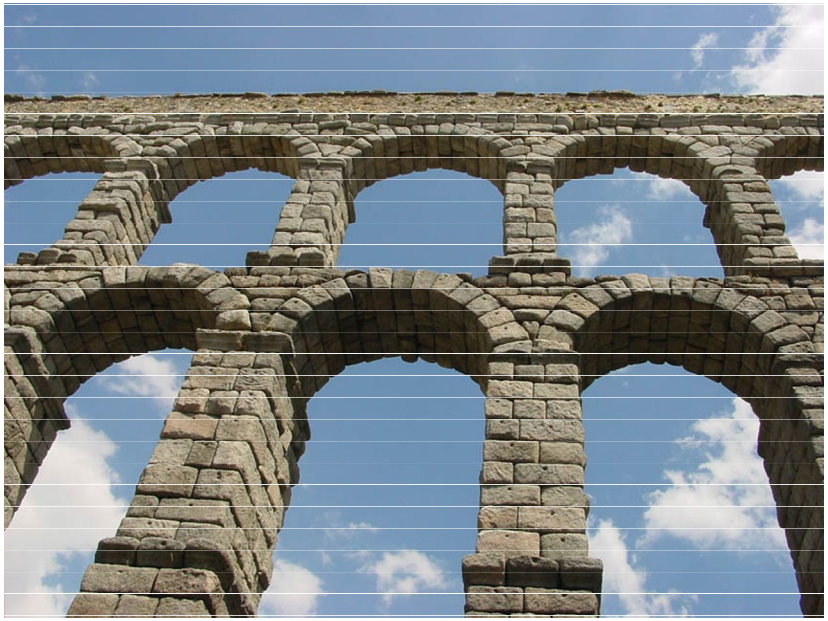
Introduction to Software Engineering



David Patterson

Ranking Top 200 Jobs (2012)

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

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

Ranking Top 200 Jobs (2012)

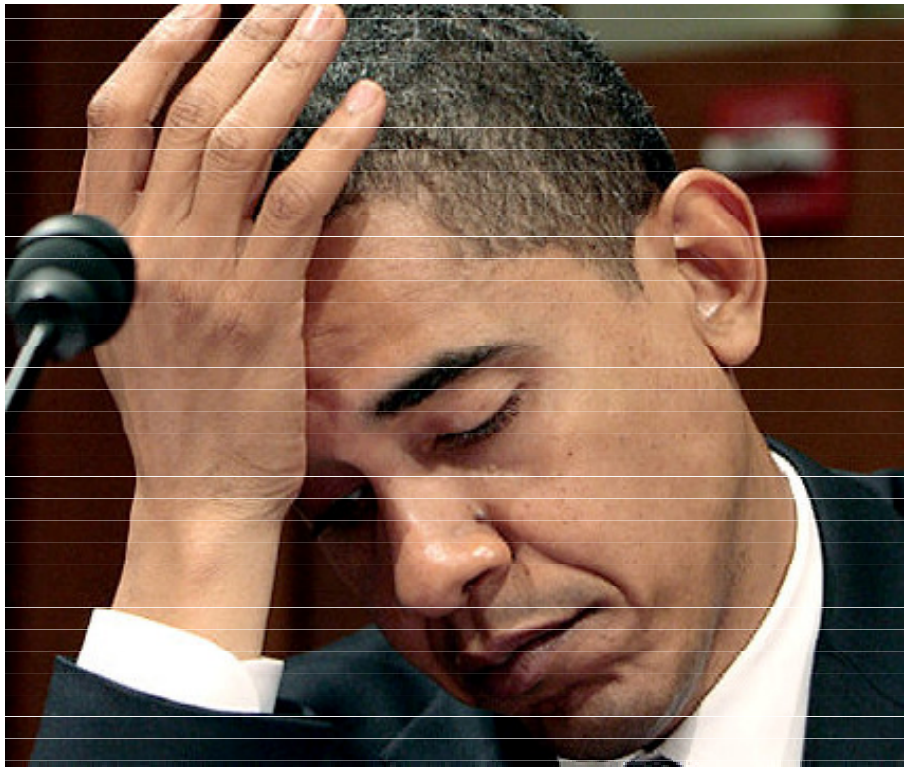
1. Software Engineer

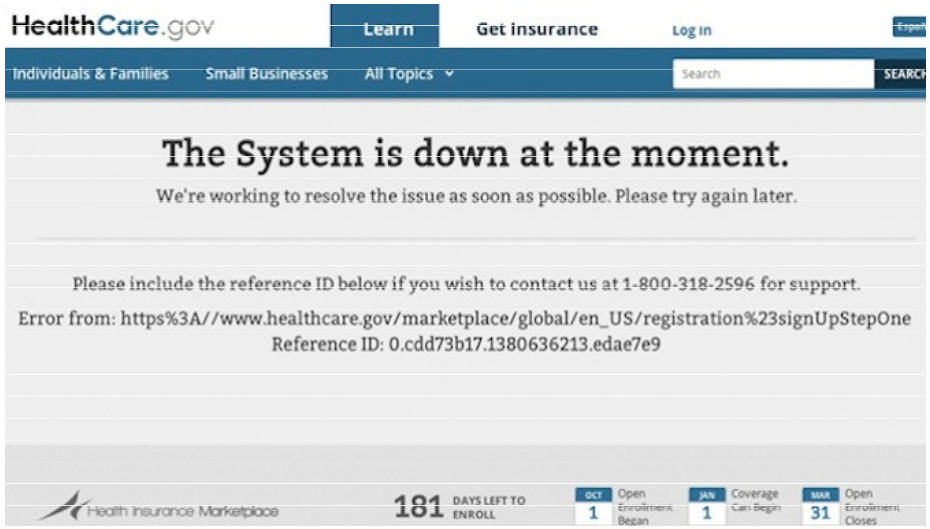
Researches, designs, develops and maintains software systems along with hardware development for medical, scientific, and industrial purposes. Income: $88,142 (+25%)

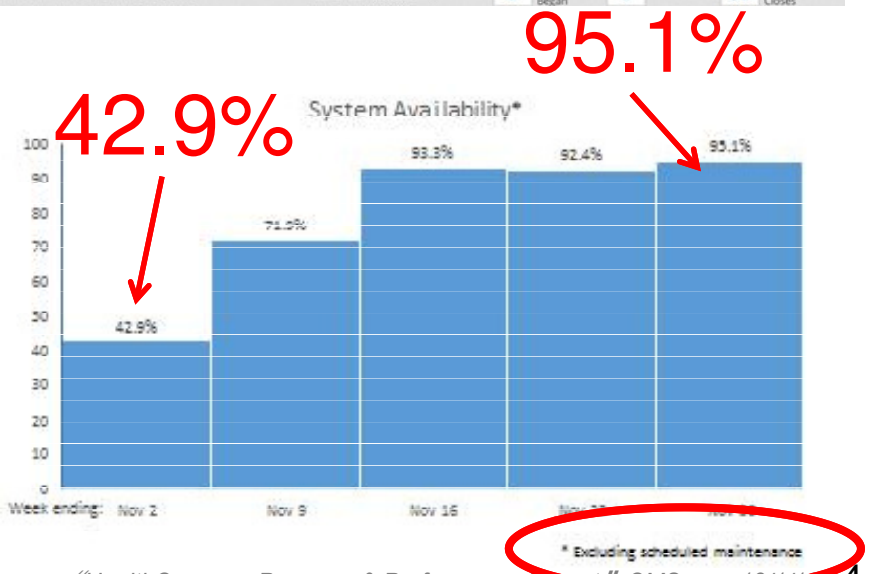
34. Programmer

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

Poor President Obama…







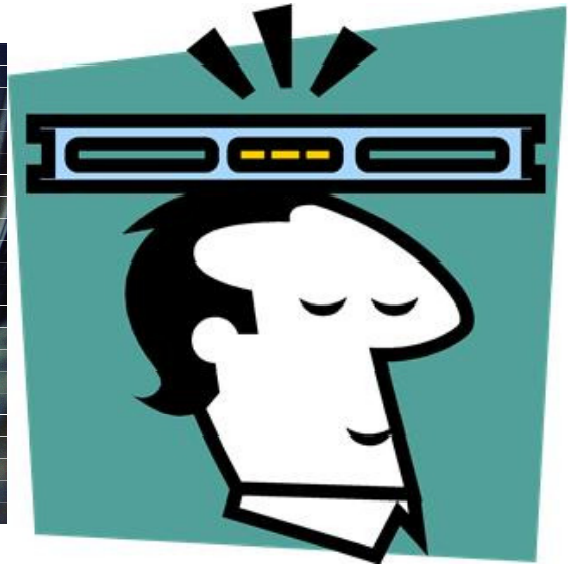
"HealthCare.gov Progress & Performance Report", CMS.gov, 12/4/13

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"***

Plan-and-Document processes: Waterfall vs. Spiral



David Patterson

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

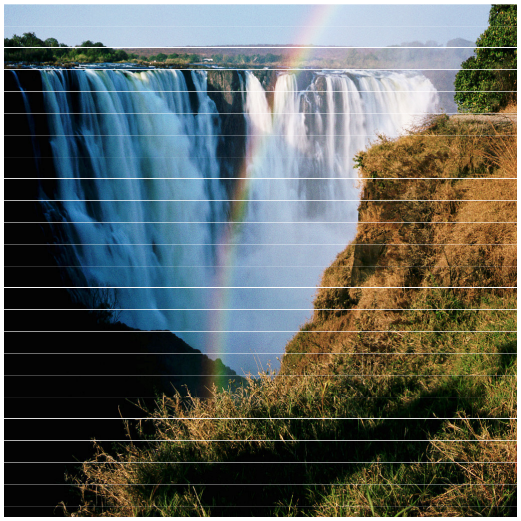
Software Engineering

- 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
- "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"

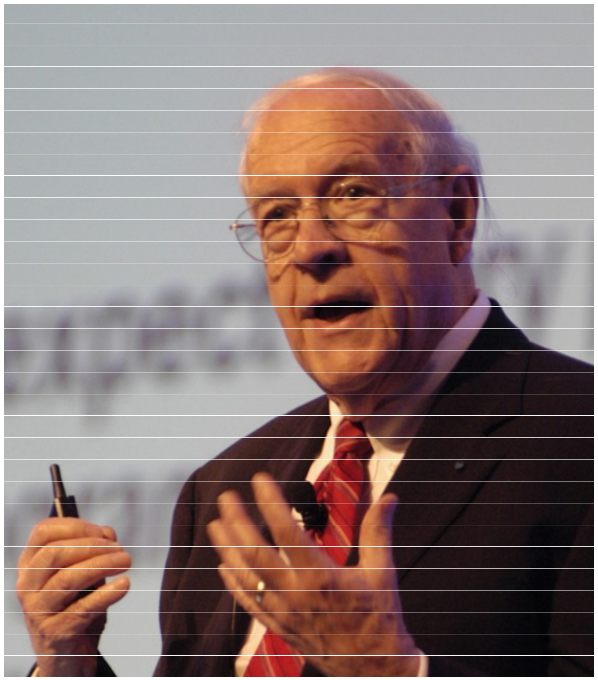
1. Requirements analysis & specification
2. Architectural design
3. Implementation & Integration
4. Verification
5. Operation & Maintenance



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

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.)

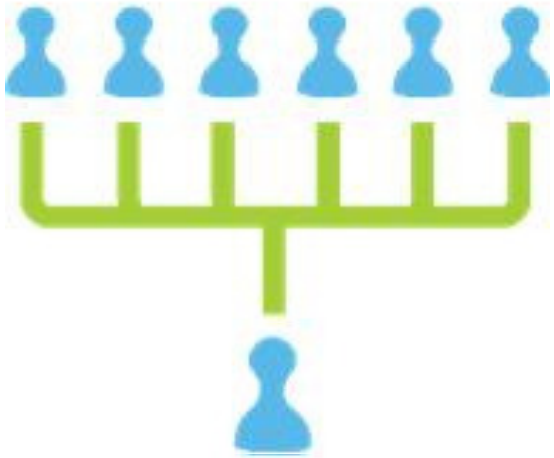
P&D depends heavily on Project Managers

- 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

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

Spiral Lifecycle (1986)

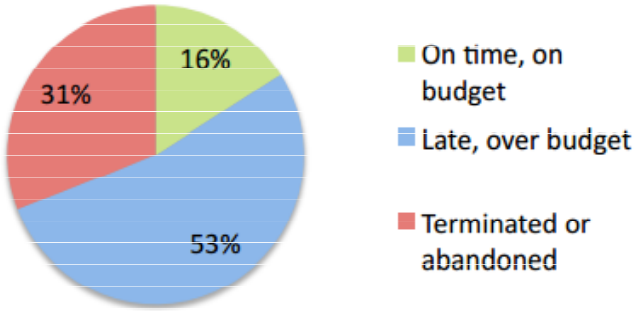
- Combine Plan-and-Document with prototypes
- Rather than plan & document all requirements 1st, develop plan & requirement documents across each iteration of prototype as needed and evolve with the project

Spiral Lifecycle

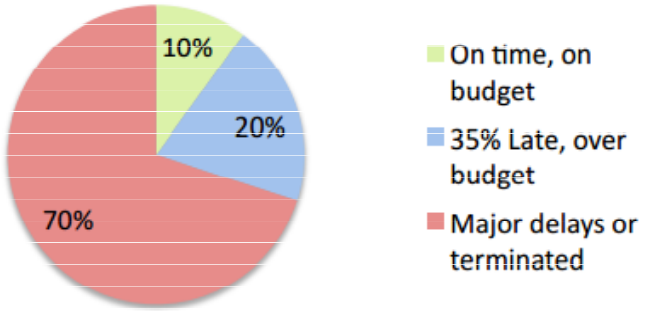
How Well do Plan-and-Document Processes Work?

- IEEE Spectrum "Software Wall of Shame"
  - 31 projects (4 from last lecture + 27): lost $17B

Software Projects (Johnson 1995)

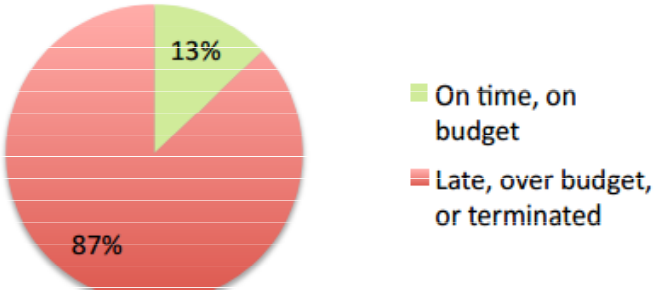


Software Projects (Jones 2004)



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.)

Alternative Process?

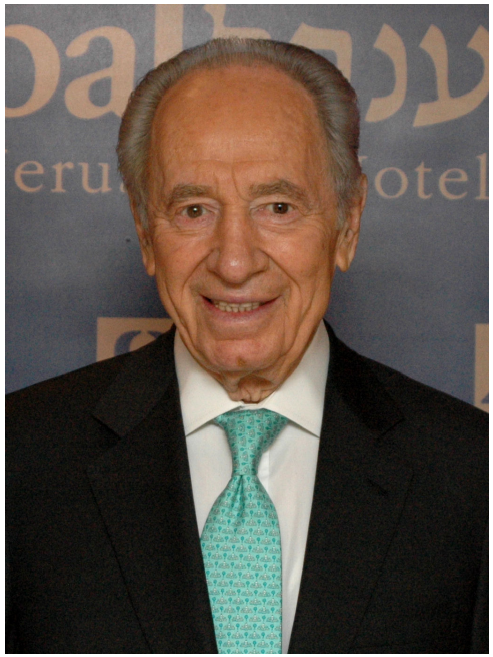
- How well can Plan-and-document hit the cost, schedule, & quality target?
- P&D requires extensive documentation and planning and depends on an experienced manager
  - Can we build software effectively without careful planning and documentation?
  - How to avoid “just hacking”?

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."

"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.**

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

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

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

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?

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

Software as a Service (SaaS)

David Patterson

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

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

Cloud Computing

David Patterson

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

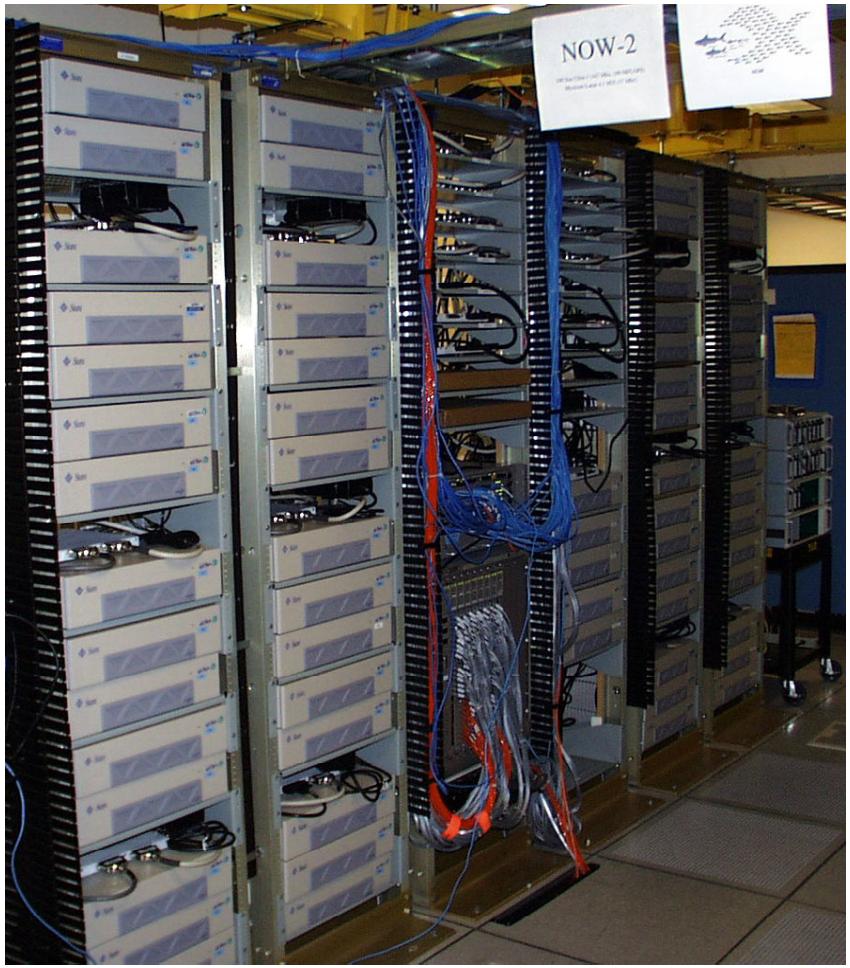
   Service & communication available 24x7

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

First Search Engine to use Cluster of PCs?



Inktomi on Network of Workstations (NOW) @ UCB in 1996

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
  - Amazon runs its own e-commerce on AWS!

2016 AWS Instances & Prices

General Purpose On Demand Instances

Instance Type	Per Hour	$ Ratio to Nano	vCPUs	Compute Units	Memory (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

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)

And in Conclusion

- 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
- Scale led to savings/CPU => reduced cost of Cloud Computing => Utility Computing