CS601 Introduction

Terence Parr

Problem statement

"What were the lessons I learned from so many years of intensive work on the practical problem of setting type by computer? One of the most important lessons, perhaps, is the fact that SOFTWARE IS HARD."

--**Donald Knuth**, Keynote address to 11th World Computer Congress (IFIP Congress 89)

For the most part, however, writing programs as a sole contributor is not the problem.

Large projects are the problem

- Very few programs of significance can be written by one person
- The communication and complexity of software explodes exponentially with the number of people / lines of code.
- Programmers make assumptions
(mars lander metric vs English units)
- Or miss assumptions (ariane 5 rocket floating point exception thrown)
- Unexpected things can happen (patriot missile 8-bit counter overflow)

Sad history of software failures

From: http://www.economist.com/node/3423238

- Sept 2004: Air traffic control in California shut down, leading to some mid air encounters. A glitch meant the computers had to be rebooted every 30 days...and somebody forgot
- America's IRS (tax collectors) computer overhaul failed completely in 1997, wasting $4 billion
- New ``Obamacare'' website was a disaster
- Estimates:
  - "30% of all software projects are canceled
  - nearly half come in over budget
  - 60% are considered failures by the organizations that initiated them
  - 9 out of 10 came in late."

https://www.youtube.com/watch?v=-kHa3WNerjU

Who's fault is it?

- At some level, it's us programmers and our managers/leaders
- That doesn't mean we are failures; the problem is just extremely complicated
- Complexity goes up with: lines of code, number of classes, number of packages number of dependencies between them, number of authors, the nature of the task.
- We often apply the wrong design / development strategy for a task
- And yet...Apple (usually) produces great software so it's possible

Why do projects fail so often?

http://spectrum.ieee.org/computing/software/why-software-fails

- Unrealistic or unarticulated project goals
- Inaccurate estimates of needed resources
- Badly defined system requirements
- Poor reporting of the project's status
- Unmanaged risks
- Poor communication among customers, developers, and users
- Use of immature technology
- Sloppy development practices
- Poor project management
- Stakeholder politics

Size of software projects

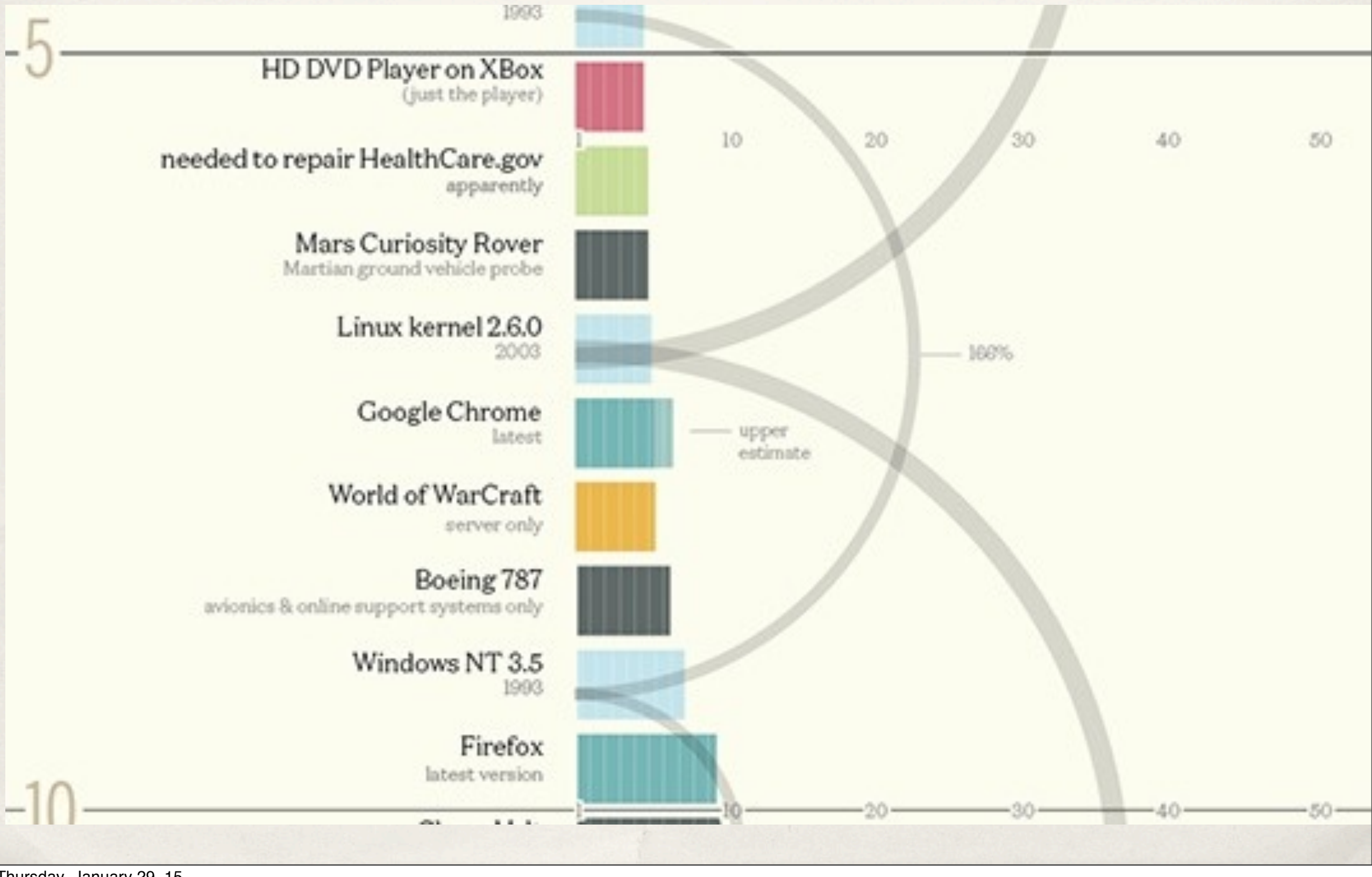
http://www.fastcodesign.com/3021256/infographic-of-the-day/infographic-how-many-lines-of-code-is-your-favorite-app#1

1M lines code is 20,000 printed pages of text @50 lines/page
That's 40 reams at 2" = 80" or 6.6'

- Unix v1.0 (1971): maybe 10k lines
- My ANTLR project: ~50k lines
- Photoshop v1.0 (1990): about 100k lines
- Space shuttle ~400k lines

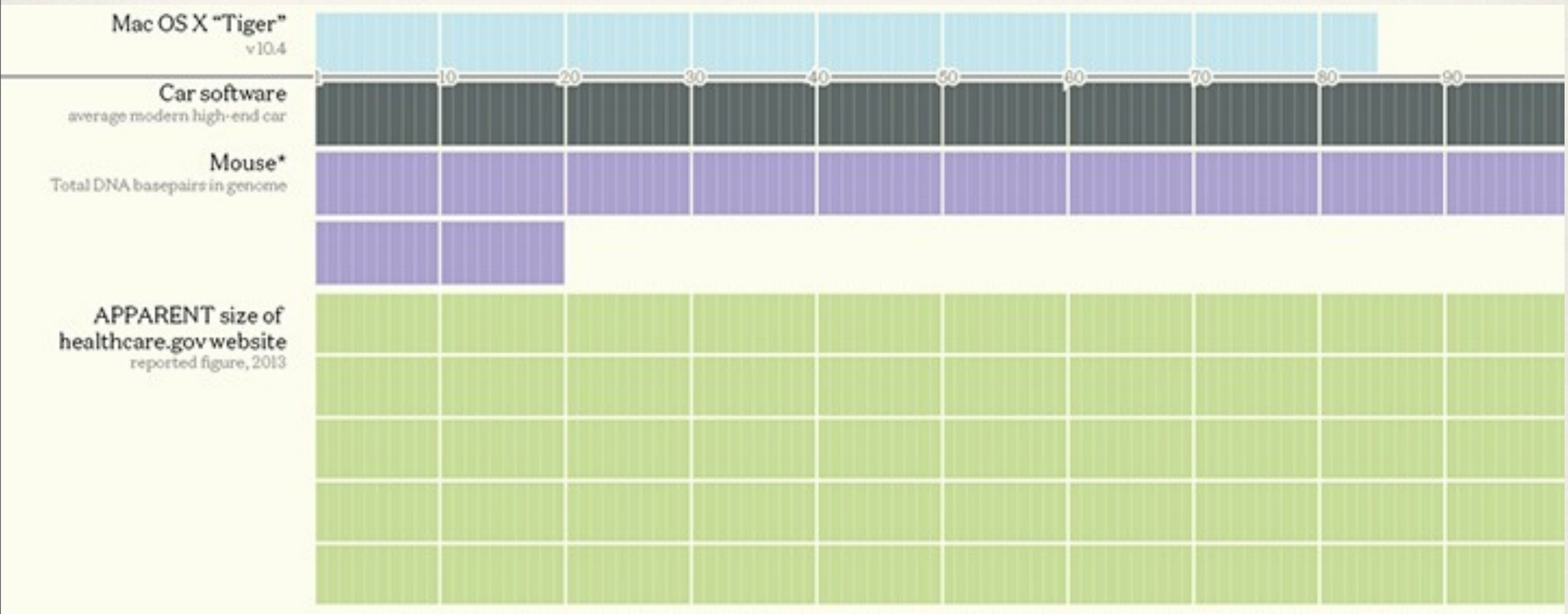
More sizes

http://www.fastcodesign.com/3021256/infographic-of-the-day/infographic-how-many-lines-of-code-is-your-favorite-app#1



More sizes Cont'd

http://www.fastcodesign.com/3021256/infographic-of-the-day/infographic-how-many-lines-of-code-is-your-favorite-app#1



Is software development a science, engineering, art, or what?

- All 3. We will discuss this in detail during the course. See my essay: http://parrt.cs.usfca.edu/doc/software-not-engineering.html

- In a nutshell, debugging requires the scientific method but software development typically requires management techniques, technical skills, cost estimation, rules of thumb, etc... like engineering plus creative solutions like an artist or architect.

What methods have we tried?

- Just start coding and see what happens
- Waterfall (Sequential)
- Iterative and Incremental development
- Spiral (Evaluate risk at each incremental milestone)
- Unified Process
- Extreme
- Agile
- Modeling languages, frameworks

Bits and pieces of each are similar; there are infinite variations

What do the methods have in common? Lifecycle stages

- Problem definition
- Requirements specification
- Design
- Coding
- Integration
- Testing
- Deployment
- Maintenance

Mix-and-match and loop to create your own methodology

Evaluation: risk reduction, satisfying customer, quality, visible progress, cost control, predictability

Choosing a methodology

From Code Complete second edition

Type	Apps	Model
Business systems	Payroll, website, inventory, games	Agile
Mission-critical	Embedded software, website, tools, services	Staged, spiral
Life-critical	Avionics, embedded, medical, OS	Staged, spiral

Choosing a methodology Cont'd

From Code Complete second edition

Type	Planning/Mgmt	Requirements	Design
Business systems	Incremental, informal change control	Informal requirements	Design, coding combined
Mission-critical	Basic up-front planning, formal change control	Semi-formal requirements	Arch design, informal detailed design
Life-critical	Extensive up-front planning, testing/QA, rigorous change control	Formal req, req inspections	Arch design, formal design, inspections

Sequential methods when...

From Code Complete second edition

- Requirements are fairly stable
- Design a straightforward and fairly well understood
- Development team familiar with application area
- Project contains little risk
- Cost of changing requirements design and code downstream is high

Iterative methods when...

From Code Complete second edition

- Requirements not well understood or expected to be unstable
- Design is complex, challenging, or both
- Developer team unfamiliar with application area
- Project contains a lot of risk, and uncertainty
- Long-term predictability not important
- Cost changing requirements, design, code likely to be low

Architecture and coding

- We will discuss all of this in detail during the course
- We will discuss and use lots of tools for programmer productivity
- And even discuss choice of language
- We discuss creating quality, clean, efficient code

What about design artifacts?

- Not *usually* a big fan: UML, Structure Charts, Petri Nets, State Transition Diagrams, flow charts, and decision tables

- More a fan of good methodology / process / strategy

- And good tools: IDEs, debuggers, profilers, performance monitoring, revision control, continuous integration, bug trackers, code reviewers, static analysis for finding bugs

Starting day 2...

- We'll start out by learning some tools
- Then jump into object-oriented programming concepts
- And move on to an in-depth discussion of threading and concurrent programming
- In general we will learn lots of technology
- And then shift gears to focus on methodology, architecture, and coding principles