: Function returns 0.15 times its argument.
    • Abstract: Given check total, function returns tip amount.
  – Example 2:
    • Concrete: This object consists of an array of objects and an integer.
    • Abstract: This object represents a stack.

“Tip” and “Stack” are abstract ideas, called abstractions.
Corresponding function or class implements the abstraction.
Abstraction

• Rules of language give meaning to code.
  – Know language + read code ⇒ know what code means.
  – This meaning is concrete.

• Program modules should also have intent, or abstract meaning.
  – Example 1:
    • Concrete: Function returns 0.15 times its argument.
    • Abstract: Given check total, function returns tip amount.
  – Example 2:
    • Concrete: This object consists of an array of objects and an integer.
    • Abstract: This object represents a stack.

• “Tip” and “Stack” are abstract ideas, called abstractions.

• Corresponding function or class implements the abstraction.
Abstractions in Software

• Every “part” of a program should implement some abstraction. Other parts of program (clients) can use the abstraction.  
  – Clients know about abstraction but not about implementation details.  
  – Different kinds of abstractions implemented by different language constructs (Chapters 3-8).

• Wisdom:  
  To build a system that works correctly, ensure that each module implements its abstraction correctly by using other modules in ways consistent with their abstract meanings.  
  – Sadly, this does not guarantee system will be totally correct.  
  – But it does help guide testing and debugging (Chapter 10).

• Design is figuring out and describing needed abstractions (Chapters 11-14).
Kinds of Abstractions

• Procedural Abstraction
  – Also known as a procedure.
  – Captures some “family” of computations.
    • Factorial function can compute 1!, 2!, 3!,…
    • Technically, this is an infinite number of potential computations!
  – Usually implemented by a static method.
Kinds of Abstractions

• Data Abstraction
  – Represents a (potentially infinite) set of objects.
  – Operations characterize behavior of objects.
  – *Language feature used for implementation?* Class.
Kinds of Abstractions

• Data Abstraction
  – Represents a (potentially infinite) set of objects.
  – Operations characterize behavior of objects.
  – Language feature used for implementation? Class.
Kinds of Abstraction

• Iteration Abstraction
  – Represents a way of examining a sequence of objects.
  – “Define your own looping construct.”
  – Representation involves objects of type `Iterator`.
  – No real support from Java language until version 5.

• Polymorphic Abstraction
  – Represents a “type-indexed family” of procedural or data abstractions.
  – E.g., the set `{IntegerList, StringList, DoubleArrayList,…}`
  – No support for this pre Java 5 either.
    • The book does it the bad old-fashioned way.
to be continued...