CS 169
Software Engineering

Armando Fox and David Patterson

Outline

Class Organization

§1.1 Introduction to Software Engineering

§1.2 Software as a Service

§1.3 Service Oriented Architecture

§1.4 Cloud Computing

§1.5 Beautiful vs. Legacy Code

§1.6 Software Quality Assurance: Testing

§1.7 Productivity: Clarity via Conciseness, Synthesis, Reuse, & Tools

Introduction to Software Engineering

(*Engineering Software as a Service* §1.1)

David Patterson

Ranking Top 200 Jobs (2012)

1. Software Engineer
28. Civil Engineer
38. Nurse
40. Physician
47. Accountant
60. Mechanical Engineer
73. Electrical Engineer
87. Attorney
104. Airline Pilot
133. Fashion Designer
137. High School Teacher
163. Police Officer
173. Flight Attendant
185. Firefighter
196. Newspaper Reporter
200. Lumberjack

InformationWeek 5/15/12. Based on salary, stress levels, hiring outlook, physical demands, and work environment (www.careercast.com)

If SW Engineering so popular, why so many SWE disasters?

(Search Wikipedia for details)

- Therac-25 lethal radiation overdose
  - 1985: Reused SW from machine with HW interlock on machine without it: SW bug => 3 died
- Mars Climate Orbiter disintegration
  - 1999: SW used wrong units (pound-seconds vs. newton-seconds) => wasted $325M
- FBI Virtual Case File project abandonment
  - 2005: give up after 5 years of work => wasted $170M
- Ariane 5 rocket explosion
  - 1996 => wasted $370M

How can you avoid infamy?

- Lessons from 60 years of SW development
- This class will review many approaches in lecture, listing pros and cons
- Will get hands on experience with one approach good for Software as a Service
  - SaaS MOOC: >10,000 completed certificates
  - UCB SaaS version Enrollments: 31➔66➔75➔112➔161➔240
  - SaaS version teaching evaluation(F12): 6.4 / 7

Software as a Service

(*Engineering Software as a Service* §1.2)

David Patterson

Software Targets

- Traditional SW: binary code installed and runs wholly on client device, which users must upgrade repeatedly
  - Must work with many versions of hardware, many versions of OS
  - New versions must go through extensive release cycle to ensure compatibility
- An alternative where develop SW that only needs to work on one HW & OS platform?

  ➔ Quicker release cycle, no user upgrades?

Software as a Service: SaaS

- 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
- Instructors think SaaS is revolutionary, the future of virtually all software

6 Reasons for SaaS

1. No install worries about HW capability, OS
2. No worries about data loss (data is remote)
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, single HW/OS environment
   => no compatibility hassles for developers
   => beta test new features on 1% of users
6. 1 copy => simplifies upgrades for developers *and* no user upgrade requests

SaaS Loves Rails

- Many frameworks/languages for SaaS
  - Django/Python, Zend/PHP, Spring/Java
- We use Ruby on Rails (“Rails”)
- Rails popular programming framework that uses Ruby – e.g., Twitter
- Ruby, a modern scripting language: object oriented, functional, automatic memory management, dynamic types, reuse via mix-ins & closures, synthesis via metaprogramming

Why take time for Ruby/Rails?

- 12 weeks to learn:
  - SaaS, Agile, Pair Programming, Behavior Driven Design, LoFi UI, Storyboards, User Stories, Test Driven Development, Enhance Legacy Code, Scrum, Velocity, JavaScript, Design Patterns, UML, Deployment, Security
  - Part time (taking 3 other classes or full time job)
- Only hope is highly productive language, tools, framework: We believe Rails is best
  - See “Crossing the Software Education Chasm,” A. Fox, D. Patterson, *Comm. ACM*, May 2012

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

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

Service Oriented Architecture (SOA)

(*Engineering Software as a Service* §1.3)

David Patterson

Software Architecture

- Can you design software so that you can recombine independent modules to offer many different apps without a lot of programming?

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!

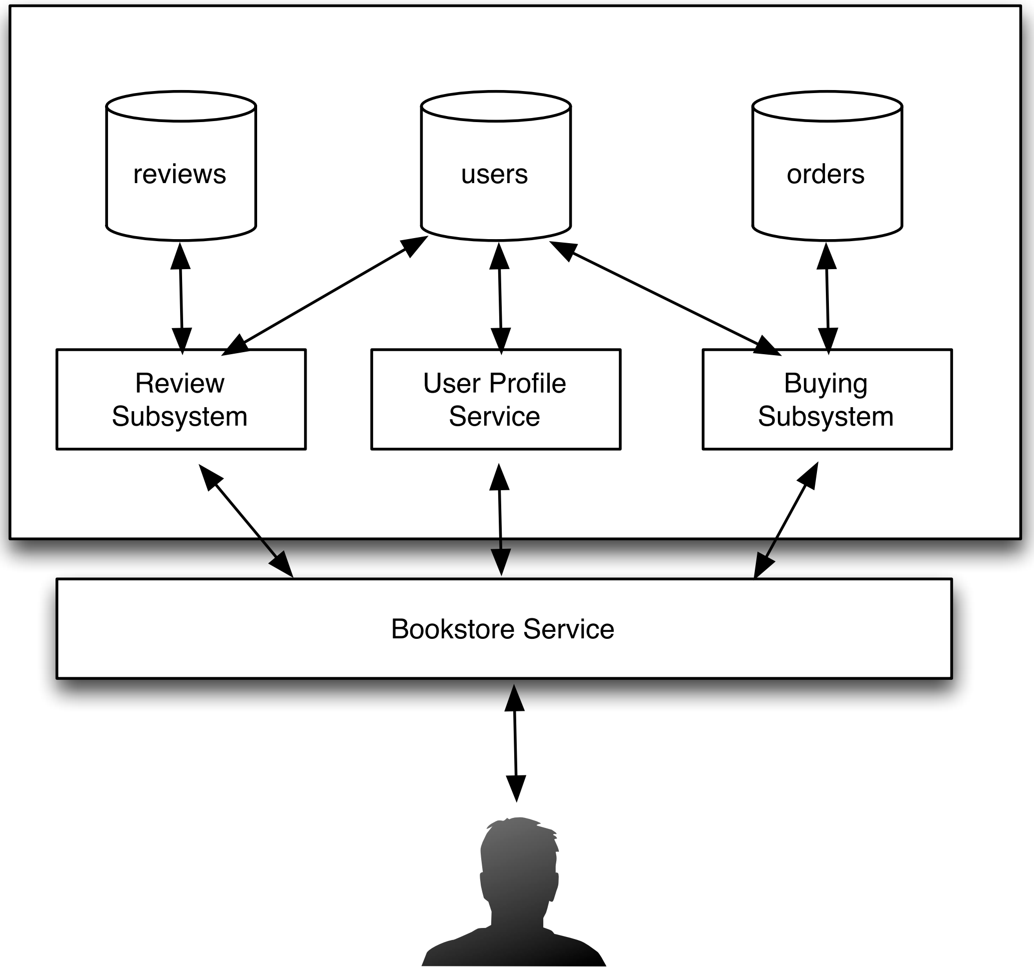
1. “All teams will henceforth expose their data and functionality through service interfaces.”
2. “Teams must communicate with each other through these interfaces.”
3. “There will be no other form of interprocess communication allowed: no direct linking, no direct reads of another team's data store, 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

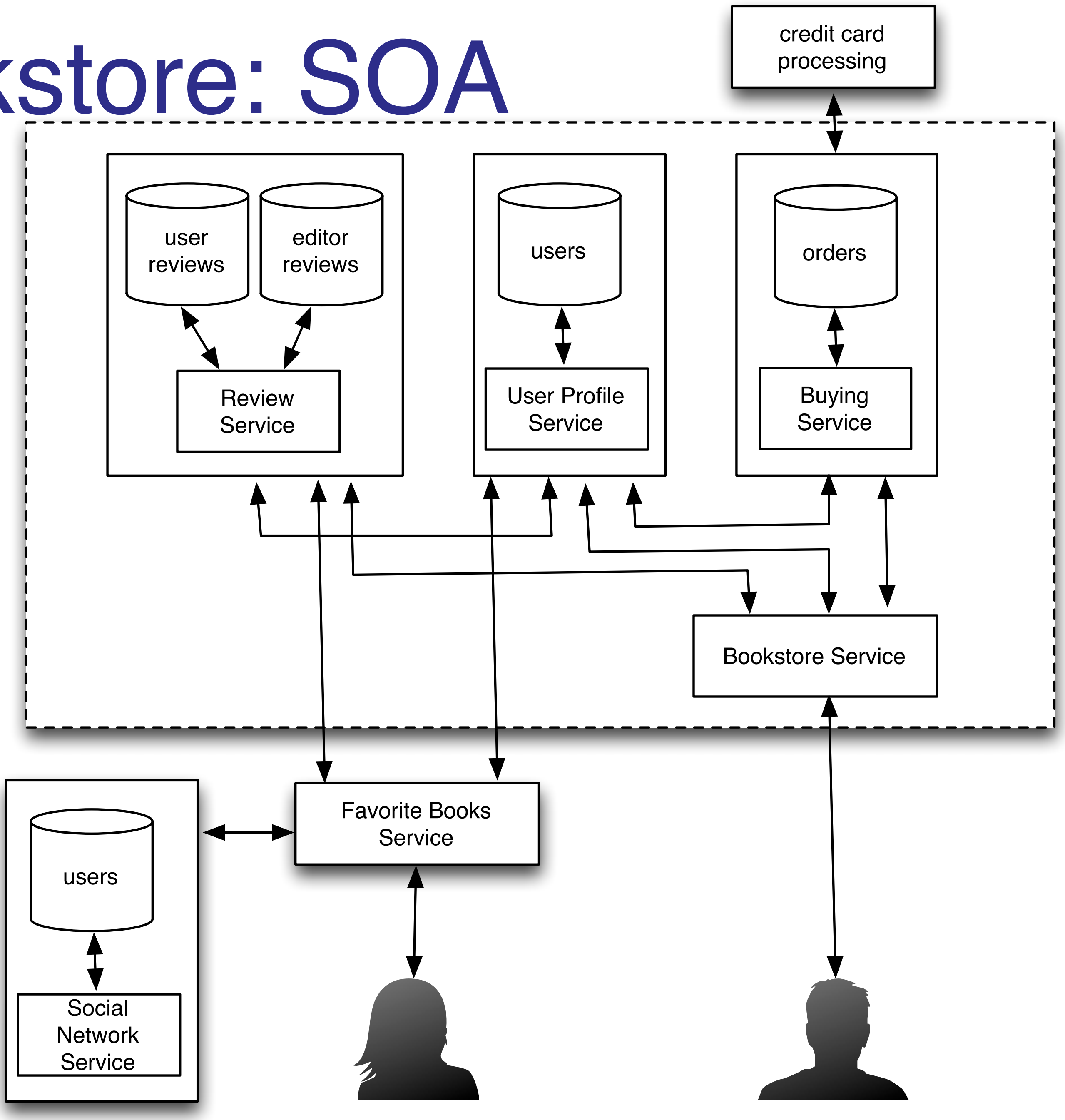
- Internal subsystems can share data directly
  - Review access user profile
- All subsystems inside single API ("Bookstore")



(Figure 1.2, *Engineering Software as a Service* by Armando Fox and David Patterson, 2nd Beta edition, 2013.)

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 Software as a Service* by Armando Fox and David Patterson, 2nd Beta edition, 2013.)

Which statements NOT true about SOA?

- ☐ SOA does not affect performance
- ☐ Silo’d systems are likely completely down on a failure, SOA must handle partial failures
- ☐ SOA improves developer productivity primarily through reuse
- ☐ No SOA service can name or access another service's data; it can only make requests for data thru an external API

Cloud Computing

(*Engineering Software as a Service* §1.4)

David Patterson

What is ideal HW for SaaS?

- Amazon, Google, Microsoft … developed hardware systems to run SaaS
- What did they use: Mainframes? Supercomputers?
- How can independent SW developers build SaaS apps and compete without similar HW investments to Amazon, Google, Microsoft?

SaaS Infrastructure?

- SaaS’s 3 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 & communication available 24x7

Services on 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. Dependability via extensive redundancy
4. Few operators for 1000s servers
   – Careful selection of identical HW/SW
   – Virtual Machine Monitors simplify operation

Warehouse Scale Computers

- Clusters grew from 1000 servers to 100,000 based on customer demand for SaaS apps
- 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%
- Earn $ offering pay-as-you-go use at less than customer's costs for as many computers as customer needs

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

2013 AWS Instances & Prices

Instance Type	Per Hour	$ Ratio to small	Virtual Cores	Compute Units	Memory (GiB)	Storage (GB)
m1.small	$0.06	1.0	1	1.0	1.7	1 x 160
m1.medium	$0.12	2.0	1	2.0	3.8	1 x 410
m1.large	$0.24	4.0	2	4.0	7.5	2 x 420
m1.xlarge	$0.48	8.0	4	8.0	15.0	4 x 420
m3.xlarge	$0.50	8.3	4	13.0	15.0	EBS
m3.2xlarge	$1.00	16.7	8	26.0	30.0	EBS
c1.medium	$0.15	2.5	2	5.0	1.7	1 x 350
c1.xlarge	$0.58	9.7	8	20.0	7.0	4 x 420
cc2.8xlarge	$2.40	40.0	32	88.0	60.5	4 x 840
m2.xlarge	$0.41	6.8	2	6.5	17.1	1 x 420
m2.2xlarge	$0.82	13.7	4	13.0	34.2	1 x 850
m2.4xlarge	$1.64	27.3	8	26.0	68.4	2 x 840
cr1.8xlarge	$3.50	58.3	32	88.0	244.0	2 x 120 SSD
hi1.4xlarge	$3.10	51.7	16	35.0	60.5	2 x 1024 SSD
hs1.8xlarge	$4.60	76.7	16	35.0	117.0	24 x 2048
t1.micro	$0.02	0.3	1	varies	0.6	EBS
cg1.4xlarge	$2.10	35.0	16	33.5	22.5	2 x 840

Supercomputer for hire

- Top 500 supercomputer competition in 2012
- 532 Eight Extra Large (@ $2.40/hour), 17000 cores = 240 TeraFLOPS
- 72nd/500 supercomputer @ ~$1300 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 attorney or accountant
- 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 apps
- ☐ Cloud computing uses HW clusters + SW layer using redundancy for dependability
- ☐ Private datacenters could match the cost of Warehouse Scale Computers if they just used the same type of hardware and software

Legacy SW vs. Beautiful SW

(*Engineering Software as a Service* §1.5)

David Patterson

Programming Aesthetics

- Do I care what others think of my code?
  - If it works, does it matter what code looks like?

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 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 Code: Vital but Ignored

- Missing from traditional SWE courses and textbooks
- #1 request from industry experts we asked: What should be in new SWE course?
  - Save work by reusing existing code (e.g., open source)
- Will have legacy lectures and programming assignments later in course
  - Helps you learn how to make beautiful code

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

Quality Assurance &Testing

(*Engineering Software as a Service* §1.6)

David Patterson

Software Quality

- What is software quality, and how to we assure it? (QA)
- V&V: What is the difference (if any) between Verification and Validation?

Software Quality

- Product quality (in general): "fitness for use"
  - Business value for customer *and* manufacturer
  - *Quality Assurance* : processes/standards => high quality products & to improve quality
- Software quality:
  1. Satisfies customers' needs—easy to use, gets correct answers, does not crash, …
  2. Be easy for developer to debug and enhance
- Software QA: ensure quality and improve processes in SW organization

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
- Testing to Assure Software Quality

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

- Black box vs. White Box testing
  - Testing based on specs vs. on implementation
- Test Coverage: % of code paths tested
- Regression testing: automatically rerun old tests so changes don't break what used to work
- Continuous Integration (CI) testing: continuous regression testing on each code check-in vs. later testing phase

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* §1.7)

David Patterson

Productivity

- 50 years of Moore's Law => 2X /1.5 years

  ⇒ HW designs get bigger

  ⇒ Faster processors and bigger memories

  ⇒ SW designs get bigger

  ⇒ Had to improve 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. ___________________

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

Synthesis

- Software synthesis
  - BitBlt: generate code to fit situation & remove conditional test
- Research Stage: 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
- Good software developer must repeatedly learn how to use new tools: lifetime learning
  - 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
- Refactor code so that can have a single place to do things

And in Conclusion: §§1.1-1.7

- Class: SW eng. Principles via Cloud app by team for customer + enhancing legacy app
- SaaS less hassle for developers and users
- Service Oriented Architecture makes it easy to reuse current code to create new apps
- Scale led to savings/CPU => reduced cost of Cloud Computing => Utility Computing
- Testing to assure software quality, which means good for customer *and* developer
- Developer Productivity: Conciseness, Synthesis, Reuse, and Tools