CSE 331 SOFTWARE DESIGN & IMPLEMENTATION

Alan Perlis ⓘ Epigrams ⓘ

It is easier to write an incorrect program than understand a correct one.

Most people find the concept of programming obvious, but the doing impossible.

It is easier to change the specification to fit the program than vice versa.

There are two ways to write error-free programs; only the third one works.

To understand a program you must become both the machine and the program.

Autumn 2011

Who are we?

Staff	
Women	3
Men	1
Undergraduate students	1
Graduate students	1
Alumni	1
Faculty	1
Beards	1
No beards	3

Students	
Women	12
Men	43
Sophomores	7
Juniors	30
Seniors♣	18
CS major	33
CompE major	15
Other major	7

♣Only three with any 400-level CSE courses

Two goals of software system building

Barry Boehm ⓘ

- Building the right system
  - Does the program meet the users' needs?
  - Determining if this is true is usually called *validation*
- Building the system right
  - Does the program meet the specification?
  - Determining if this is true is usually called *verification*

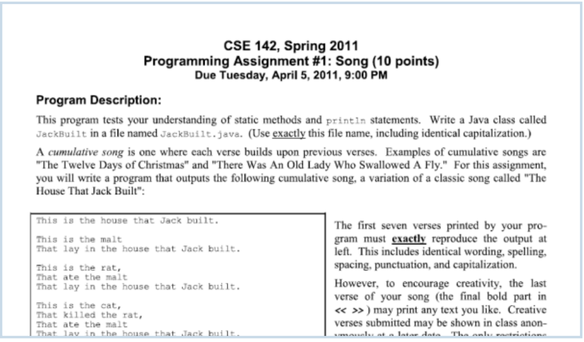
In CSE331, the second goal is *the* focus – that is, we focus (almost) *only* on creating a correctly functioning artifact

It can be surprisingly hard to specify, design, implement, test, debug and maintain even a simple program

"Does the program meet the specification?"

- You know what a *program* is – we'll focus on Java programs, but the ideas are much more general
- What is a *specification*?
  - "a detailed description or assessment of requirements, dimensions, materials, etc., as of a proposed building, machine, bridge, etc." [Dictionary.com Unabridged. Retrieved May 25, 2011]
  - It's the basis for a contract: "if you build something that does *X* [then we will 'pay you $19.55,' 'give you a 4.0,' etc.]" – *X* is the specification, defining how we can tell if something (for us, a program) *meets* the specification
  - Ambiguity in specifications is common, often inadvertent, sometimes necessary, and keeps lawyers wealthy

A familiar kind of specification



CSE 142, Spring 2011
Programming Assignment #1: Song (10 points)
Due Tuesday, April 5, 2011, 9:00 PM

Program Description:

This program tests your understanding of static methods and `println` statements. Write a Java class called `JackBuilt` in a file named `JackBuilt.java`. (Use exactly this file name, including identical capitalization.)

A *cumulative song* is one where each verse builds upon previous verses. Examples of cumulative songs are "The Twelve Days of Christmas" and "There Was An Old Lady Who Swallowed A Fly." For this assignment, you will write a program that outputs the following cumulative song, a variation of a classic song called "The House That Jack Built":

```
This is the house that Jack built.

This is the malt
That lay in the house that Jack built.

This is the rat,
That ate the malt
That lay in the house that Jack built.

This is the cat,
That killed the rat,
That ate the malt
That lay in the house that Jack built.
```

The first seven verses printed by your program must **exactly** reproduce the output at left. This includes identical wording, spelling, spacing, punctuation, and capitalization.

However, to encourage creativity, the last verse of your song (the final bold part in << >>) may print any text you like. Creative verses submitted may be shown in class anonymously at a later date. The only restrictions

Another familiar kind of spec

A CSE 143 assignment

```
// Interface Queue defines a set of operations for manipulating a FIFO
// (First In First Out) structure that can be used to store elements
// of type E.

public interface Queue<E> {
    // post: given value inserted at the end of the queue
    public void enqueue(E value);

    // pre : !isEmpty()
    // post: removes and returns the value at the front of the queue
    public E dequeue();

    // post: returns true if the queue is empty, false otherwise
    public boolean isEmpty();

    // post: returns the current number of elements in the queue
    public int size();
}
```

- ***pre-condition*** holds before execution
- ***post-condition*** holds after execution

Specification Jeopardy: `Hello World`

7

- Prints "Hello, World"
- Prints "Hello, World" without quotation marks
- Prints any string starting with "H"
- Prints any string with 12 characters in it
- Does anything
- Does anything as long as it terminates
- …

```
public static void main(String[] args) {
    System.out.println("Hello, World");
}
```

This and many later code examples will be partial – in this case, the class is omitted – but the full context should be clear.

- Every program meets (we'll usually say *satisfies*) an unbounded number of specifications
- It can be tricky to accurately specify the "right" amount of information – it takes experience

Specification Jeopardy: `Double`

8

```
public static void main(String[] args) {
  System.out.print(Integer.parseInt(args[0])*2);
}
```

- `// post: prints twice the first input argument`
- `// post: prints any integer`
- `// pre:  input argument ≥ 0`
  `// post: prints any non-negative integer`
- `// post: prints twice the first input argument OR`
  `         throws java.lang.NumberFormatException`
- `// pre:  input argument * 2 ≤ java.lang.integer.MAX_VALUE`
  `// post: prints twice the first input argument`
- `// post: if input argument * 2 ≤ java.lang.integer.MAX_VALUE`
  `//       prints twice the first input argument otherwise`
  `//       throws java.lang.NumberFormatException`
- …don't forget `MIN_VALUE` …

And more

9

```
public static void main(String[] args) {
  System.out.print(Integer.parseInt(args[0])*2);
}
```

- `// … no input argument?`
  - `Should it throw java.lang.ArrayIndexOutOfBoundsException`
- `// … non-integer input argument?`
  - `Should it throw java.lang.NumberFormatException`
- `// … more than one input argument?`

- Again, it's tricky, requiring careful case analysis – imaginable inputs? desired outputs? what is really intended? how is it intended to be used? … ?
- And, again, it takes experience (which should not be mistaken for intelligence)

Specifications: Q&A instead of A&Q

10

- Is it easier to simply write down specifications before the program exists?
- What would a post-condition be for a method that sorts an integer array of length **N** in non-decreasing order?
- *How do you precisely describe when an array is sorted?*

In small groups, spend 2-3 minutes sketching a sorting post-condition (any syntax is OK)

A flaw

11

- Most groups probably found a post-condition like

  `∀i,j∈[0,N-1] • i<j ⇒ A[i] ≤ A[j]`

  - That is, for any two elements in the array, the one with the lower index must not be greater than the one with the higher index
- But "undesired" satisfying programs are also allowed

  ```
  for(int i=0; i<N; i++){
    A[i] := i;
  }
  ```

  - Sorting means reordering the original elements, not simply creating a sorted array
- So, we need to add a clause to the post-condition that says

  `A is a permutation of A_orig`

"Incomplete" specifications: `sqrt`

12

-
  ```
  double sqrt(double x, double epsilon)
  pre:  x ≥ 0
  post: abs(return*return–x) < epsilon
  ```
- Perhaps this is the desired specification; perhaps not
  - What would `sqrt(81.0d,0.001d)` return?
- Might want the positive square root to be returned
- Or might want a *non-deterministic* specification that
  - allows a satisfying program to return different values for different invocations or
  - allows different satisfying programs to return different values

Getting to "correct"

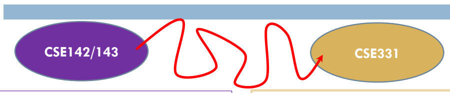
- There is no precise notion of a "correct" specification
  - However, there can be incorrect specifications – ones that cannot be implemented, ones that are inconsistent, and ones that are just bad …
- This is really a validation question, "Does the specification meet the needs of the users?"
- This is because there is no precise way to assess user needs – although there is a lot known about this, it is far beyond what we can cover in 331
- So we will focus on a precise notion – does a given program (implementation) satisfy a given specification?

Why is writing satisfying programs hard?

- Many ways to satisfy a specification – have to find and choose a "good" way?
  - "Goodness" is an ill-defined mix of customer needs, business needs, technologies, etc.
- Software systems are complex
  - Many difficult decisions and structures
  - Requires teams that effectively communicate and coordinate
- Customer needs evolve
  - Programs must change to meet spec changes
- …and more, much more…!

"Software entities are more complex for their size than perhaps any other human construct, because no two parts are alike… If they are, we make the two similar parts into one… In this respect software systems differ profoundly from computers, buildings, or automobiles, where repeated elements abound." Fred Brooks ⓘ "No Silver Bullet — Essence and Accidents of Software Engineering". *IEEE Computer* 20 (4): 10–19 (1987) ⓘ

The Tao (道) of CSE331

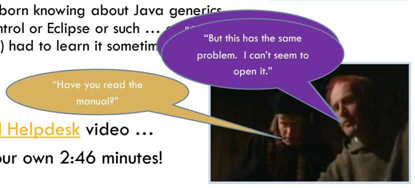


Programming (in Java)
- Control (loops, conditionals, methods, parameter passing, recursion, etc.)
- Variables
- Abstract data types (ADTs):
- Stacks, linked lists, …
- Interfaces, inheritance and encapsulation
- Basics of complexity and performance tradeoffs
- Using off-the-shelf components from Java Collections

Designing and implementing more realistic software (in Java, but more general)
- Abstraction and specification
- Writing, understanding and reasoning about code
- Program design and documentation: process and tools
- What makes a design good or bad?
- Pragmatic considerations
- Testing
- Debugging and defensive programming
- Software management issues

An observation

- Some of you are eyeballing others, worrying that "they know stuff I don't know"
- This is likely true **but**
  - You also know stuff they don't **and**
  - *Nobody* was born knowing about Java generics or version control or Eclipse or such … (who knows it) had to learn it sometime
- See Medieval Helpdesk video … watch it on your own 2:46 minutes!


"But this has the same problem. I can't seem to open it."
"Have you read the manual?"

If you are having trouble…

- "Ask" yourself what's going on – programming by permutation rarely succeeds, so *think* first!
- Look for information – it can be hard, but learning how to do this effectively is a great investment
- Ask others for help – course staff, friends, students in the class, etc.
- **The one *epic fail* is to stay stuck on something for a long time**

DON'T DO THIS! Really. I'm totally serious. Really.

In one of my first jobs, I had some bugs I simply couldn't find. After far too many hours (days), the president of the company sat me down and said, "OK, David, tell me why these bugs can't happen." I found them really quickly!

Performance

- How fast a program runs can be important – and we may at times talk about performance
- But it is not even close to the primary focus of 331 – *correctness is much more important*
- These quotations about performance are from people with extraordinary experience and insight

Michael Jackson ⓘ

Rule 1: Don't do it.

Rule 2 (for experts only): Don't do it yet.

Bill Wulf ⓘ

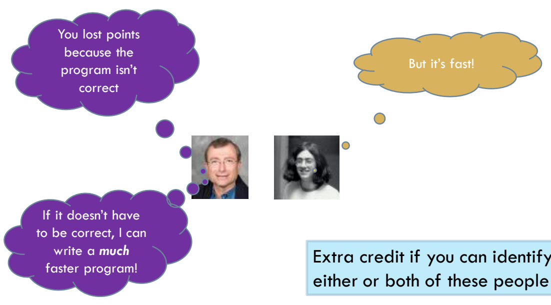
More computing sins are committed in the name of efficiency (without necessarily achieving it) than for any other single reason – including blind stupidity.

Don Knuth ⓘ

We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil.

An old conversation, oft-repeated

19


You lost points because the program isn't correct
But it's fast!
If it doesn't have to be correct, I can write a *much* faster program!

Extra credit if you can identify either or both of these people

Prerequisite: Java at the 142/143 level Examples

20

- Sharing
  - Distinction between == and `equals()`
    - Are two objects the same object, or do they have equal values?
    - And what does "equal" mean?
  - Aliasing (multiple references to the same object)
- Subtyping
  - Varieties: classes, interfaces
  - Inheritance and overriding
- Object-oriented dispatch
  - Expressions have a compile-time type
  - Objects/values have a run-time type
- **The first two assignments will largely focus on making sure you're (back) up to speed on this kind of material**

Logistics

21

- http://www.cs.washington.edu/cse331
  - There's useful information in there … if you can't find what you're looking for, make sure to ask!
  - Especially the calendar
    http://www.cs.washington.edu/education/courses/cse331/11au/calendar/calendar.html
- We'll also be using piazza.com for information and Q&A threads, etc.
- And Catalyst as well, although not for posting

Collaboration policy

22

- Discussion is permitted … indeed, encouraged!
- Representing someone else's work as your own is not permitted
- Familiarize yourselves with the CSE, COE and UW policies on academic honesty – we rely on them heavily – we will pursue cases of academic dishonesty
- If you have a question about what is allowed, ask!
- Please apply the Gilligan's rule
  - if you watch mindless TV (Hulu?) for 30-60 minutes after a discussion with classmates, and
  - you can still reproduce the materials from memory (no notes, no email, etc.), then
  - you can consider it your work/knowledge
- Loopholes are not loopholes – spirit of the law applies over the letter of the law

Next steps

23

- Assignment 0: on the web now, due Friday 11:59PM
  - Objective: get your Java environment configured and running, get a couple of tiny Java programs running, a survey, etc.
  - Warning: this can be quite complicated, with different operating systems (variants of Linux, Windows, Mac OS), different versions of Java, different versions of Eclipse, and more – we have some expertise, but sometimes it's hard for us, too
  - **Sections tomorrow: in the lab, with the staff there, bouncing from student-to-student to help you do this**
  - **TODAY: Make sure you can login to UW CSE machines! Both Linux and Windows (which can have different passwords!)**
    - Forgot CSENetID? Email support@cs (they reset it, you retrieve it at reception)
    - Forgot Windows password (but remember CSENetID)? Click here
- Assignment 1: on the web before Friday lecture; due next Wednesday 11:59PM
- Lectures: Specifications (F), testing/JUnit (M), equality (W)