Yale

Welcome to CPSC 639 Software Engineering

http://cs.yale.edu/homes/ejk/sweng2017/

Day 1: Organization & Goals

Majority of slides are Instructor's slides accompanying the textbook
"Engineering Software as a Service: An Agile Approach Using Cloud Computing"

Slides thanks to Armando Fox, David Patterson, Ruzica Piskac, Rupak Majumdar

Yale

Welcome

- Instructor: Eric Koskinen
  - AKW 302
  - eric.koskinen@yale.edu
- Office Hours:
  - Friday 3:45 – 5
  - and by appointment
- TF: **TBD**

Yale

Why are you here?

Yale

What is *missing* from our students' dev skills?

1. Legacy code

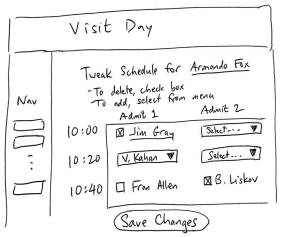
2. Working with non-technical customers

3. Testing

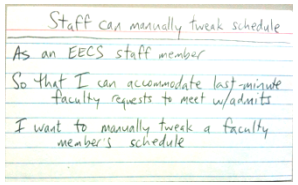
Yale

CPSC 1639 in 2 slides

Talk to customer



Lo-fi UI mockup



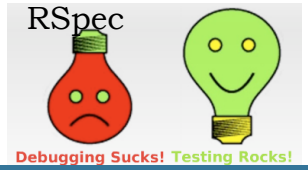
User stories & scenarios

Legacy Code

Design patterns

Cucumber

Behavior-driven Design / user stories



Test-first dev. (unit/funct.)

PIVOTAL TRACKER

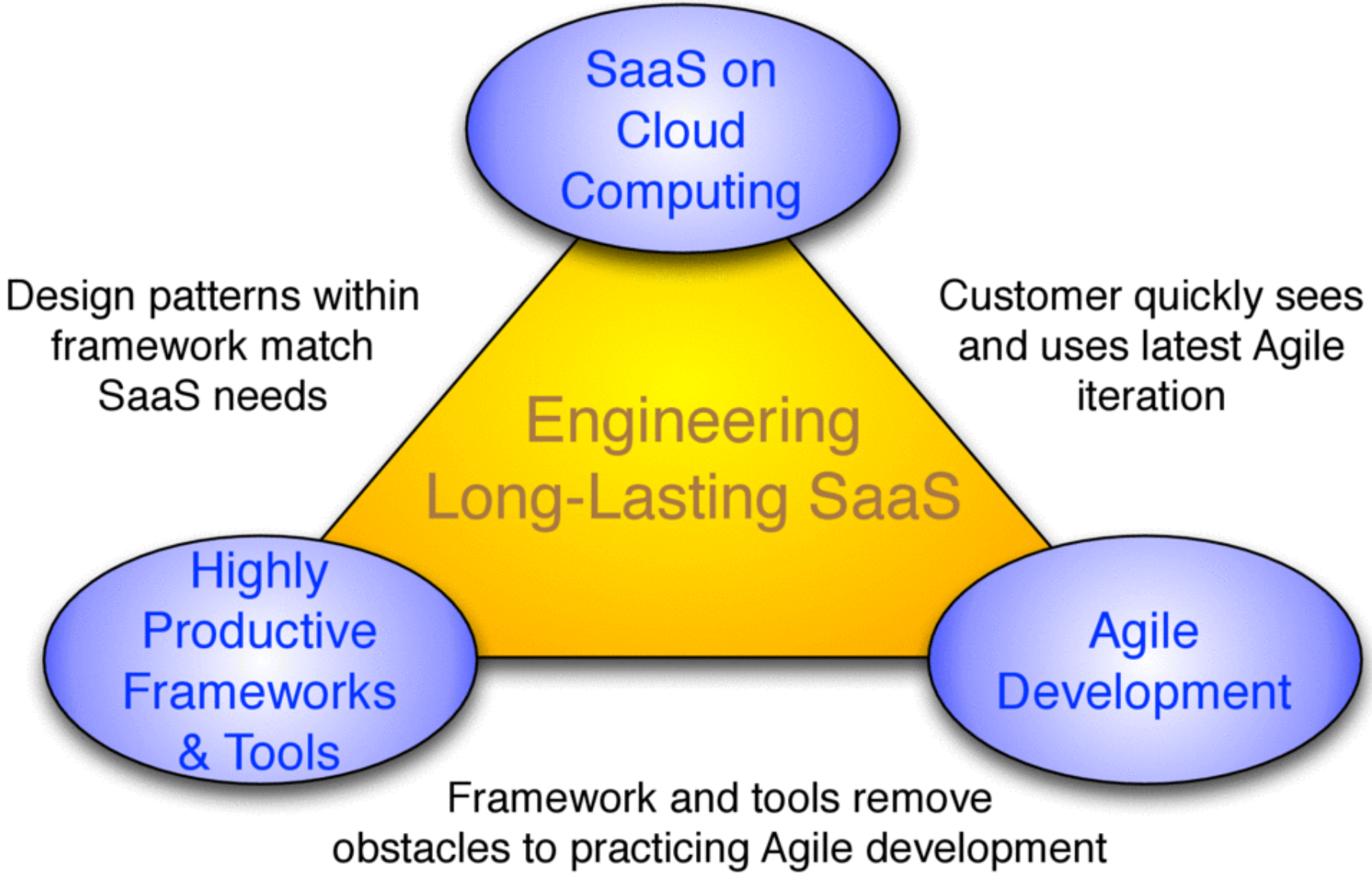
Measure Velocity

heroku

Deploy

Yale

CPSC 639 in 2 slides



Yale

What success looks like: you will…

- Develop the technical sophistication to become part of a *community of practice* in agile software engineering
- Leave a happy customer with a *usable, deployed, well tested software artifact* that can be further improved later
  - ➔ see *projects.saas-class.org* for examples
- What a good repo looks like: https://github.com/saasbook/coursequestionbank

Yale

Introduction to Software Engineering

(*Engineering Software as a Service* §1.1)

David Patterson

Yale

Who is this woman, what year is it, and what is she doing?

- Margaret Hamilton, 1965
- Director & Supervisor of Software Programming [sic] for Project Apollo
- Work on priority-based asynchronous scheduling prevented last-minute abort of the first moon landing
- Coined term “Software Engineering” at first-ever conference on topic (1968)
  - …convened to address the “software c
- **New in 2016:** Apollo 11 source code is on GitHub! `chrislgarry/Apollo-11`

Ranking Top 200 Jobs (2012)

1. Software Engineer

28. Civil Engineer

34. Programmer

40. Physician

47. Accountant

60. Mecha

73. Electri

87. Attorney

104. Airline Pilot

173. Flight Attendant

200. Lumberjack

Researches, designs, develops and maintains software systems along with hardware development for medical, scientific, and industrial purposes. Income: $88,142 (+25%)

Organizes and lists the instructions for computers to process data and solve problems in logical order. Income: $71,178

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

Yale

Is it stress-free?

http://www.youtube.com/watch?v=kYUrqdUyEpI

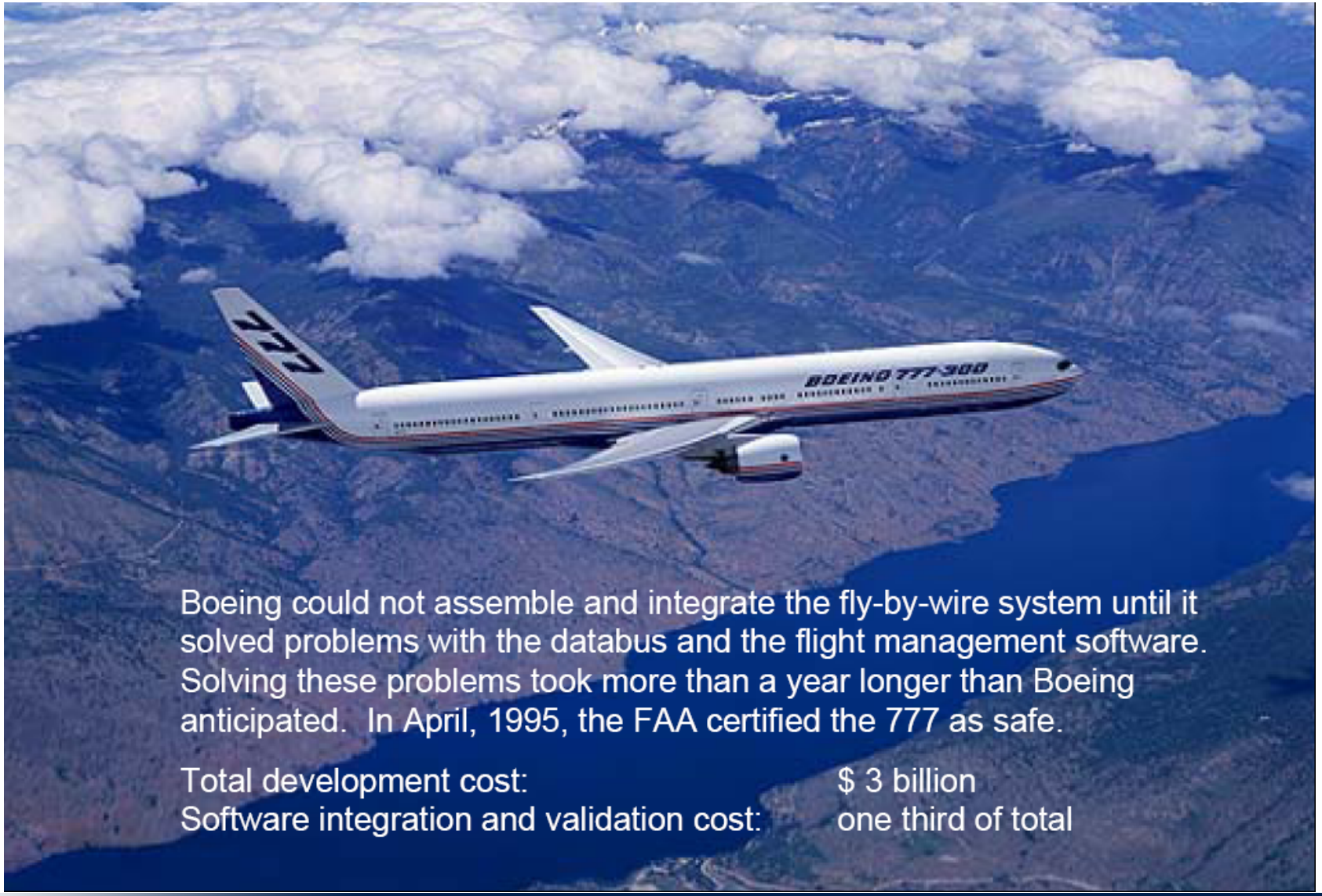
Yale

Is it stress-free?

- Maiden flight of the Ariane 5 rocket on the 4th of June 1996
- Software error: attempt to cram a 64-floating point number to a 16-bit integer failed
- Financial loss: $500,000,000 (including indirect costs: $2,000,000,000)

Yale

Is it stress-free?



Boeing could not assemble and integrate the fly-by-wire system until it solved problems with the databus and the flight management software. Solving these problems took more than a year longer than Boeing anticipated. In April, 1995, the FAA certified the 777 as safe.

Total development cost:	$ 3 billion
Software integration and validation cost:	one third of total

Yale

Is it stress-free?

- More examples:
  - 1985: Therac-25, 3 deaths.
  - 1999, mars climate orbiter, two libraries using incompatible measurement units. $370M

- Financial Impact
  - A study at Cambridge University in 2013
  - The global cost of software bugs is
  - . . . around 312 billion of dollars annually

Poor President Obama…

"HealthCare.gov Progress & Performance Report", CMS.gov, 12/4/13

Yale

How can you avoid infamy?

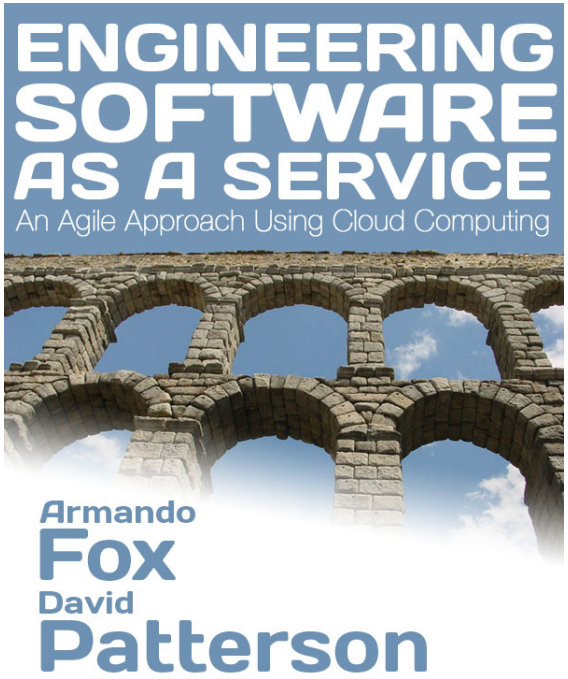
- Lessons from 60 years of SW development
- This class will review many approaches in lecture, listing pros and cons
- *Understand that software engineering is more than just programming*

- *#1 advice from previous projects:* ***"Trust the process & follow it"***

Yale

Do these things now

- Get book: http://saasbook.info
  - ***First Edition*** (≥ v1.2.0)
  - Kindle $10, print+kindle $37
- **Before next week!**
  - Think about projects!
  - Signup for Piazza
  - Signup for GitHub (Source Control)
    - https://education.github.com/
  - Signup for Cloud9 (IDE)
  - Signup for Heroku (Deployment)



Yale

Why this book?

- **Up-to-date**: Modern techniques, tools, and methods most widely used by the world's leading software companies. Brad Green, Engineering Manager at Google, says *"I'd be far more likely to prefer graduates of this program than any other I've seen."* Jacob Maine, Development & Operations Engineer at Pivotal Labs, calls it *"A great handbook for learning what it takes to actually deploy production quality software."*
- **Learn by doing**: Every concept is accompanied by hands-on examples using Ruby, Rails, and cloud computing. Fred Brooks, Turing Award winner and software engineering legend, says *"It is a pleasure to see a student text that emphasizes the production of real useful software."*
- **Comprehensive**: While focused on Agile and SaaS, the book also compares them with "plan and document" methodologies (waterfall, spiral, RUP, etc.) so students learn to identify which ones are appropriate in which scenarios.

Yale

Online Resources

http://www.saasbook.info/students

- Tutorials on other tools (GitHub, etc.)
- Cucumber tutorials
- Cloud9 tutorials
- Heroku tutorials
- Language resources:
  - Online "get started" tutorials on Ruby
  - Pointers to online HTML/CSS tutorials
- Feel free to suggest more on Piazza

Yale

Other Books

- Steve McConnell: "Code Complete: A Practical Handbook of Software Construction", ISBN-10: 0735619670
- Roger Pressman: "Software Engineering: A Practitioner's Approach", ISBN-10: 0073375977
- Ian Sommerville: "Software Engineering", ISBN-10: 0137035152
- Frederick Brooks: “The Mythical Man-Month”, ISBN 0-201-83595-9

Yale

Course Organization

- Grading—approximate breakdown
  - 20% homeworks
  - 5% in-lecture *microquizzes (graded on participation)*
  - 10% Midterm exam
  - 10% Final exam
  - 50% Projects
  - 5%: Participation and Altruism
- A typical week
  - Monday eve: **read/watch stuff before lecture**
  - Tuesday: project meetings
  - Wednesday eve: **read/watch more stuff before lecture**
  - Thursday: project meetings
  - Thursday 11:59pm. Typically a HW or milestone will be due
  - Friday : lecture

Yale

Projects

- Some smaller projects / homeworks
  - Get feed wet
  - Learn tools, learn languages
  - Experience the process a couple of times
- One Big Project
  - Can be (almost) anything
  - Done in teams of 6-7 students
  - You do everything
  - Gather requirements, design, code, and test
  - This class should be very close to a startup experience

Yale

Project Timeline

- Project nominations
  - Start thinking about the project proposal **today**
  - Nomination will be due a few days after next class
  - More detailed instruction next week
- Project selection, team assignments – next week
- Projects will be reviewed and analyzed by others teams (and the instructors)
- Requirements and specification
- Project design & plan
- Design review - Done by other teams
- Testing - Performed by other teams (and the instructors)

Yale

Project Fair

Yale

Project Structure

- We will simulate the “real world”
- Maintaining/extending other people’s code
- Need for
  - Specifications
  - Interfaces
  - Documentation
- Importance of institutional knowledge
- You might be randomly assigned to a different team!

Yale

Things we won't do

- It is not a course about learning a new language
  - But if you don't have any experience in Ruby on Rails or Scala/ Play, you will learn a new language (languages)
  - Javascript
  - Version Control (git)
- Don't expect to learn programming tricks
  - But you'll learn techniques for "programming in the large"
- Don't expect to learn management skills from the lectures
  - Some things you learn by doing, not through lectures!

Academic Integrity

- Academic Integrity at Yale
- Don’t use work from uncited sources
- *You can learn more about the conventions of using sources by referring to the Yale College Writing Center's Web site* (from the Academic Integrity at Yale web site)

Yale

Academic Integrity

- *Collaboration* is OK and encouraged
- … but not on exams!
- Default penalty: failing the class

- If you wouldn't want fellow students, parents, or professors to know about it, it's probably cheating
- If it feels like cheating, it probably is
- Your instructors join you in pledging to adhere to this code

Yale

Pause

Yale

The Myth of Multitasking

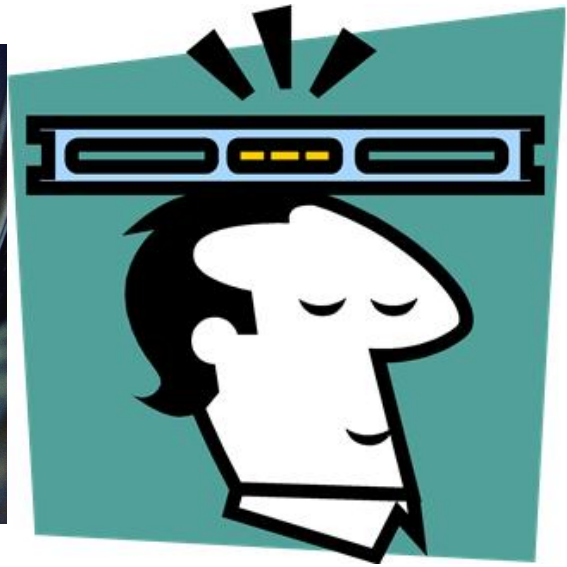
- https://www.youtube.com/watch?v=PriSFBu5CLs&t=7m50s
- (Clifford Nass, Stanford)
- Allowing yourself to respond to distraction (incoming email, IM, etc.) triggers small dopamine release
- Over time, you get addicted to it
- Result: Multitaskers waste far more brainpower than monotaskers when actually distracted
- Long-term effects can be hard to reverse

Yale

The Rules
(and we really mean it!)

Yale

Plan-and-Document processes: Waterfall vs. Spiral



(*Engineering Long Lasting Software,* §1.2)

David Patterson

Yale

How to Avoid SW Infamy?

- Can't we make building software as predictable in schedule and cost and quality as building a bridge?
- If so, what kind of development process to make it predictable?

Which aspect of the software lifecycle consumes the most resources?

- ☐ Design
- ☐ Development
- ☐ Testing
- ☐ Maintenance

Yale

Software Engineering

- Bring engineering discipline to SW
- Term coined ~ 20 years after 1st computer
- Find SW development methods as predictable in quality, cost, and time as civil engineering

Yale

Software Engineering **Myths**

Management

- "We have books with rules. Isn't that everything my people need?"
  - Which book do you think is perfect for you?
- "If we fall behind, we add more programmers"
  - "Adding people to a late software project, makes it later"
    – Fred Brooks (The Mythical Man Month)
- "We can outsource it"
  - If you do not know how to manage and control it internally, you will struggle to do this with outsiders

Yale

Software Engineering **Myths**

Customer

- "We can refine the requirements later"
  - A recipe for disaster

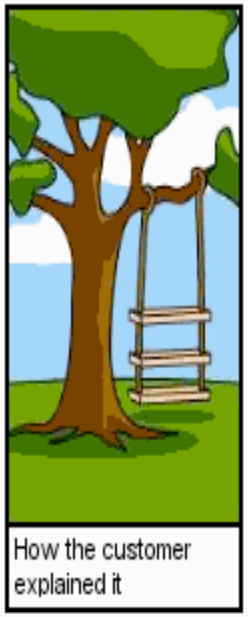
- "The good thing about software is that we can change it later easily"
  - As time passes, cost of changes grows rapidly

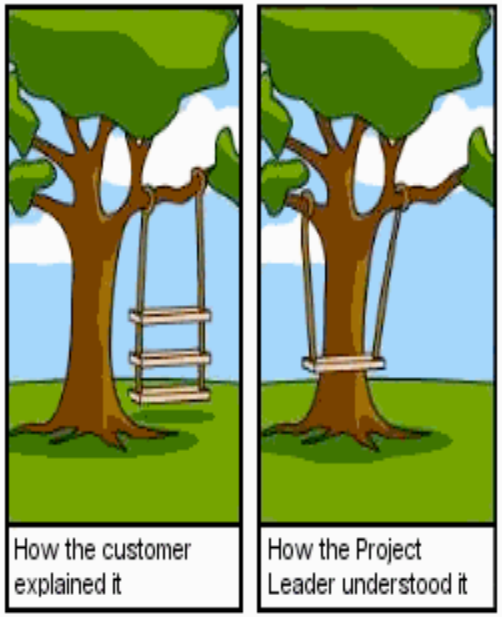
Yale

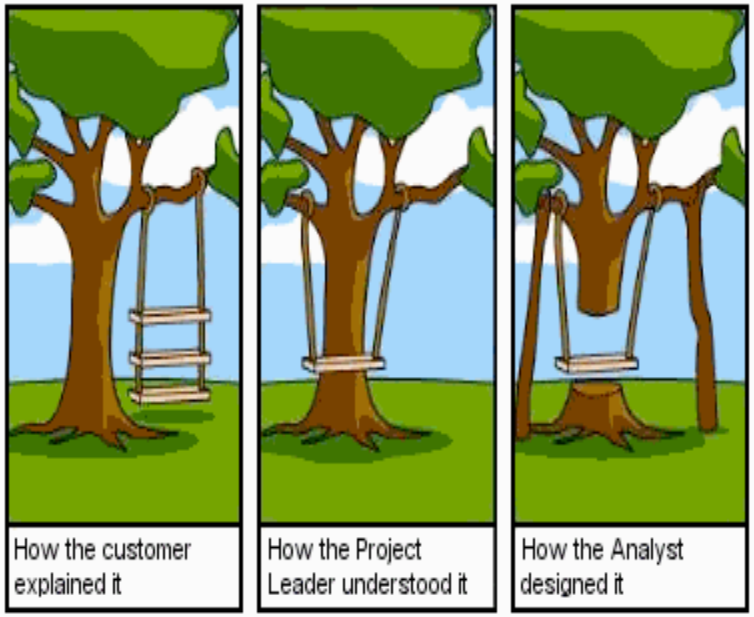
Software Engineering **Myths**

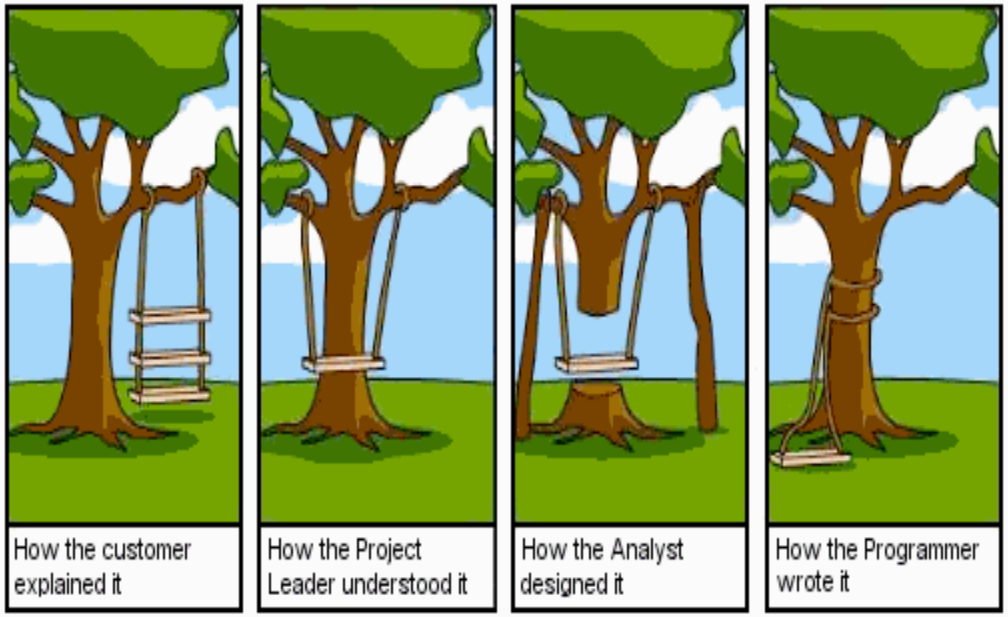
Practitioner

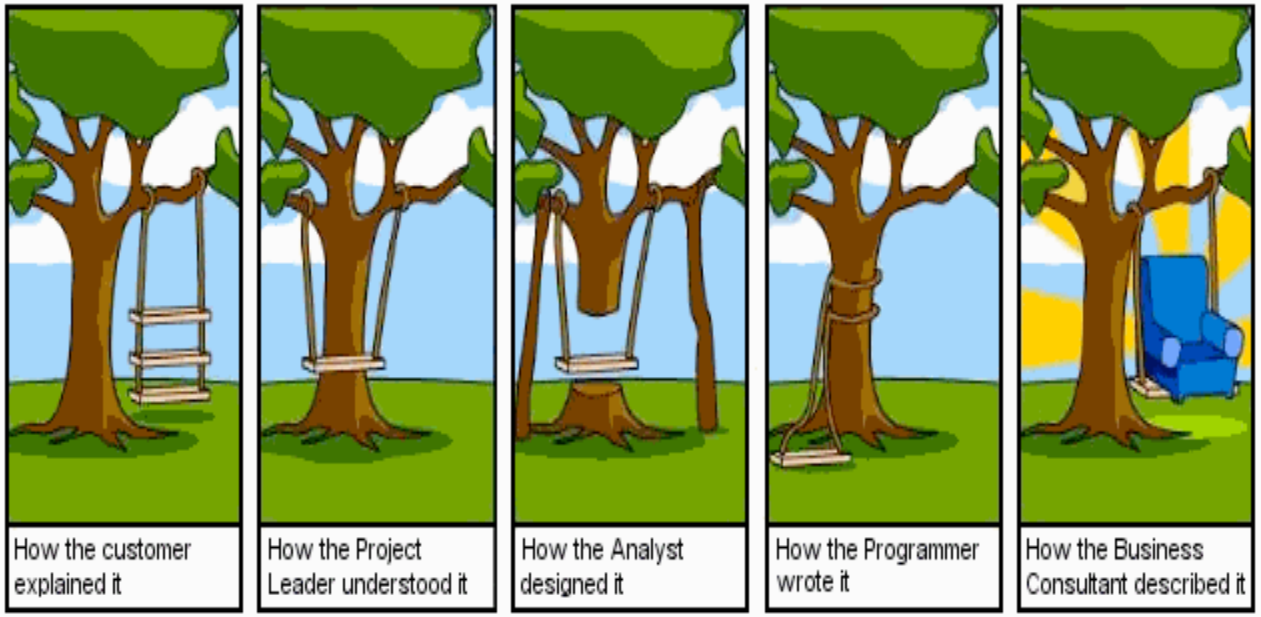
- "Let's write the code, so we'll be done faster"
  - "The sooner you begin writing code, the longer it'll take to finish"
  - 60-80% of effort is expended after first delivery
- "Until I finish it, I cannot assess its quality"
  - Software and design reviews are more effective than testing (find 5 times more bugs)
- "There is no time for software engineering"
  - But is there time to redo the software?

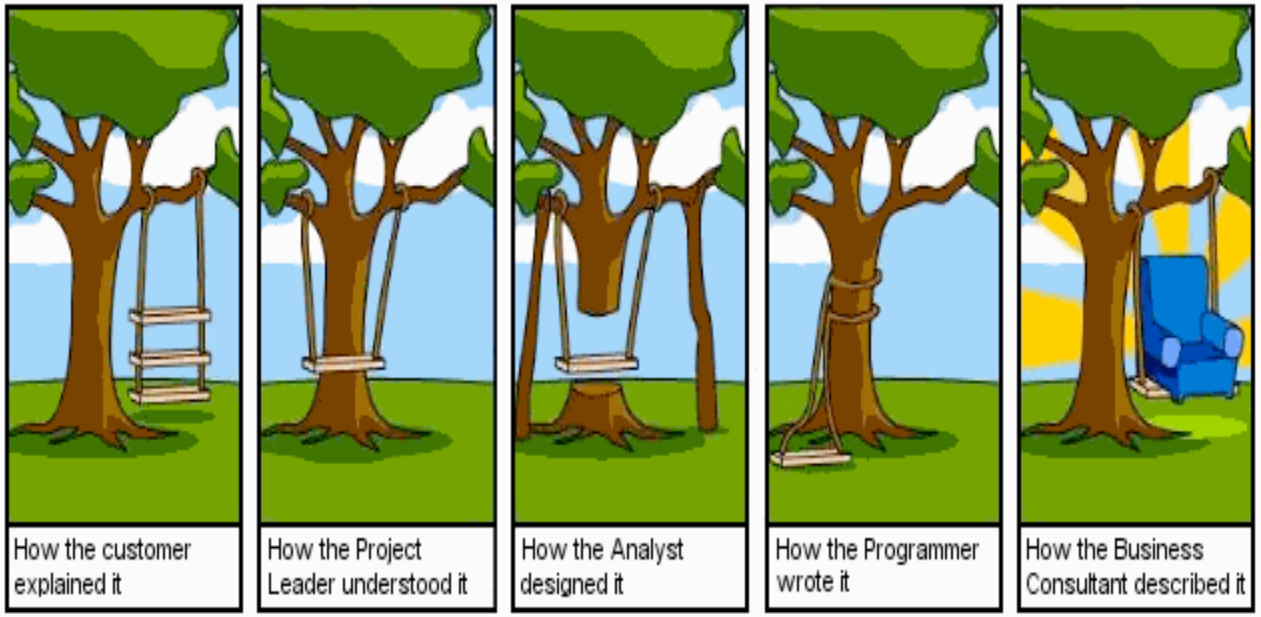

How the customer
explained it

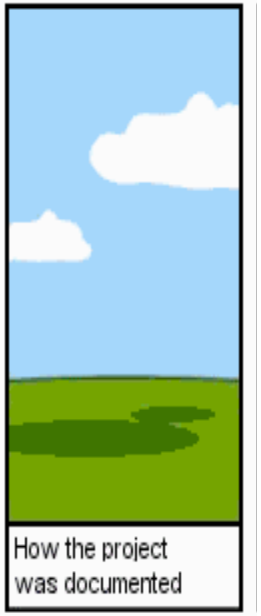

How the customer explained it
How the Project Leader understood it

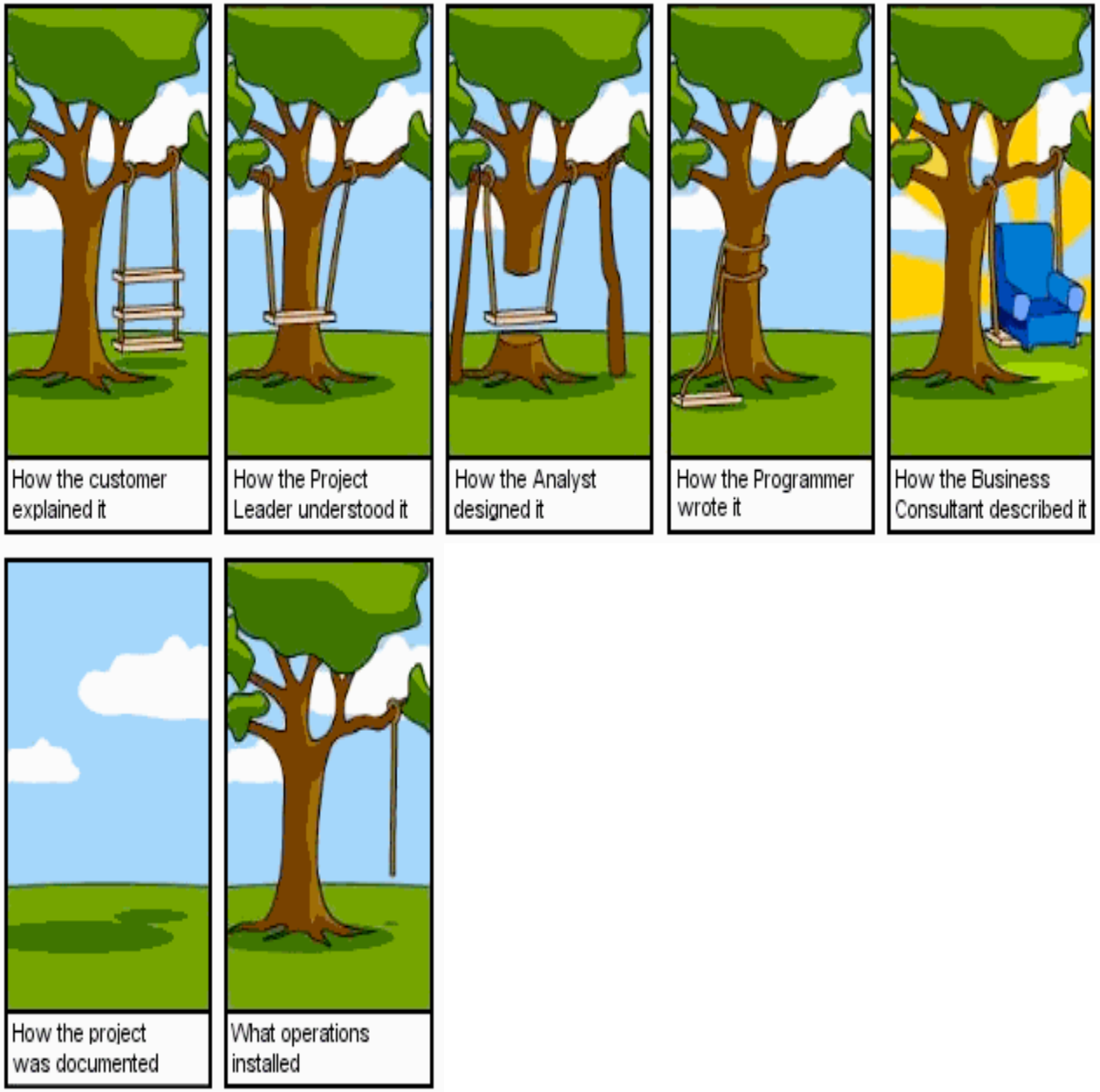

How the customer explained it
How the Project Leader understood it
How the Analyst designed it

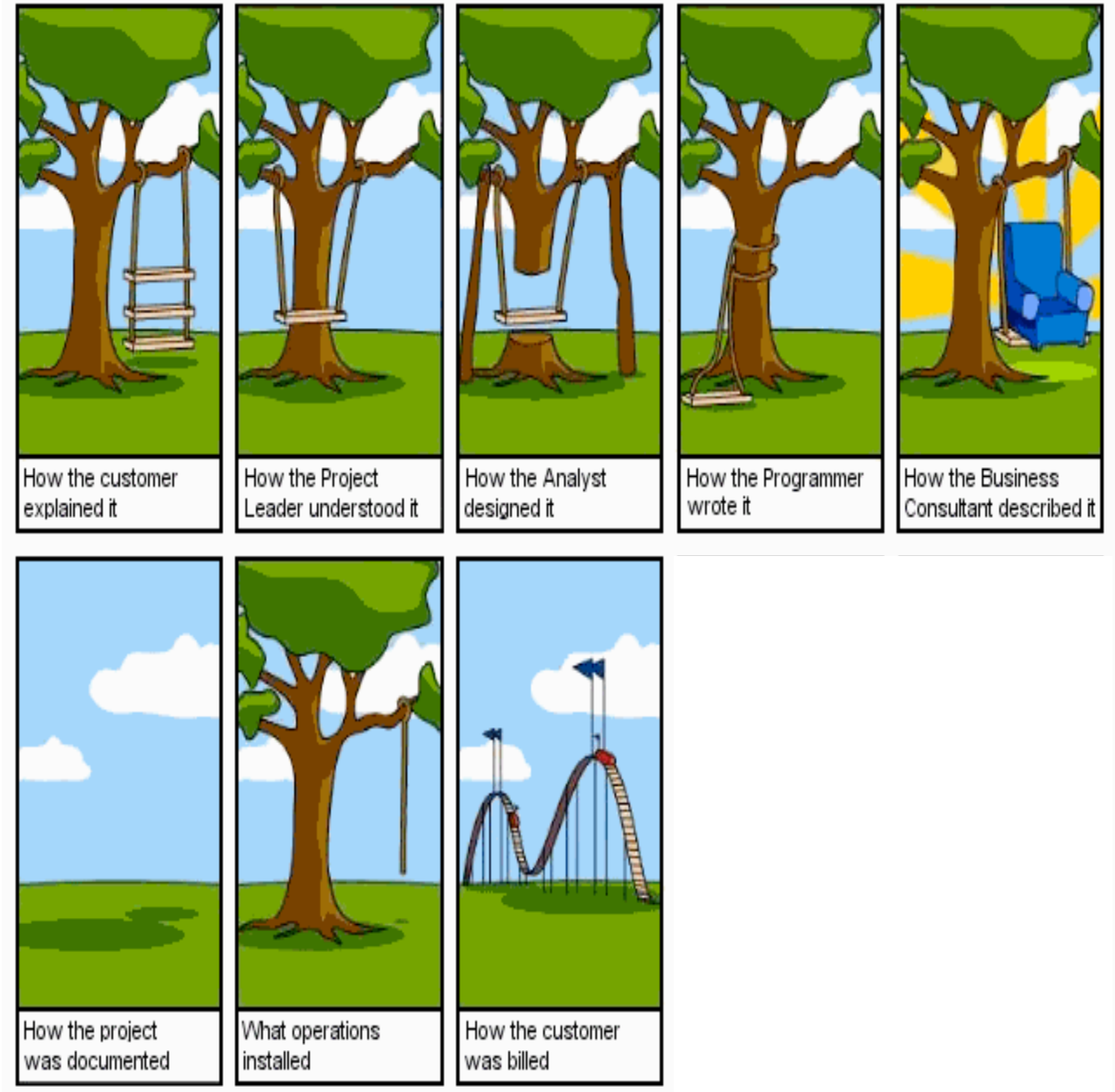

How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it

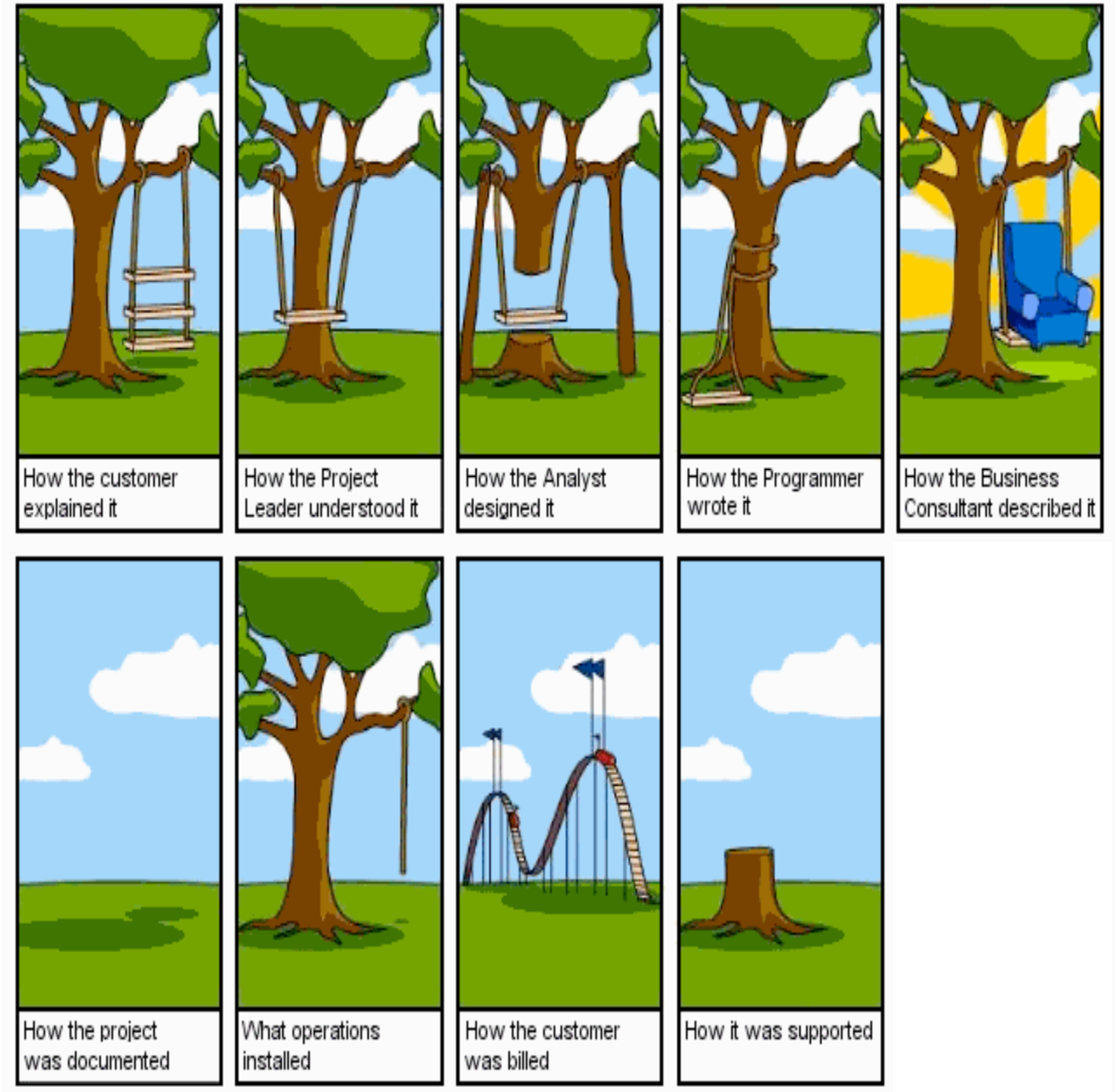

How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it

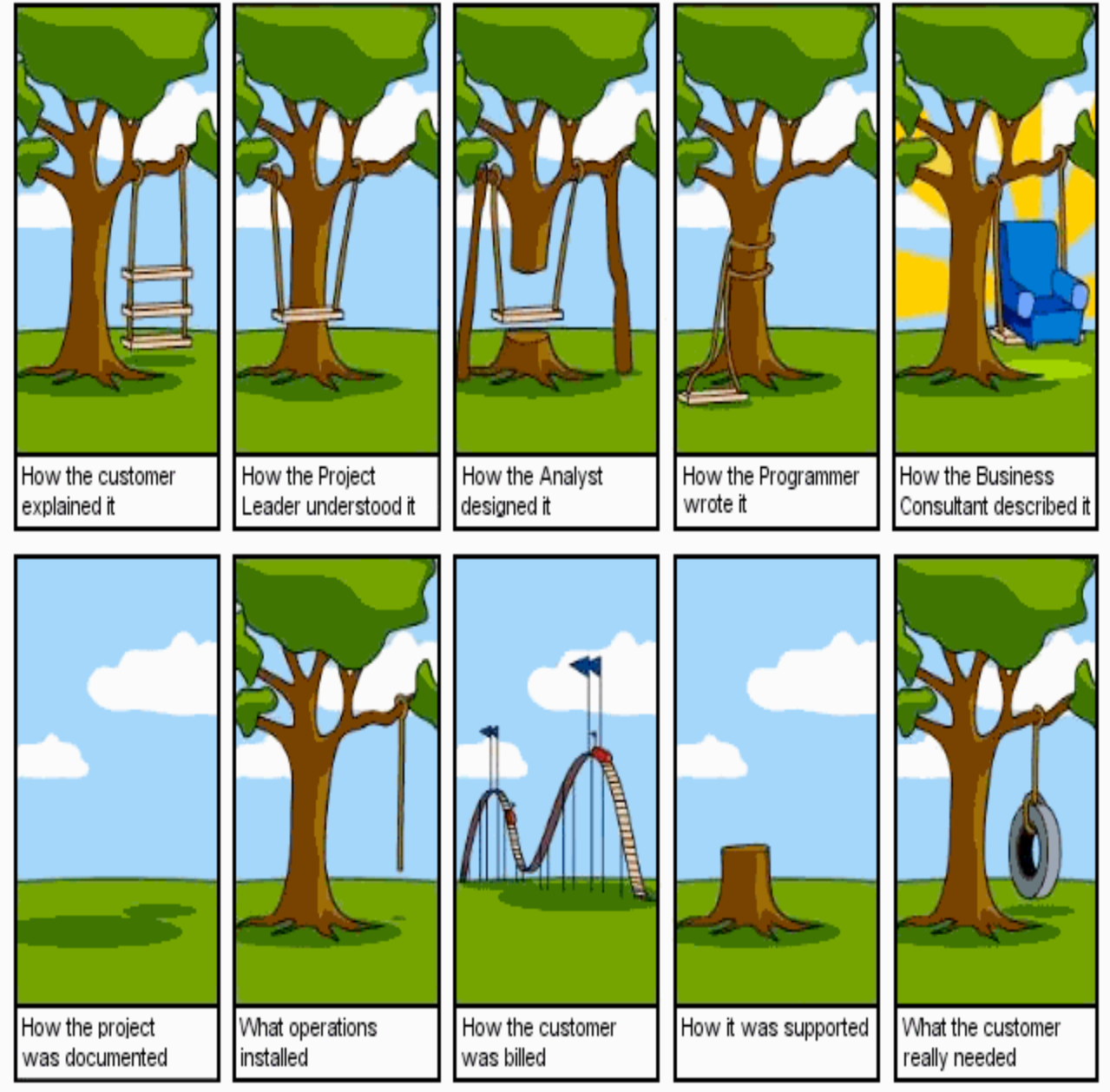

How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it


How the project was documented


How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed


How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed


How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported


How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed

Software Engineering

Methods

Yale

Plan-and-Document

- Before coding, project manager makes plan
- Write detailed documentation all phases of plan
- Progress measured against the plan
- Changes to project must be reflected in documentation and possibly to plan

1st Development Process: Waterfall (1970)

- 5 phases of Waterfall "lifecycle"

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

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

Yale

How well does Waterfall work?

- *And the users exclaimed with a laugh and a taunt: "It's just what we asked for, but not what we want." —Anonymous*
- *"Plan to throw one [implementation] away; you will anyhow."*
  - Fred Brooks, Jr. (1999 Turing Award winner)
- Often after build first one, developers learn right way they should have built it

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

Yale

P&D depends heavily on Project Managers

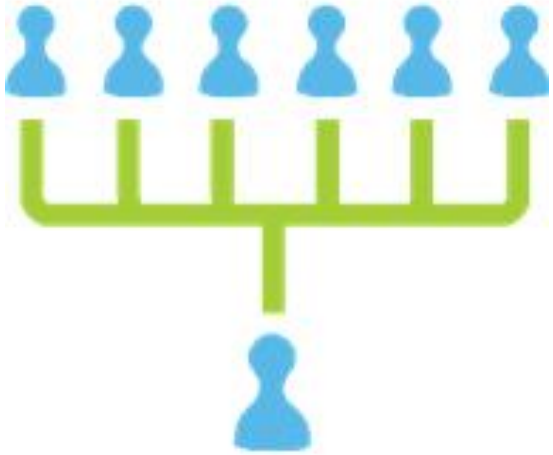
- P&D depends on Project Managers
  - Write contract to win the project
  - Recruit development team
  - Evaluate software engineers performance, which sets salary
  - Estimate costs, maintain schedule, evaluate risks & overcomes them
  - Document project management plan
  - Gets credit for success or blamed if projects are late or over budget

Yale

P&D Team Size

"Adding manpower to a late software project makes it later."

Fred Brooks, Jr.,
The Mythical Man-Month



- It takes time for new people to learn project
- Communication time grows with size, leaving less time for work
- Groups 4 to 9 people, but hierarchically composed for larger projects

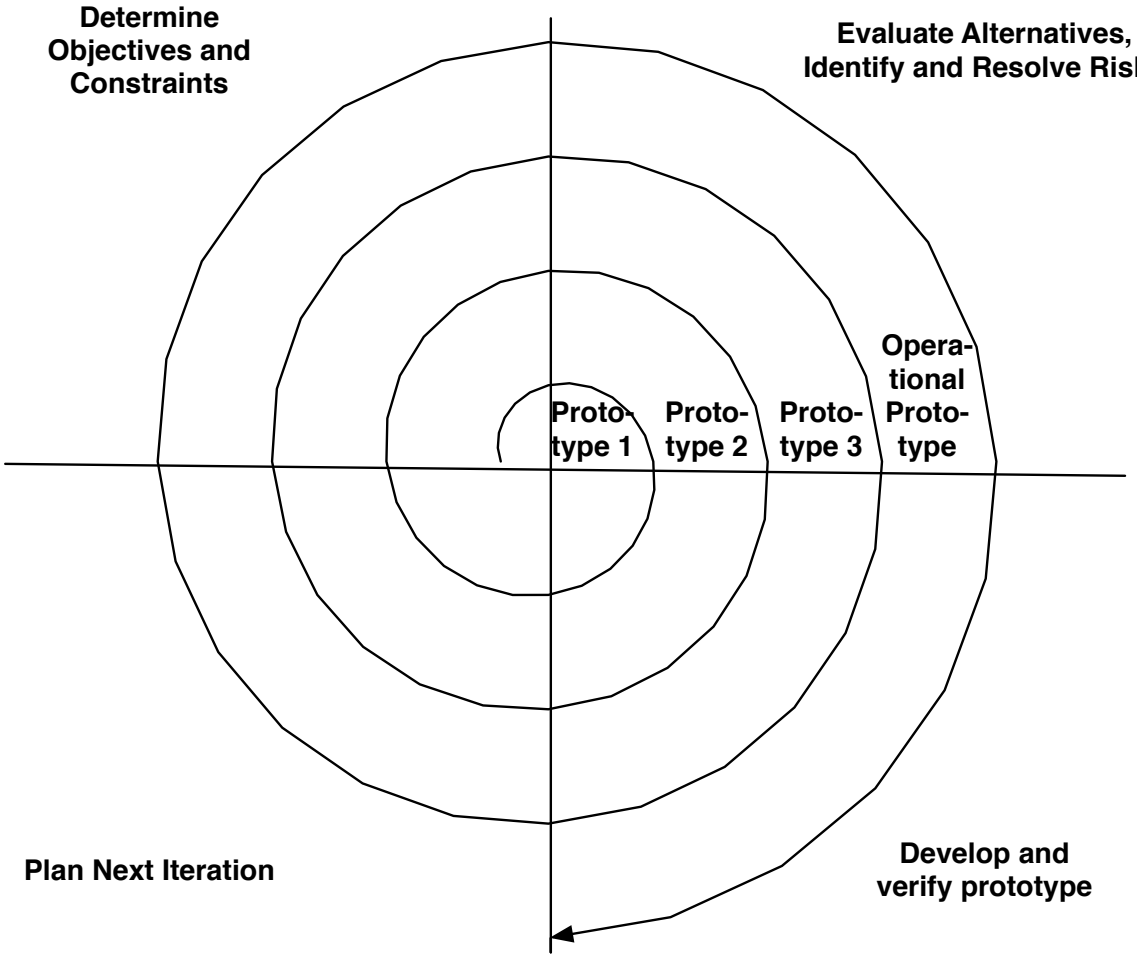
Yale

Spiral Lifecycle (1986)

- Combine Plan-and-Document with prototypes
- Rather than plan & document all requirements 1st, develop plan & requirement documents across each iteration of prototype as needed and evolve with the project

Yale

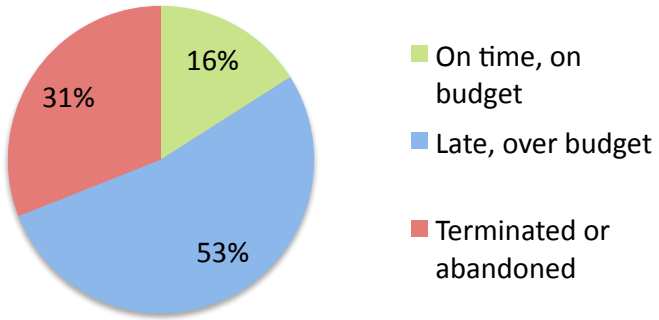
Spiral Lifecycle



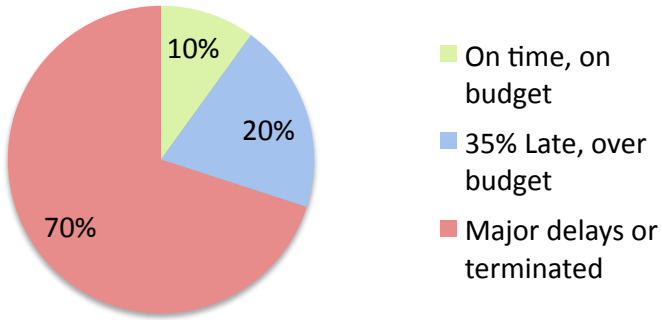
How Well do Plan-and-Document Processes Work?

- IEEE Spectrum "Software Wall of Shame"
  - 31 projects: lost $17B

Software Projects (Johnson 1995)

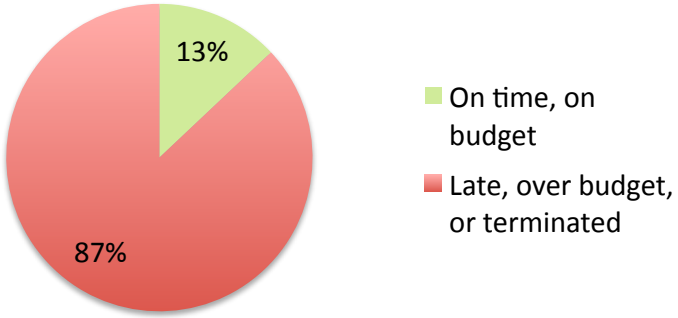


Software Projects (Jones 2004)



3X

Software Projects (Taylor 2000)



3/~500 new development projects on time and budget

(Figure 1.6, *Engineering Long Lasting Software* by Armando Fox and David Patterson, 2nd Beta edition, 2013.)

Yale

Alternative Process?

- How well can Plan-and-document hit the cost, schedule, & quality target?
- P&D requires extensive documentation and planning and depends on an experienced manager
  - Can we build software effectively without careful planning and documentation?
  - How to avoid “just hacking”?

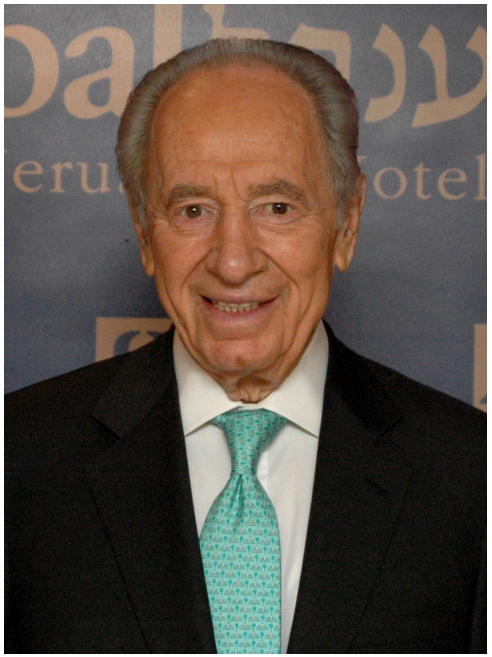
Yale

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)

Yale

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."

Yale

"Extreme Programming" (XP) version of Agile lifecycle

- If short iterations are good, make them as short as possible (weeks vs. years)
- If simplicity is good, always do the simplest thing that could possibly work
- If testing is good, test all the time. Write the test code before you write the code to test.
- If code reviews are good, review code continuously, by programming in pairs, taking turns looking over each other's shoulders.
- **But you have to do all of them.**

Yale

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 ~1 to 2 weeks)
- Agile emphasizes Test-Driven Development (TDD) to reduce mistakes, written down User Stories to validate customer requirements, Velocity to measure progress

Yale

Agile Then and Now

- Controversial in 2001
  - "… yet another attempt to undermine the discipline of software engineering… nothing more than an attempt to legitimize hacker behavior."

    – Steven Ratkin, "Manifesto Elicits Cynicism,"

    IEEE Computer, 2001
- Accepted in 2013
  - 2012 study of 66 projects found majority using Agile, even for distributed teams

Yale

Fallacies and Pitfalls

- Fallacy: The Agile lifecycle is best for all software development
  - Good match for some SW, especially SaaS
  - But not for NASA, code subject to regulations
- Per topic, will practice Agile way to learn but will also see Plan & Document perspective
  - Note: you will see new lifecycles in response to new opportunities in your career, so expect to learn new ones

Yale

Yes: Plan-and-Document
No: Agile (Sommerville, 2010)

1. Is specification required?
2. Are customers unavailable?
3. Is the system to be built large?
4. Is the system to be built complex (e.g., real time)?
5. Will it have a long product lifetime?
6. Are you using poor software tools?
7. Is the project team geographically distributed?
8. Is team part of a documentation-oriented culture?
9. Does the team have poor programming skills?
10. Is the system to be built subject to regulation?

Yale

Which does Agile *not* use that P&D processes *do* use?

- Requirements elicitation
- Documentation
- Progress estimation
- Unit & functional testing
- System/integration testing
- User acceptance testing
- Continuous refactoring of design

Yale

Software as a Service (SaaS) and Service-Oriented Architecture (SOA)

(*Engineering Software as a Service* §1.4-1.5)

David Patterson

Yale

Why is SaaS > SWS?

1. No install worries about HW capability, OS
2. Data stored safely, persistently on servers
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

Yale

Shrink-wrapped software (SWS)

- client-specific binary, frequent upgrades
  - Must work w/many versions of HW, OS, Libraries, …
  - Hard to maintain
  - Extensive compatibility testing per release
- Alternative: server-centric app, thin client
  - Search, email, commerce, social nets, video…
  - Now also productivity (Google Docs/Office 365), finance (TurboTax Online), IDEs (Codenvy)…
  - Your instructors believe this is future of SW

Yale

Building large SaaS?

- Can you design software so that you can recombine independent modules to offer many different apps without a lot of programming?
  - Solves "Agile only good for small teams"
- "[Amazon CEO Jeff Bezos] realized long before the vast majority of Amazonians that Amazon needs to be a platform."
  Steve Yegge, Googler, former Amazonian, in a 2011 blog post

Yale

2002: 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.”

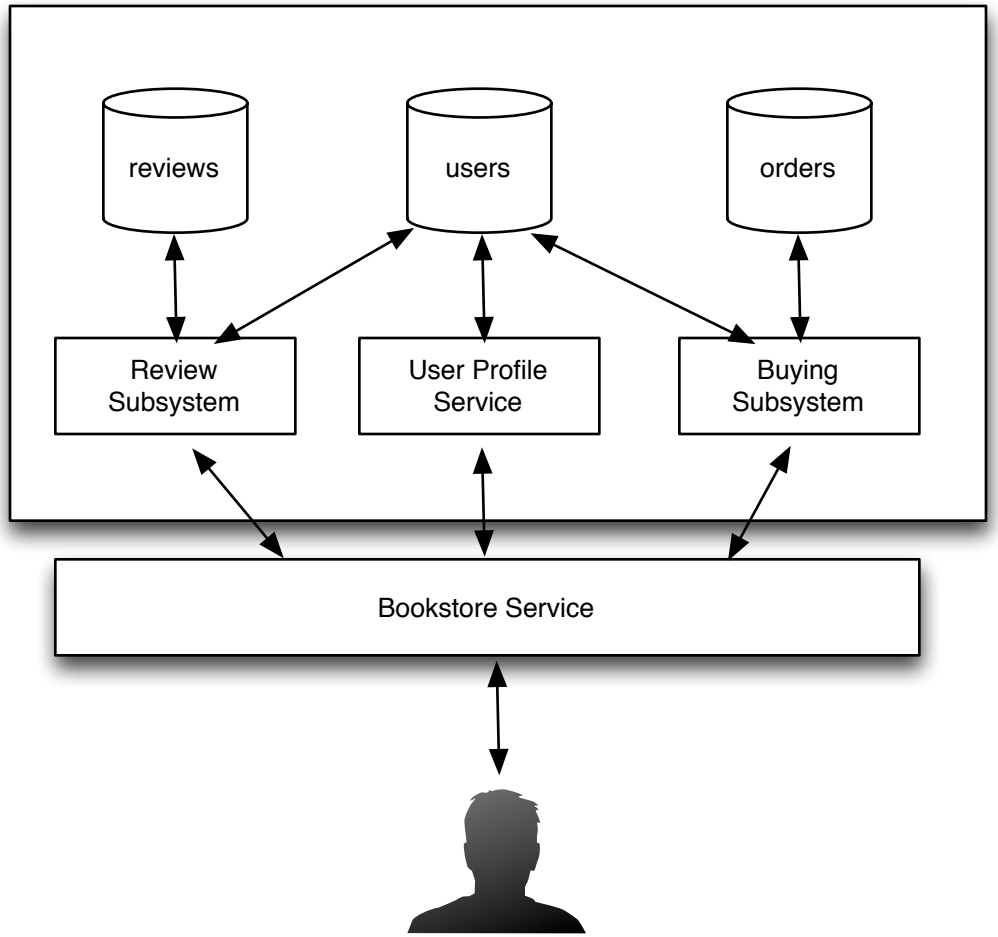
Yale

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

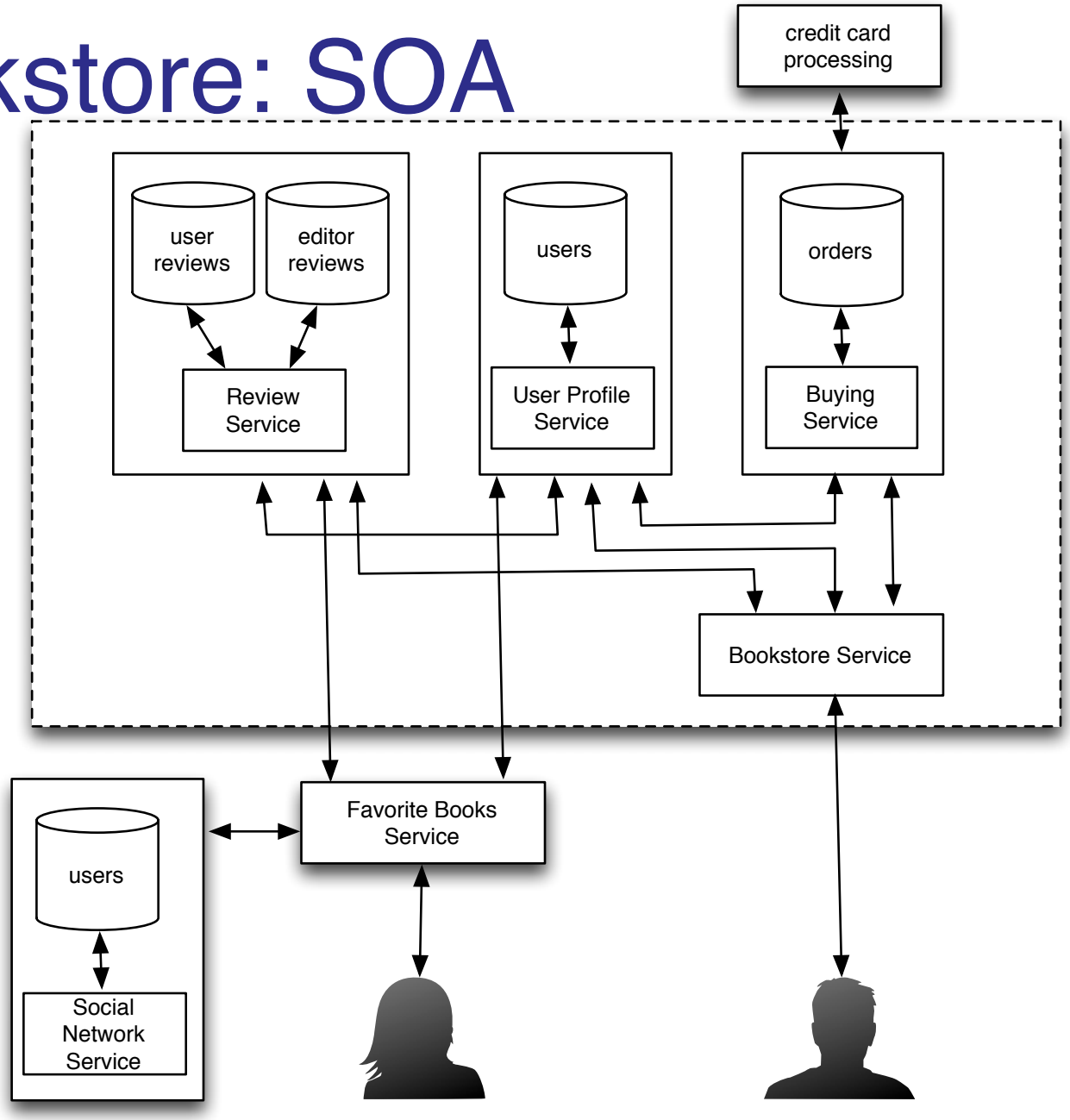
- Internal subsystems can share data directly
  - Review access user profile
- All subsystems inside single API ("Bookstore")



(Figure 1.7, *Engineering Software as a Service* by Armando Fox and David Patterson, 2014.)

Bookstore: SOA

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



((Figure 1.7, *Engineering Software as a Service* by Armando Fox and David Patterson, 2014.)

Yale

Which of the following is a *disadvantage* of SOA compared to a silo design?

- ☐ SOA may be harder to debug & tune
- ☐ SOA results in lower developer productivity
- ☐ SOA's complexity is a poor match for small teams
- ☐ SOA is more expensive to deploy than Silo, because more servers are needed to handle the same workload

Yale

Jump

(*Engineering Software as a Service* §1.6)

David Patterson

Yale

Ideal hardware infrastructure for SaaS?

- 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

Yale

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

Yale

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

Yale

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
  - Amazon runs its own e-commerce on AWS!

Yale

2016 AW… Prices

General Pu… stances

Elastic Block Storage
$0.10 per GB-month

Instance Type	Per Hour	$ Ratio to Nano	vCPUs	Compute Units	Me… (GiB)	Storage (GB)
t2.nano	$0.007	1	1	Variable	0.5	EBS
t2.micro	$0.013	2	1	Variable	1	EBS
t2.small	$0.026	4	1	Variable	2	EBS
t2.medium	$0.052	8	2	Variable	4	EBS
t2.large	$0.104	16	2	Variable	8	EBS
m4.large	$0.120	18	2	7	8	EBS
m4.xlarge	$0.239	37	4	13	16	EBS
m4.2xlarge	$0.479	74	8	26	32	EBS
m4.4xlarge	$0.958	147	16	54	64	EBS
m4.10xlarge	$2.394	368	40	125	160	EBS
m3.medium	$0.067	10	1	3	4	1 x 4 SSD
m3.large	$0.133	20	2	7	8	1 x 32 SSD
m3.xlarge	$0.266	41	4	13	15	2 x 40 SSD
m3.2xlarge	$0.532	82	8	26	30	2 x 80 SSD

Yale

Higher-level services built on top of elastic clouds ("utility computing")

- Apps, eg FarmVille on AWS
  - Prior biggest online game 5M users
  - 4 days =1M; 2 months = 10M; 9 months = 75M
- Heroku (platform/deployment)
- CodeClimate (code analysis and quality as a service)
- Travis CI (testing as a service)
- Pivotal Tracker (Agile project management)
- GitHub (source & configuration management)

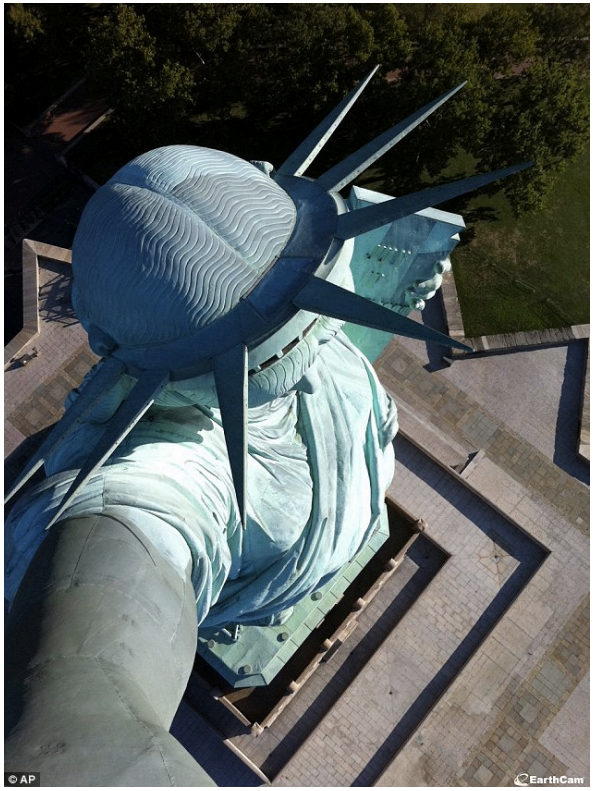
Yale

Which statement about private datacenters vs. public utility cloud computing (such as AWS) is true?

- ☐ Private datacenters are not shared by multiple companies/competitors
- ☐ Private datacenters may be the only option for some highly-regulated apps
- ☐ Private datacenters are inherently more secure than public utility computing
- ☐ Private datacenters could match the low cost of public utility computing if they just used the same type of hardware and software

Yale

Legacy SW vs. Beautiful SW and Software Quality



(*Engineering Software as a Service* §1.7-1.8

David Patterson

In general, which statement regarding the relationship between bug fix costs and enhancement costs is most accurate?

- ☐ $(Bug fixing) ≥ ~2x $(Enhancing)
- ☐ $(Bug fixing) ≈ $(Enhancing)
- ☐ $(Enhancing) ≈ ~2x $(Bug fixing)
- ☐ $(Enhancing) ≈ 3-4x $(Bug fixing)

Yale

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
- *Vital but ignored topic in most SWE courses*
- Contrasts with **beautiful code**: meets customers' needs and easy to evolve

Yale

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

Yale

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

Yale

Exhaustive testing is infeasible

- Divide and conquer: perform different tests at different phases of SW development
  - Upper level doesn't redo tests of lower level
- *Coverage:* various measurements of what % of system is "exercised" by test suite

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

Yale

Question: Which statement is NOT true about testing?

- ☐ Tests that have outlived their usefulness should be discarded
- ☐ While difficult to achieve, 100% test coverage ensures correct implementation
- ☐ Higher level tests typically delegate more detailed testing to lower levels
- ☐ Unit testing works within a single class and module testing works across classes

Yale

And in Conclusion

- CPSC639: SW Eng. Principles via Cloud app by team for customer + enhancing legacy app
- Agile vs. Plan & Document: Small teams, quick iterations w/customer feedback, to reduce risk of building the wrong thing
- SaaS vs: SWS: less hassle for developers & users
- Service Oriented Architecture: build large systems by combining smaller standalone services➔reuse!
- Scale led to savings/CPU => reduced cost of Cloud Computing => Utility Computing
- Testing to assure software quality, which means good for customer *and* developer

Yale

Discussion

Yale

Language

- This course is not a “how to program in X” course
- We will work with multiple languages
- This is the reality of the software environment
- Choice of Language of Focus

- The book is in Ruby
- Widely used in industry
- Works out-of-the-box with many frameworks/tools

Yale

- Exciting new language
- Bleeding-edge
- LinkedIn social graph analysis
- Twitter migrated main message queue from Ruby to Scala for performance
- Actors / Akka
- Behind Apache Flint and Apache Spark (ML infrastructure)

Yale

Survey : Now

- Name?
- Years of programming?
- Have you worked on a “real” software product?
- How many CS courses have you taken?
- Languages you are comfortable with?
- Languages you would like to learn?
- Names of friends in the room? (for creating groups)

Yale

Applications

Some ideas:

- HealthCare systems
- Games – Oculus

Research-Oriented:

- Smart Contracts for Ethereum (like BitCoin)
- Concurrent editing (like GoogleDocs)

Yale

Survey : Take Home

- 2 domains of interest
  - e.g. healthcare
  - e.g.
- 2 specific project ideas
  - e.g.
  - e.g.

Yale

The End.