CS 169
Software Engineering

Armando Fox, David Patterson,
and Koushik Sen

Spring 2012

Outline

- Class Organization (AF)
- Engineering SW is Different from HW
  (Book Sections 1.1-1.2 or §1.1-§1.2)
- Development Processes: Waterfall vs. Agile (§1.3)
- Assurance (§1.4)
- Productivity (§1.5)
- Software as a Service (§1.6) if time permits
- Service Oriented Architecture (§1.7) if time permits
- Cloud Computing (§1.8) if time permits

Course Goals

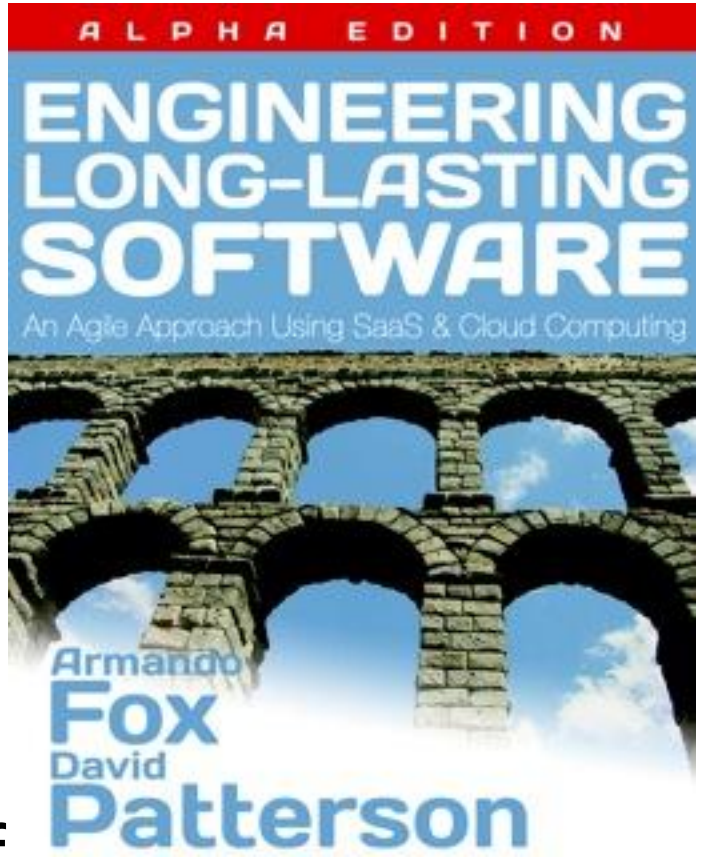
- Understand new challenges, opportunities, and open problems of SaaS relative to SWS (shrink-wrapped software)
- Take a SaaS project from conception to public deployment
  - Server side: Ruby on Rails
  - Client side: HTML, CSS, AJAX
  - Deploy using cloud computing
- Can be “connected” app (eg Facebook) or smartphone app (using HTML/AJAX/CSS)

Prereqs & course format

- 4 units, letter grade(see homepage for breakdown)
- These are real prereqs: CS61[ABC] or equivalents
- Format
  - Before lecture: reading, and sometimes online self-test
  - In lecture: put reading in context
  - After lecture: 6 homeworks, for hands-on practice; section for additional detail & worked examples
- 5 30-minute quizzes (every 2 weeks); no final
- Required “2-pizza team” project
  - Design, develop, deploy to “production ISP” (Heroku)
  - Outsiders invited (VC’s, etc) to final demos
  - Many projects have continued after the class

Textbook

- http://saasbook.info
- Print & Kindle ebook available
- 2-3 new chapters will be released during semester
- Kindle ebooks will get free updates thru Fall 2012
- We'll distribute hardcopy of additional chapters to you



Online SaaS Course

- First 5 weeks of on-campus course
- We will use their autograding technology for grading CS 169 homeworks
- Available 5 weeks after our course starts
- Please delay questions until end of “segment”, which is typically 6 to 10 minutes

Programming Homeworks

- Some done on your own, others in pairs
- Due Friday Midnight (see Syllabus)
  - Late policy: ¾ credit if 1 day late, ½ credit if 2 days late, 0 if later
- Can be done on your own computer or Amazon EC2
  - Download VirtualBox (for Mac, Win, Linux) and deploy "bookware VM" image (instructions on saasbook.info)
  - OR, deploy & use VM image on Amazon EC2 (instructions coming soon to saasbook.info) - http://aws.amazon.com/free/
  - Unsupported: install ALL courseware on your own computer (Ruby 1.9.2, Rails 3.1, lots of libraries...) – see saasbook.info for details
- Later HW's and Project will be deployed using cloud computing (details TBA)

Course Organization

- Grading
  - 1/3 - six homeworks
  - 1/3 - five quizzes (30 mins, in-class)
  - 1/3 - project
  - Discretionary bonus points during final grading: Participation and Altruism
- A typical week
  - Tue/Thu: lecture, office hours
  - Fri 11:59pm: homework due
  - Mon: section—discuss HW, review for quizzes
- Use Piazza for questions/discussion

YOUR BRAIN ON COMPUTERS; Hooked on Gadgets, and Paying a Mental Price

NY Times, June 7, 2010, by Matt Richtel

SAN FRANCISCO -- When one of the most important e-mail messages of his life landed in his in-box a few years ago, Kord Campbell overlooked it.

Not just for a day or two, but 12 days. He finally saw it while sifting through old messages: a big company wanted to buy his Internet start-up.

''I stood up from my desk and said, 'Oh my God, oh my God, oh my God,' '' Mr. Campbell said. ''It's kind of hard to miss an e-mail like that, but I did.''

The message had slipped by him amid an electronic flood: two computer screens alive with e-mail, instant messages, online chats, a Web browser and the computer code he was writing.

While he managed to salvage the $1.3 million deal after apologizing to his suitor, Mr. Campbell continues to struggle with the effects of the deluge of data. Even after he unplugs, he craves the stimulation he gets from his electronic gadgets. He forgets things like dinner plans, and he has trouble focusing on his family. His wife, Brenda, complains, ''It seems like he can no longer be fully in the moment.''

This is your brain on computers.

Scientists say juggling e-mail, phone calls and other incoming information can change how people think and behave. They say our ability to focus is being undermined by bursts of information.

These play to a primitive impulse to respond to immediate opportunities and threats. The stimulation provokes excitement -- a dopamine squirt -- that researchers say can be addictive. In its absence, people feel bored.

The resulting distractions can have deadly consequences, as when cellphone-wielding drivers and train engineers cause wrecks. And for millions of people like Mr. Campbell, these urges can inflict nicks and cuts on creativity and deep thought, interrupting work and family life.

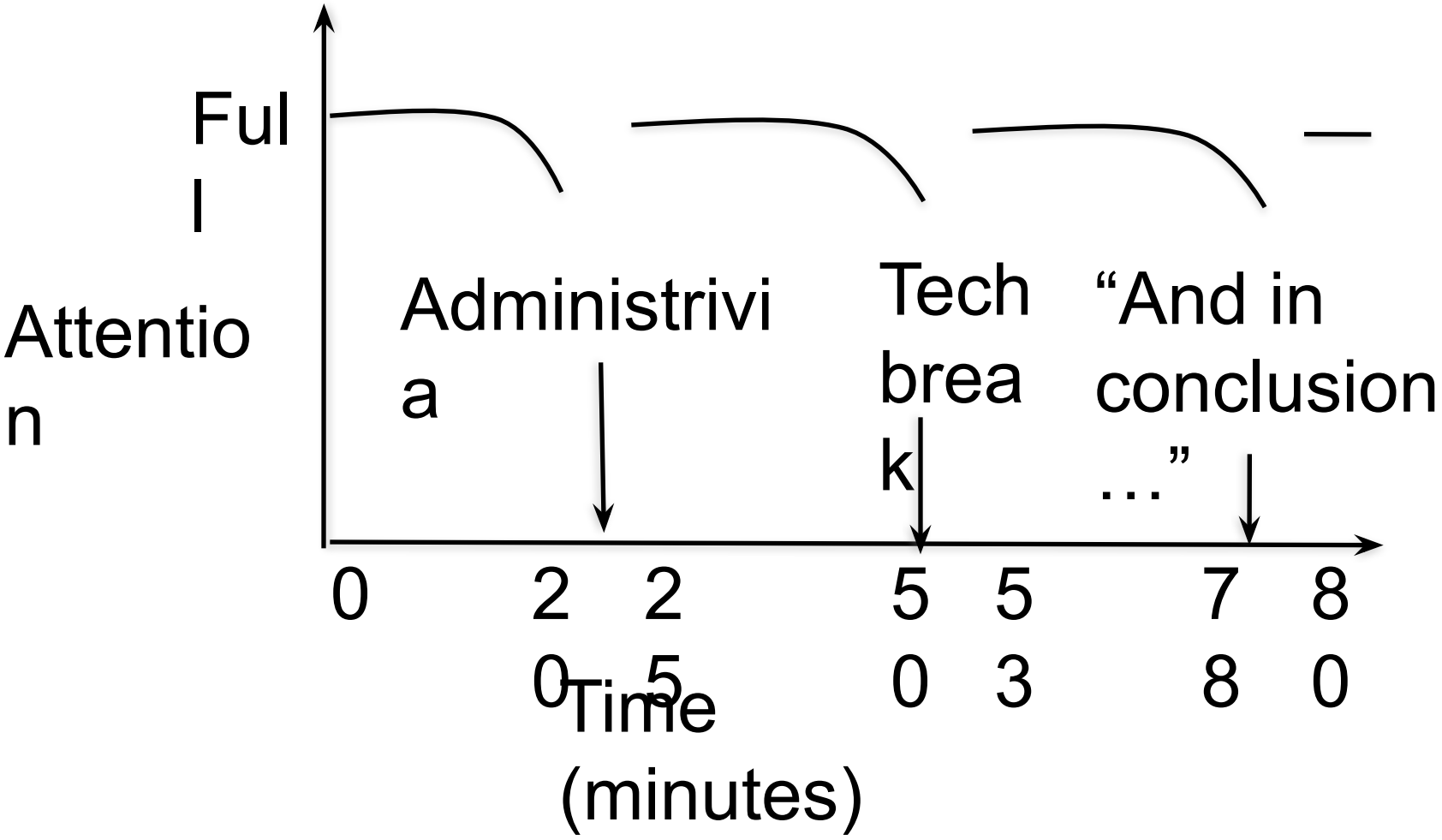
While many people say multitasking makes them more productive, research shows otherwise. Heavy multitaskers actually have more trouble focusing and shutting out irrelevant information, scientists say, and they experience more stress.

And scientists are discovering that even after the multitasking ends, fractured thinking and lack of focus persist. In other words, this is also your brain off computers.

The Rules
(and we really mean it!)

*

Architecture of a Lecture

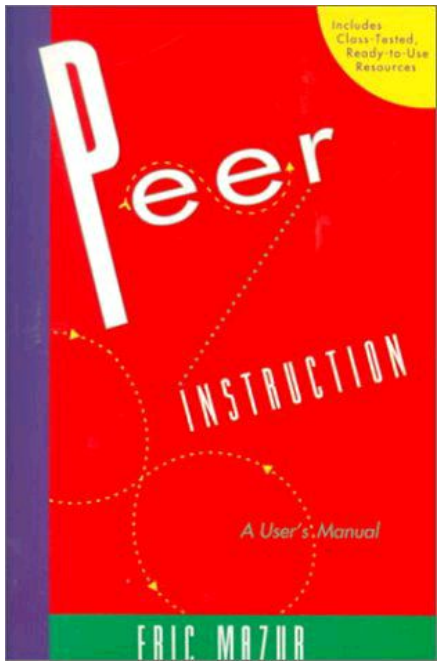


Peer Instruction

- Increase real-time learning in lecture, test understanding of concepts vs. details

  `mazur-www.harvard.edu/education/pi.phtml`

- As complete a "segment" ask multiple choice question
  - <1 minute: decide yourself, vote
  - <2 minutes: discuss in pairs, then team vote; flash card to pick answer
    - Try to convince partner; learn by teaching
- Mark and save flash cards



1 2 3 4

Lecture & Section

- Section
- Office hours & locations on homepage
  - Instructor & GSI office hours: concept/HW help
  - Staffed lab hours (TBA): programming help and hackfests

Meet the staff

- The GSIs are Michael Driscoll, Richard Xia
- Lab staff are Allen Chen, David Eliahu, Omer Spillinger, and Richard Zhao

Outline

- Class Organization (AF)
- Engineering SW is Different from HW
  (Next 5 slides, Book Sections 1.1-1.2 or §1.1-§1.2)
- Development Processes: Waterfall vs. Agile (§1.3)
- Assurance (§1.4)
- Productivity (§1.5)
- Software as a Service (§1.6) if time permits
- Service Oriented Architecture (§1.7) if time permits
- Cloud Computing (§1.8) if time permits

Engineering Software is Different from Engineering Hardware (*Engineering Long Lasting Software* §1.1-§1.2)

David Patterson

Engineering Software is Different from Hardware

- Q: Why so many SW disasters and no HW disasters?
  - Ariane 5 rocket explosion
  - Therac-25 lethal radiation overdose
  - Mars Climate Orbiter disintegration
  - FBI Virtual Case File project abandonment
- A: Nature of 2 media & subsequent cultures

Independent Products vs. Continuous Improvement

- Cost of field upgrade
- HW ≈ ∞
  - HW designs must be finished before manufactured and shipped
  - Bugs: Return HW (lose if many returns)
- SW ≈ 0
  - Expect SW gets better over time
  - Bugs: Wait for upgrade
- HW decays, SW long lasting

Independent Products vs. Continuous Improvement

- Cost of field upgrade
- HW_____
  - HW designs must be finished before manufactured and shipped
  - Bugs: Return HW (lose if many returns)
- SW_____
  - Expect SW gets better over time
  - Bugs: Wait for upgrade
- HW decays, SW long lasting

Legacy SW vs. Beautiful SW

- **Legacy code**: old SW that continues to meet customers' needs, but difficult to evolve due to design inelegance or antiquated technology
  - 60% SW maintenance costs adding new functionality to legacy SW
  - 17% for fixing bugs
- Contrasts with **beautiful code:** meets customers' needs and easy to evolve

Legacy SW vs. Beautiful SW

- **Legacy code**: old SW that continues to meet customers' needs, but difficult to evolve due to design inelegance or antiquated technology
  - ___% SW maintenance costs adding new functionality to legacy SW
  - ___% for fixing bugs
- Contrasts with **beautiful code:** meets customers' needs and easy to evolve

Legacy Code: Key but Ignored

- Missing from traditional SWE courses and textbooks
- Number 1 request from industry experts we asked: What should be in new SWE course?
- **NEW**: assignment to enhance legacy code in 2nd half of Berkley course

Question: Which type of SW is considered an epic failure?

- ☐ Beautiful code
- ☐ Legacy code
- ☐ Unexpectedly short-lived code
- ☐ Both legacy code and unexpectedly short lived code

Development processes: Waterfall vs. Agile

(*Engineering Long Lasting Software* §1.3)

David Patterson

Development Processes: Waterfall vs. Agile

- Waterfall "lifecycle" or development process
  - A.K.A. "Big Design Up Front" or BDUF

1. Requirements analysis and specification
2. Architectural design
3. Implementation and Integration
4. Verification
5. Operation and Maintenance

- Complete one phase before start next one
  - Why? Earlier catch bug, cheaper it is
  - Extensive documentation/phase for new people

How well does Waterfall work?

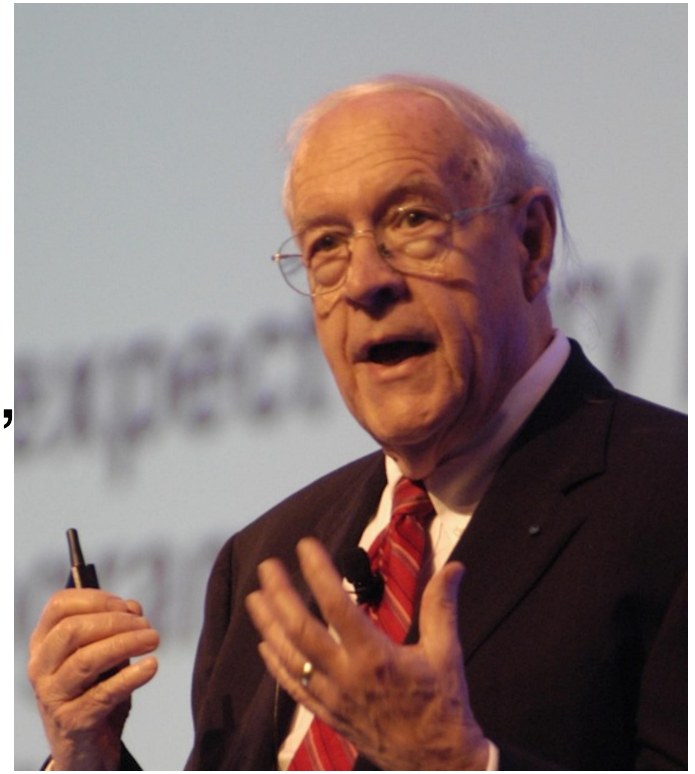
- Works well for important software with specs that won't change: NASA spacecraft, aircraft control, …
- But often when customer sees result, wants big changes
- But often after built first one, developers learn right way they should have built it

How well does Waterfall work?

- *"Plan to throw one [implementation] away; you will, anyhow*."

- Fred Brooks, Jr.

(received 1999 Turing Award for contributions to computer architecture, operating systems, and software engineering)



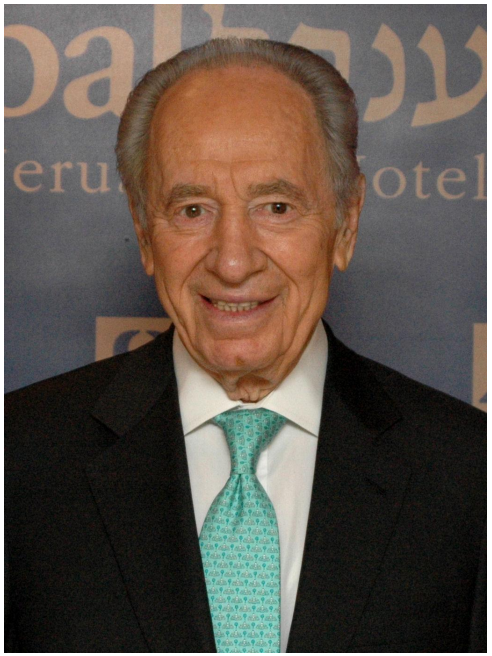
(Photo by Carola Lauber of SD&M www.sdm.de. Used by permission under CC-BY-SA-3.0.)

Peres's Law

"If a problem has no solution,
it may not be a problem,
but a fact, not to be solved,
but to be coped with over time."

— Shimon Peres

(winner of 1994
Nobel Peace Prize
for Oslo accords)



(Photo Source: Michael Thaidigsmann, put in public domain,
See http://en.wikipedia.org/wiki/File:Shimon_peres_wjc_90126.jpg)

Agile Manifesto, 2001

"We are uncovering better ways of developing SW by doing it and helping others do it. Through this work we have come to value

- Individuals and interactions over processes & tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more."

Agile Manifesto, 2001

"We are uncovering better ways of developing SW by doing it and helping others do it. Through this work we have come to value

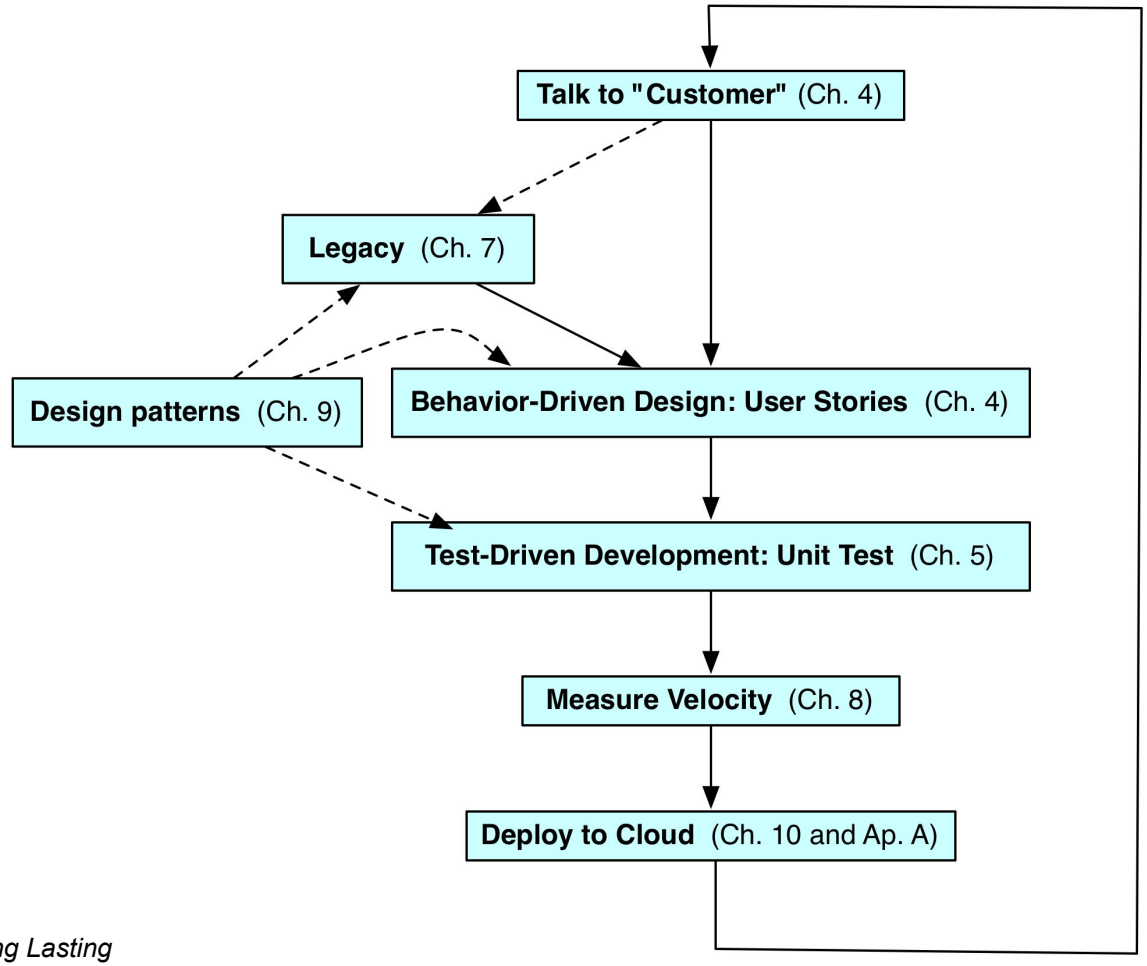
- Individuals and interactions over processes & tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more."

Agile lifecycle

- Embraces change as a fact of life: continuous improvement vs. phases
- Developers continuously refine working but incomplete prototype until customers happy, with customer feedback on each Iteration (every ~2 weeks)
- Agile emphasizes Test-Driven Development (TDD) to reduce mistakes, written down User Stories to validate customer requirements, Velocity to measure progress

Agile Iteration/ Book Organization



(Figure 1.4, *Engineering Long Lasting Software* by Armando Fox and David Patterson, Alpha edition, 2012.)

Question: What is NOT a key difference between Waterfall and Agile lifecycles?

- ☐ Waterfall uses long sequential phases, Agile uses quick iterations
- ☐ Waterfall has no working code until end, Agile has working each code iteration
- ☐ Waterfall uses written requirements, but Agile does not use anything written down
- ☐ Waterfall has an architectural design phase, but you cannot incorporate SW architecture into the Agile lifecycle

Assurance: Testing and Formal Methods

(*Engineering Long Lasting Software* §1.4)

David Patterson

Assurance

- Verification: Did you build the thing right?
  - Did you meet the specification?
- Validation: Did you build the right thing?
  - Is this what the customer wants?
  - Is the specification correct?
- Hardware focus generally Verification
- Software focus generally Validation
- 2 options: Testing and Formal Methods

Assurance

- Verification: Did you build the thing right?
  - Did you meet the specification?
- Validation: Did you build the right thing?
  - Is this what the customer wants?
  - Is the specification correct?
- Hardware focus generally_____
- Software focus generally_____
- 2 options: Testing and Formal Methods

Testing

- Exhaustive testing infeasible
- Divide and conquer: perform different tests at different phases of SW development
  - Upper level doesn't redo tests of lower level

_____ test: integrated program meets its specifications
_____ test: interfaces between units have consistent assumptions, communicate correctly
_____ test: across individual units
___ test: single method does what was expected

Testing

- Exhaustive testing infeasible
- Divide and conquer: perform different tests at different phases of SW development
  - Upper level doesn't redo tests of lower level

System or acceptance test: integrated program meets its specifications
Integration test: interfaces between units have consistent assumptions, communicate correctly
Module or functional test: across individual units
Unit test: single method does what was expected

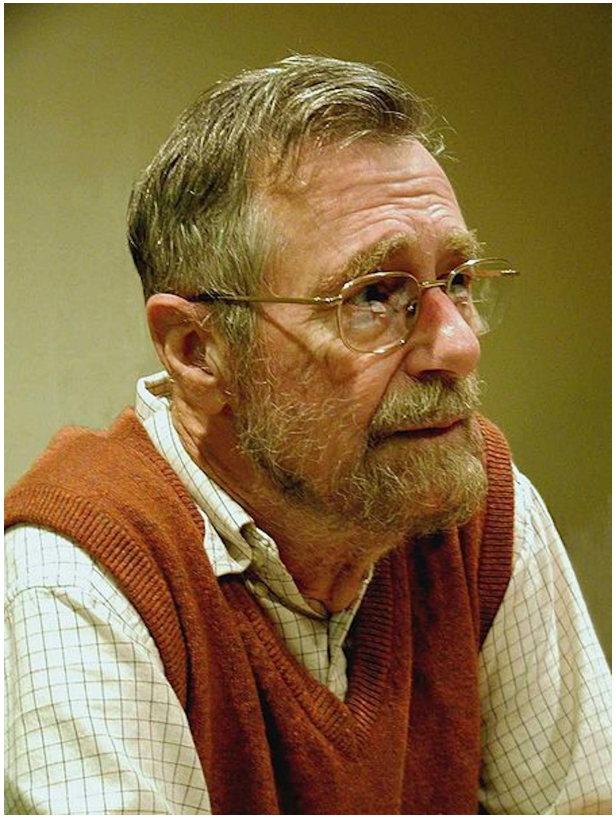
More Testing

- Coverage: % of code paths tested
- Regression Testing: automatically rerun old tests so changes don't break what used to work
- Continuous Integration Testing: continuous regression testing vs. later phases
- Agile => Test Driven Design (TDD)
  write tests *before* you write the code you wish you had (tests drive coding)

Limits of Testing

- Program testing can be used to show the presence of bugs, but never to show their absence!
  - Edsger W. Dijkstra

(received the 1972 Turing Award for fundamental contributions to developing programming languages)



Formal Methods

- Start with formal specification & prove program behavior follows spec. Options:

1. Human does proof
2. Computer via automatic theorem proving
   - Uses inference + logical axioms to produce proofs from scratch
3. Computer via model checking
   - Verifies selected properties by exhaustive search of all possible states that a system could enter during execution

Formal Methods

- Computationally expensive, so use only if
  - Small, fixed function
  - Expensive to repair, very hard to test
  - E.g., Network protocols, safety critical SW
- Biggest: OS kernel 10K LOC @ $500/LOC
  - NASA SW $80/LOC
- This course: rapidly changing SW, easy to repair, easy to test => no formal methods
  - Discuss again on future of engineering SW

Question: Which statement is NOT true about testing?

- ☐ With better test coverage, you are more likely to catch faults
- ☐ While difficult to achieve, 100% test coverage insures design reliability
- ☐ Each higher level test delegates more detailed testing to lower levels
- ☐ Unit testing works within a single class and module testing works across classes

Productivity: Conciseness, Synthesis, Reuse, and Tools

(*Engineering Long Lasting Software* §1.5)

David Patterson

Productivity

- Moore's Law => 2X transistors/1.5 years
  - ⇒ HW designs get bigger
  - ⇒ Faster processors and bigger memories
  - ⇒ SW designs get bigger
  - ⇒ Must improve HW & SW productivity
- 4 techniques
  1. Clarity via conciseness
  2. Synthesis
  3. Reuse
  4. Automation and Tools

Clarity via conciseness

1. Syntax: shorter and easier to read
   `assert_greater_than_or_equal_to(a,7)`
   vs. a.should be ≥ 7
2. Raise the level of abstraction:
   - HLL programming languages vs. assembly lang
   - Automatic memory management (Java vs.C)
   - Scripting languages: reflection, metaprogramming

Clarity via conciseness

1. Syntax: shorter and easier to read
   `assert_greater_than_or_equal_to(a,7)`
   vs. _____
2. Raise the level of abstraction:
   - HLL programming languages vs. assembly lang
   - Automatic memory management (Java vs.C)
   - Scripting languages: reflection, metaprogramming

Synthesis

- Software synthesis
  - BitBlt: generate code to fit situation & remove conditional test
- Future Research: Programming by example

Reuse

- Reuse old code vs. write new code
- Techniques in historical order:

1. Procedures and functions
2. Standardized libraries (reuse single task)
3. Object oriented programming: reuse and manage collections of tasks
4. Design patterns: reuse a general strategy even if implementation varies

Automation and Tools

- Replace tedious manual tasks with automation to save time, improve accuracy
  - New tool can make lives better (e.g., make)
- Concerns with new tools: Dependability, UI quality, picking which one from several
- We think good software developer must repeatedly learn how to use new tools
  - Lots of chances in this course: Cucumber, RSpec, Pivotal Tracker, …

Question: Which statement is TRUE about productivity?

- ☐ Copy and pasting code is another good way to get reuse
- ☐ Metaprogramming helps productivity via program synthesis
- ☐ Of the 4 productivity reasons, the primary one for HLL is reuse
- ☐ A concise syntax is more likely to have fewer bugs and be easier to maintain

DRY

- “Every piece of knowledge must have a single, unambiguous, authoritative representation within a system.”
  - Andy Hunt and Dave Thomas, 1999
- Don't Repeat Yourself (DRY)
  - Don’t want to find many places have to apply same repair
- Don’t copy and paste code!

Software as a Service (SaaS)

(*Engineering Long Lasting Software* §1.6)

David Patterson

Software as a Service: SaaS

- Traditional SW: binary code installed and runs wholly on client device
- SaaS delivers SW & data as service over Internet via thin program (e.g., browser) running on client device
  - Search, social networking, video
- Now also SaaS version of traditional SW
  - E.g., Microsoft Office 365, TurboTax Online

6 Reasons for SaaS

1. No install worries about HW capability, OS
2. No worries about data loss (at remote site)
3. Easy for groups to interact with same data
4. If data is large or changed frequently, simpler to keep 1 copy at central site
5. 1 copy of SW, controlled HW environment => no compatibility hassles for developers
6. 1 copy => simplifies upgrades for developers *and* no user upgrade requests

SaaS Loves Agile & Rails

- Frequent upgrades matches Agile lifecycle
- Many frameworks for Agile/SaaS
- We use Ruby on Rails ("Rails")
- Ruby, a modern scripting language: object oriented, functional, automatic memory management, dynamic types, reuse via mix-ins, synthesis via metaprogramming
- Rails popular – e.g., Twitter

Which is weakest argument for a Google app's popularity as SaaS?

- ☐ Don't lose data: Gmail
- ☐ Cooperating group: Documents
- ☐ Large/Changing Dataset: YouTube
- ☐ No field upgrade when improve app: Search

Outline

- Class Organization (AF)
- Engineering SW is Different from HW (§1.1-§1.2)
- Development Processes: Waterfall vs. Agile (§1.3)
- Assurance (§1.4)
- Productivity (§1.5)
- Software as a Service (§1.6) if time permits
- Service Oriented Architecture (§1.7) if time permits (Next 6 slides)
- Cloud Computing (§1.8) if time permits

Service Oriented Architecture(SOA)

(*Engineering Long Lasting Software* §1.7)

David Patterson

Service Oriented Architecture

- SOA: SW architecture where all components are designed to be services
- Apps composed of interoperable services
  - Easy to tailor new version for subset of users
  - Also easier to recover from mistake in design
- Contrast to “SW silo” without internal APIs

CEO: Amazon shall use SOA!

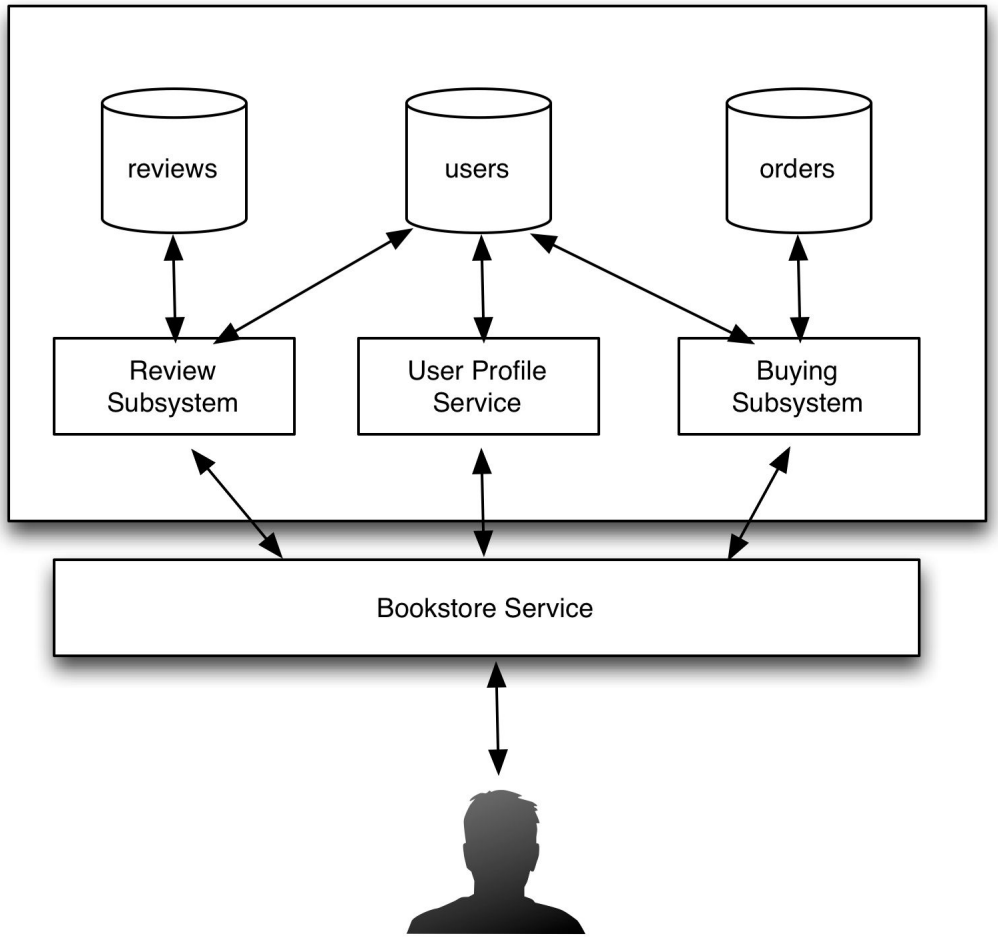
1. “All teams will henceforth expose their data and functionality through service interfaces
2. Teams must communicate with each other through these interfaces
3. There will be no other form of interprocess communication allowed: no direct linking, no direct reads of another team's data store, no shared-memory model, no back-doors whatsoever. The only communication allowed is via service interface calls over the network.

CEO: Amazon shall use SOA!

4. It doesn't matter what [API protocol] technology you use.
5. Service interfaces, without exception, must be designed from the ground up to be externalizable. That is to say, the team must plan and design to be able to expose the interface to developers in the outside world. No exceptions.
6. Anyone who doesn't do this will be fired.
7. Thank you; have a nice day!”

Bookstore: Silo

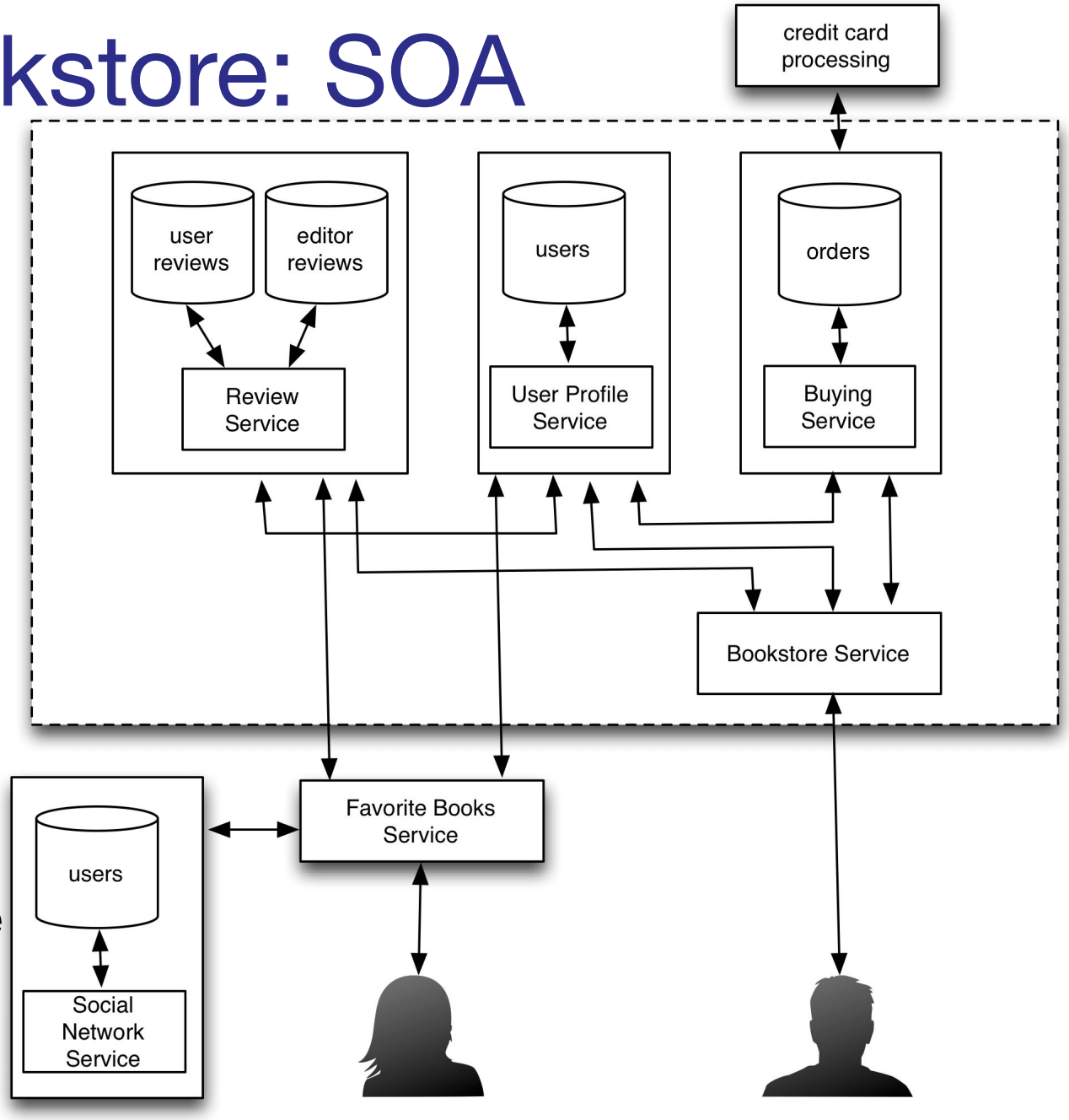
- Internal subsystems can share data directly
  - Review access user profile
- All subsystems inside single API ("Bookstore")



(Figure 1.2, *Engineering Long Lasting Software* by Armando Fox and David Patterson, Alpha edition, 2012.)

Bookstore: SOA

- Subsystems independent, as if in separate datacenters
  - Review Service access User Service API
- Can recombine to make new service ("Favorite Books")



(Figure 1.3, *Engineering Long Lasting Software* by Armando Fox and David Patterson, Alpha edition, 2012.)

Which statements NOT true about SOA?

- ☐ Debugging is easier with SOA
- ☐ Security can be harder with SOA
- ☐ SOA improves developer productivity primarily through reuse
- ☐ No service can name or access another service's data; it can only make requests for data thru an external API

Cloud Computing, Fallacies and Pitfalls, and End of Chapter 1

(*Engineering Long Lasting Software* §§1.8, 1.9, 1.12)

David Patterson

SaaS Infrastructure?

- SaaS demands on infrastructure

1. Communication: allow customers to interact with service
2. Scalability: fluctuations in demand during + new services to add users rapidly
3. Dependability: service and communication continuously available 24x7

Clusters

- Clusters: Commodity computers connected by commodity Ethernet switches

1. More scalable than conventional servers
2. Much cheaper than conventional servers
   - 20X for equivalent vs. largest servers
3. Few operators for 1000s servers
   - Careful selection of identical HW/SW
   - Virtual Machine Monitors simplify operation
4. Dependability via extensive redundancy

Warehouse Scale Computers

- Economies of scale pushed down cost of largest datacenter by factors 3X to 8X
  - Purchase, house, operate 100K v. 1K computers
- Traditional datacenters utilized 10% - 20%
- Make profit offering pay-as-you-go use at less than your costs for as many computers as you need

Utility Computing / Public Cloud Computing

- Offers computing, storage, communication at pennies per hour +
- No premium to scale:

  1000 computers @ 1 hour
  = 1 computer @ 1000 hours
- Illusion of infinite scalability to cloud user
  - As many computers as you can afford
- Leading examples: Amazon Web Services, Google App Engine, Microsoft Azure

2012 AWS Instances & Prices

Instance	Per Hour	Ratio to Small	Compute Units	Virtual Cores	Compute Unit/ Core	Memory (GB)	Disk (GB)	Address
Standard Small	$0.085	1.0	1.0	1	1.00	1.7	160	32 bit
Standard Large	$0.340	4.0	4.0	2	2.00	7.5	850	64 bit
Standard Extra Large	$0.680	8.0	8.0	4	2.00	15.0	1690	64 bit
High-Memory Extra Large	$0.500	5.9	6.5	2	3.25	17.1	420	64 bit
High-Memory Double Extra Large	$1.200	14.1	13.0	4	3.25	34.2	850	64 bit
High-Memory Quadruple Extra Large	$2.400	28.2	26.0	8	3.25	68.4	1690	64 bit
High-CPU Medium	$0.170	2.0	5.0	2	2.50	1.7	350	32 bit
High-CPU Extra Large	$0.680	8.0	20.0	8	2.50	7.0	1690	64 bit
Cluster Quadruple Extra Large	$1.300	15.3	33.5	16	2.09	23.0	1690	64 bit
Eight Extra Large	$2.400	28.2	88.0	32	2.75	60.5	1690	64 bit

Supercomputer for hire

- Top 500 supercomputer competition
- 290 Eight Extra Large (@ $2.40/hour)
  = 240 TeraFLOPS
- 42nd/500 supercomputer @ ~$700 per hour
- Credit card => can use 1000s computers
- FarmVille on AWS
  - Prior biggest online game 5M users
  - What if startup had to build datacenter?
  - 4 days =1M; 2 months = 10M; 9 months = 75M

IBM Watson for Hire?

- Jeopardy Champion IBM Watson
- Hardware: 90 IBM Power 750 servers
  - 3.5 GHz 8 cores/server
- 90 @~$2.40/hour = ~$200/hour
- Cost of human lawyer or account
- For what tasks could AI be as good as highly trained person @ $200/hour?
- What would this mean for society?

Which statements NOT true about SaaS, SOA, and Cloud Computing?

- ☐ Clusters are collections of commodity servers connected by LAN switches
- ☐ The Internet supplies the communication for SaaS
- ☐ Cloud computing uses HW clusters + SW layer using redundancy for dependability
- ☐ Private datacenters could match cost of Warehouse Scale Computers if they just purchased the same type of hardware

Fallacies and Pitfalls

- Fallacy: If a software project is falling behind schedule, catch up by adding people
  - Adding people actual makes it worse!
  1. Time for new people to learn about project
  2. Communication increases as project grows, which reduces time available get work done

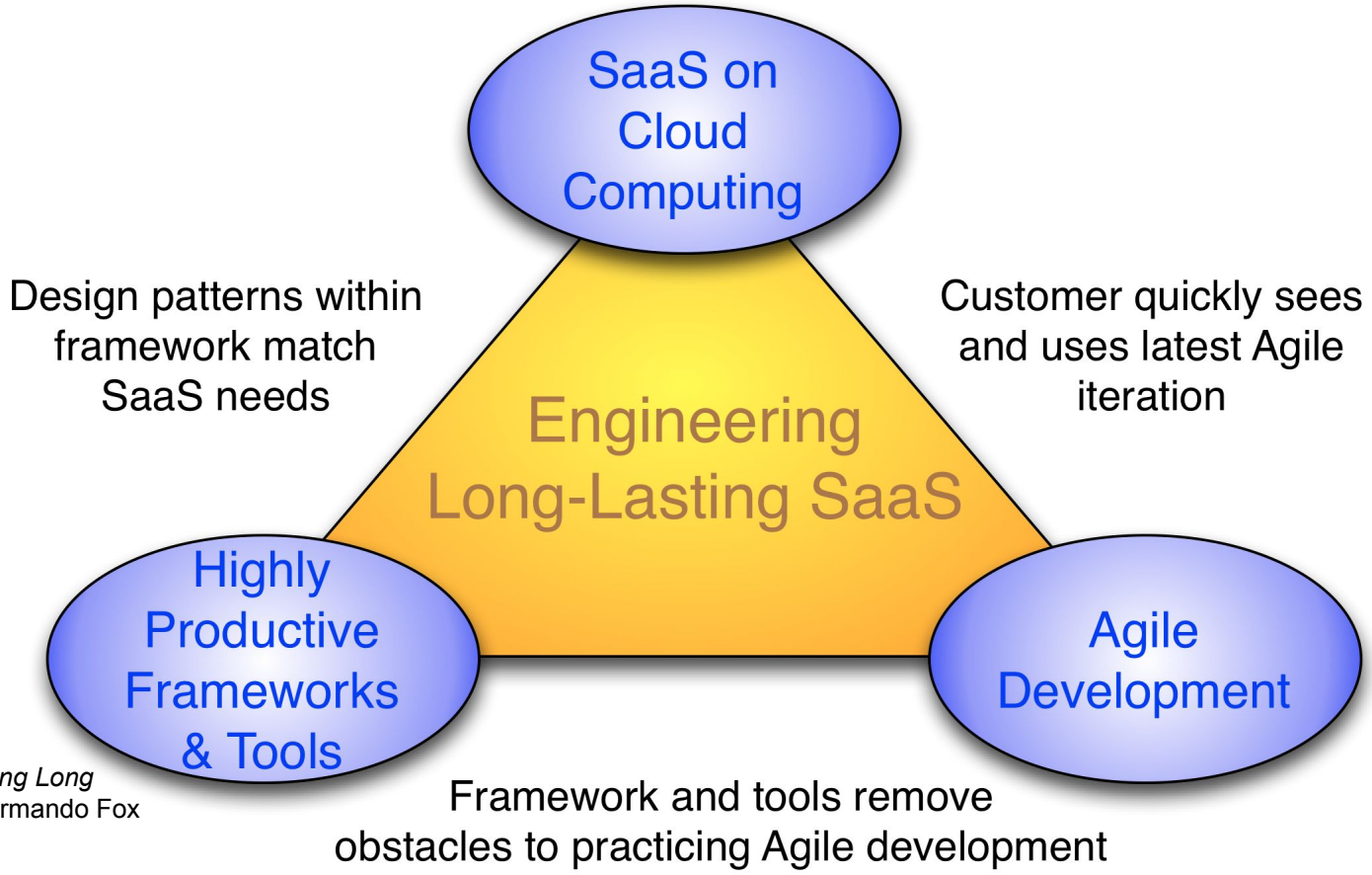
"Adding manpower to a late software project makes it later."
Fred Brooks, Jr. *The Mythical Man Month*

Fallacies and Pitfalls

- Pitfall: Ignoring the cost of software design
  - Since ≈0 cost to manufacture software, might believe ≈0 cost to remanufacture the way the customer wants
  - Ignores the cost of design and test
- (Is cost ~no cost of manufacturing software/data same rationale to pirate data? No one should pay for development, just for manufacturing?)

Summary: Engineering SW is More Than Programming

- Long-lasting, evolvable SW vs. short life of HW led to different development processes



(Figure 1.6, *Engineering Long Lasting Software* by Armando Fox and David Patterson, Alpha edition, 2012.)