CSE 331
Software Design & Implementation

Hal Perkins

Autumn 2012

Wrapup

10 weeks ago…

- We have 10 weeks to move to a level well above novice programmer:
  - Principled, systematic programming: What does it mean to get it right? How do we know when we get there? What are best practices for doing this?
  - Effective use of languages and tools: Java, IDEs, debuggers, JUnit, JavaDoc, svn
    - The principles are ultimately more important than the details
  - Larger programs

A huge thanks to the folks who made it work

Kellen Donohue

Wing Lam

James Okada

CSE 331 goals

Enable you to

- manage complexity
- ensure correctness
- write modest programs

  (modest by industry standards, that is….)

CSE 331 topics

Manage complexity:
- Abstraction
- Specification
- Modularity
- Program design & organization
  - OO design, dependences, design patterns, tradeoffs
- Subtyping
- Documentation

Ensure correctness:
- Reasoning
- Testing
- Debugging

Write programs:
- Practice and feedback
- Introduction to: tools (version control, debuggers), understanding libraries, software process, requirements, usability

Divide and conquer: Modularity, abstraction, specs

No one person can understand all of a realistic system

Modularity permits focusing on just one part

Abstraction enables ignoring detail

Specifications (and documentation) formally describe behavior

Reasoning relies on all three to understand/fix errors

Or to avoid them in the first place

Getting it right ahead of time

Design: predicting implications

- Example: understanding interconnections, module dependency diagrams

Understanding the strengths and weaknesses

- If you don’t understand a design, you can’t use it

Documentation matters!

Documentation

Everyone wants good documentation when using a system

Not everyone likes writing documentation

Documentation is often the most important part of a user interface

What's obvious to you may not be obvious to others

An undocumented software system has zero commercial value.

John Chapin

CTO of Vanu, Inc.

Testing

Helps you understand what you didn't understand while designing and implementing

A good test suite exercises each behavior

- Theory: revealing subdomains, proves correctness
- Practice: code coverage, value coverage, boundary values
- Practice: testing reveals errors, never proves correctness

A good test suite makes a developer fearless during maintenance

Maintenance

Maintenance accounts for most of the effort spent on a *successful* software system

- often 90% or more

A good design enables the system to adapt to new requirements while maintaining quality

- Think about the long term, but don't prematurely optimize

Good documentation enables others to understand the design

A good test suite greatly reduces the risks of changes

- And is a big part of the documentation/history of the project

Correctness

In the end, only correctness matters

- *Near*-correctness is often easy!
- Getting it right can be difficult

How to determine the goal?

- Requirements
- Design documents for the customer

How to increase the likelihood of achieving the goal?

- Unlikely without use of modularity, abstraction, specification, documentation, design, …
- Doing the job right is usually justified by return on investment (ROI)

How to verify that you achieved it?

- Testing
- Reasoning (formal or informal) helps!
- Use proofs and tools as appropriate

Working in a team

No one person can understand all of a realistic system

- Break the system into pieces
- Use modularity, abstraction, specification, documentation

Different points of view bring value

- Diversity is not just a “feel good” issue

Work effectively with others

- Sometimes challenging, usually worth it

Manage your resources effectively

- Time, people
- Engineering is about tradeoffs

Both technical and management contributions are critical

How CSE 331 fits together

Lectures: ideas	⇒ Assignments: get practice
Specifications	⇒ Design classes
Testing	⇒ Write tests
Subtyping	⇒ Write subclasses
Equality & identity	⇒ Override equals, use collections
Polymorphism	⇒ Write generic class
Design patterns	⇒ Larger designs
Reasoning, debugging	⇒ Correctness, testing
Events	⇒ GUIs
Usability, teamwork	⇒ (For fun and for future use)

What you have learned in CSE 331

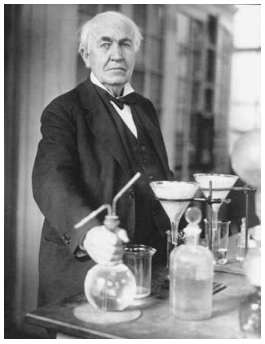
Compare your skills today to 3 months ago

- Theory: abstraction, specification, design
- Practice: implementation, testing
- Theory & practice: correctness
- Bottom line: Much of what we've done would be easy for you today
  - This is a measure of how much you have learned

There is no such thing as a "born" programmer!

Your next project can be more ambitious

> Genius is 1% inspiration and 99% perspiration.
> Thomas A. Edison



What you will learn later

Your next project can be much more ambitious

Know your limits

- Be humble (reality helps you with this)

You will continue to learn

- Building interesting systems is never easy
  - Like any worthwhile endeavor
- Practice is a good teacher
  - Requires thoughtful introspection
  - Don’t learn *only* by trial and error!

What comes next?

Classes

- CSE 403 Software Engineering
  - Focuses more on requirements, sofware lifecycle, teamwork
- Capstone projects
- Any class that requires software design and implementation

Research

- In software engineering & programming systems
- In any topic that involves software

Having an impact on the world

- Jobs (and job interviews)
- Larger programming projects

The purpose of computing is insight, not numbers.
Richard W. Hamming
Numerical Methods for Scientists and Engineers

Go forth and conquer

System building is fun!

- It's even more fun when you build it successfully

Pay attention to what matters

- Use the techniques and tools of CSE 331 effectively