Toward software engineering in practice

Claire Le Goues

15-214

April 27, 2017

institute for SOFTWARE RESEARCH

Learning Goals

- Broad scope of software engineering
- Importance of nontechnical issues
- Introduction to key challenges

Software is Everywhere
Software is Important
(duh)

2016 [edit]

This list is up to date as of June 30, 2016. Indicated changes in market value are relative to the previous quarter.

Rank	First quarter[8]	Second quarter	Third quarter	Fourth quarter
1	Apple Inc ▲596,988.7	Apple Inc ▼515,590.0		
2	Alphabet ▼514,923.5	Alphabet ▼475,160.0		
3	Microsoft ▼434,130.1	Microsoft ▼399,710.0		
4	Amazon Inc. ▲356,119.4	Exxon Mobil ▲388,710.0		
5	Berkshire Hathaway ▲349,813.4	Berkshire Hathaway ▲356,810.0		
6	Exxon Mobil ▲346,616.5	Amazon Inc. ▼337,650.0		
7	Facebook ▲326,357.8	Johnson & Johnson ▲333,650.0		
8	Johnson & Johnson ▲309,604.4	Facebook ▲326,880.0		
9	General Electric ▼305,545.7	General Electric ▼306,400.0		
10	Wells Fargo ▼246,035.0	Wells Fargo ▼238,950.0		

Gov't example: Software is integral to DoD systems.

Table 1

System Functionality Requiring Software (source: *PM Magazine*)

Weapon system	Year	Functions performed in software (%)
F-4	1960	8
A-7	1964	10
F-111	1970	20
F-15	1975	35
F-16	1982	45
B-2	1990	65
F-22	2000	80

Quoting an Air Force lieutenant general, "The only thing you can do with an F- 22 that does not require software is take a picture of it."

Crouching Dragon, Hidden Software: Software in Dod Weapon Systems (Ferguson, IEEE Software, 2001)

Failed Software Projects

- SAGE (Semi-Automatic Ground Environment); started 1951, almost obsolete when finished in 1963; higher costs than Manhattan project
- FBI Virtual Case File stopped in 2005 after 3 years and 170 M$
- London stock exchange stopped Taurus project 1993 after 11 years when 13200% over budget

Toyota Case: Single Bit Flip That Killed

Junko Yoshida

10/25/2013 03:35 PM EDT

During the trial, embedded systems experts who reviewed Toyota's electronic throttle source code testified that they found Toyota's source code defective, and that it contains bugs -- including bugs that can cause unintended acceleration.

"We did a few things that NASA apparently did not have time to do," Barr said. For one thing, by looking within the real-time operating system, the experts identified "unprotected critical variables." They obtained and reviewed the source code for the "sub-CPU," and they "uncovered gaps and defects in the throttle fail safes."

The experts demonstrated that "the defects we found were linked to unintended acceleration through vehicle testing," Barr said. "We also obtained and reviewed the source code for the black box and found that it can record false information about the driver's actions in the final seconds before a crash."

Stack overflow and software bugs led to memory corruption, he said. And it turns out that the crux of the issue was these memory corruptions, which acted "like ricocheting bullets."

Barr also said more than half the dozens of tasks' deaths studied by the experts in their experiments "were not detected by any fail safe."

Bookout Trial Reporting

http://www.eetimes.com/document.asp?doc_id=1319903&page_number=1 (excerpts)

"Task X death in combination with other task deaths"

institute for SOFTWARE RESEARCH

Healthcare.gov: Government IT Project Failure at its Finest

Posted: 10/18/2013 6:33 pm

Read more > Project Management, Government, Healthcare, It Projects, Open Source, Business News

The *BusinessWeek* article on the Healthcare.gov failure is nothing if not instructive. From the piece:

> Healthcare.gov isn't just a website; it's more like a platform for building health-care marketplaces. Visiting the site is like visiting a restaurant. You sit in the dining room, read the menu, and tell the waiter what you want, and off he goes to the kitchen with your order. The dining room is the front end, with all the buttons to click and forms to fill out. The kitchen is the back end, with all the databases and services. The contractor most responsible for the back end is CGI Federal. Apparently it's this company's part of the system that's burning up under the load of thousands of simultaneous users.

The restaurant analogy is a good one. Projects with scopes like these fail for all sorts of reasons. *Why New Systems Fail* details a bunch of culprits, most of which are people-related.

As I read the article, a few other things jumped out at me, as they virtually guarantee failure:

- The sheer number of vendors involved
- The unwillingness of key parties involved with the back-end to embrace transparency

“But we’re CMU students and we are really, really smart!”

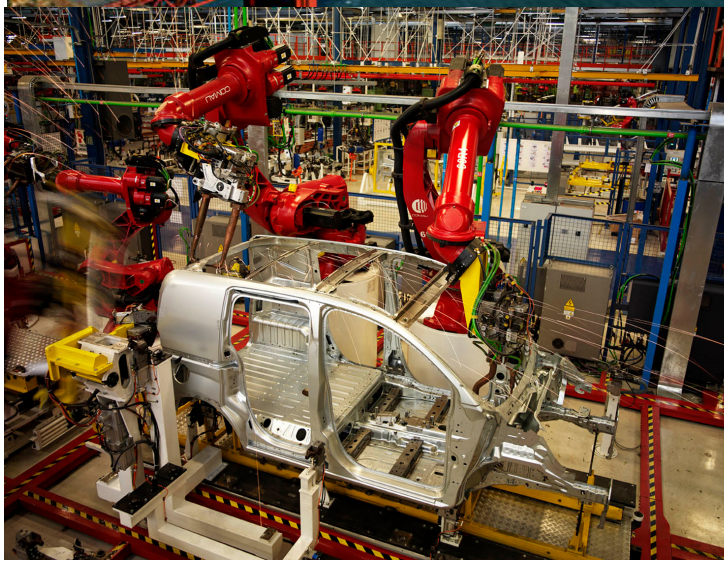
What is engineering? And how is it different from hacking/ programming?

Software *Engineering*?

1968 NATO Conference on Software Engineering

- Provocative Title
- Call for Action
- “Software crisis”

Envy of Engineers



- Producing a car/bridge
  - Estimable costs and risks
  - Expected results
  - High quality
- Separation between plan and production
- Simulation before construction
- Quality assurance through measurement
- Potential for automation

Software Engineering?

„The Establishment and use of sound ***engineering principles*** *in order to obtain* ***economically*** *software that is* ***reliable*** *and works* ***efficiently*** *on* ***real*** *machines."*

[Bauer 1975, S. 524]

Healthcare.gov: Government IT Project Failure at its Finest

Posted: 10/18/2013 6:33 pm

Read more > Project Management, Government, Healthcare, It Projects, Open Source, Business News

The *BusinessWeek* article on the Healthcare.gov failure is nothing if not instructive. From the piece:

> Healthcare.gov isn't just a website; it's more like a platform for building health-care marketplaces. Visiting the site is like visiting a restaurant. You sit in the dining room, read the menu, and tell the waiter what you want, and off he goes to the kitchen with your order. The dining room is the front end, with all the buttons to click and forms to fill out. The kitchen is the back end, with all the databases and services. The contractor most responsible for the back end is CGI Federal. Apparently it's this company's part of the system that's burning up under the load of thousands of simultaneous users.

The restaurant analogy is a good one. Projects with scopes like these fail for all sorts of reasons. *Why New Systems Fail* details a bunch of culprits, most of which are people-related.

As I read the article, a few other things jumped out at me, as they virtually guarantee failure:

- The sheer number of vendors involved
- The unwillingness of key parties involved with the back-end to embrace transparency

What happened with HealthCare.gov?

- Poor team and process coordination.
- Changing requirements.
- Inadequate quality assurance infrastructure.
- Architecture unsuited to the ultimate system load.

Process

How to develop software?

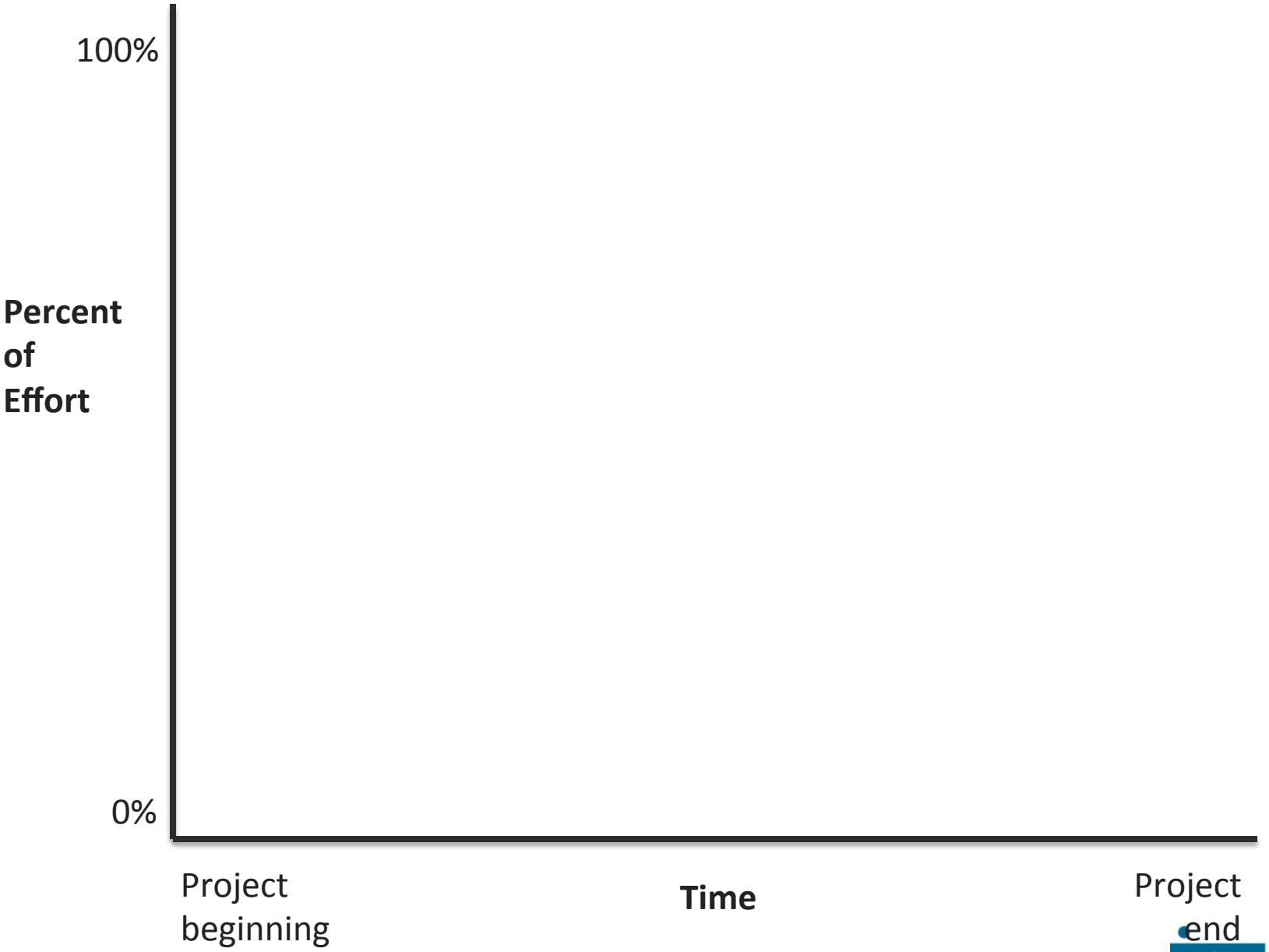
1. Discuss the software that needs to be written
2. Write some code
3. Test the code to identify the defects
4. Debug to find causes of defects
5. Fix the defects
6. If not done, return to step 1

Software Process

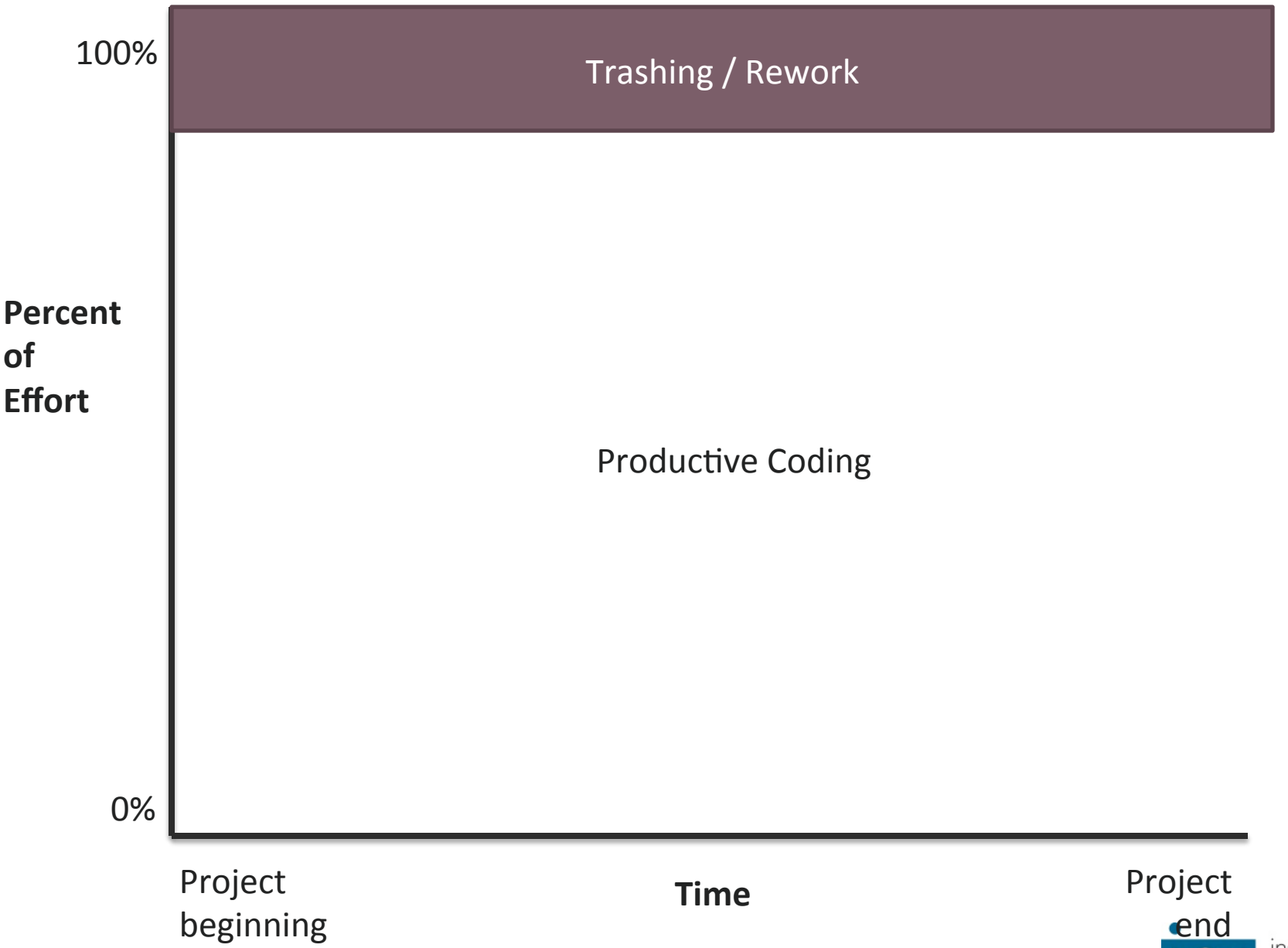
"The set of activities and associated results that produce a software product"

What makes a good process?

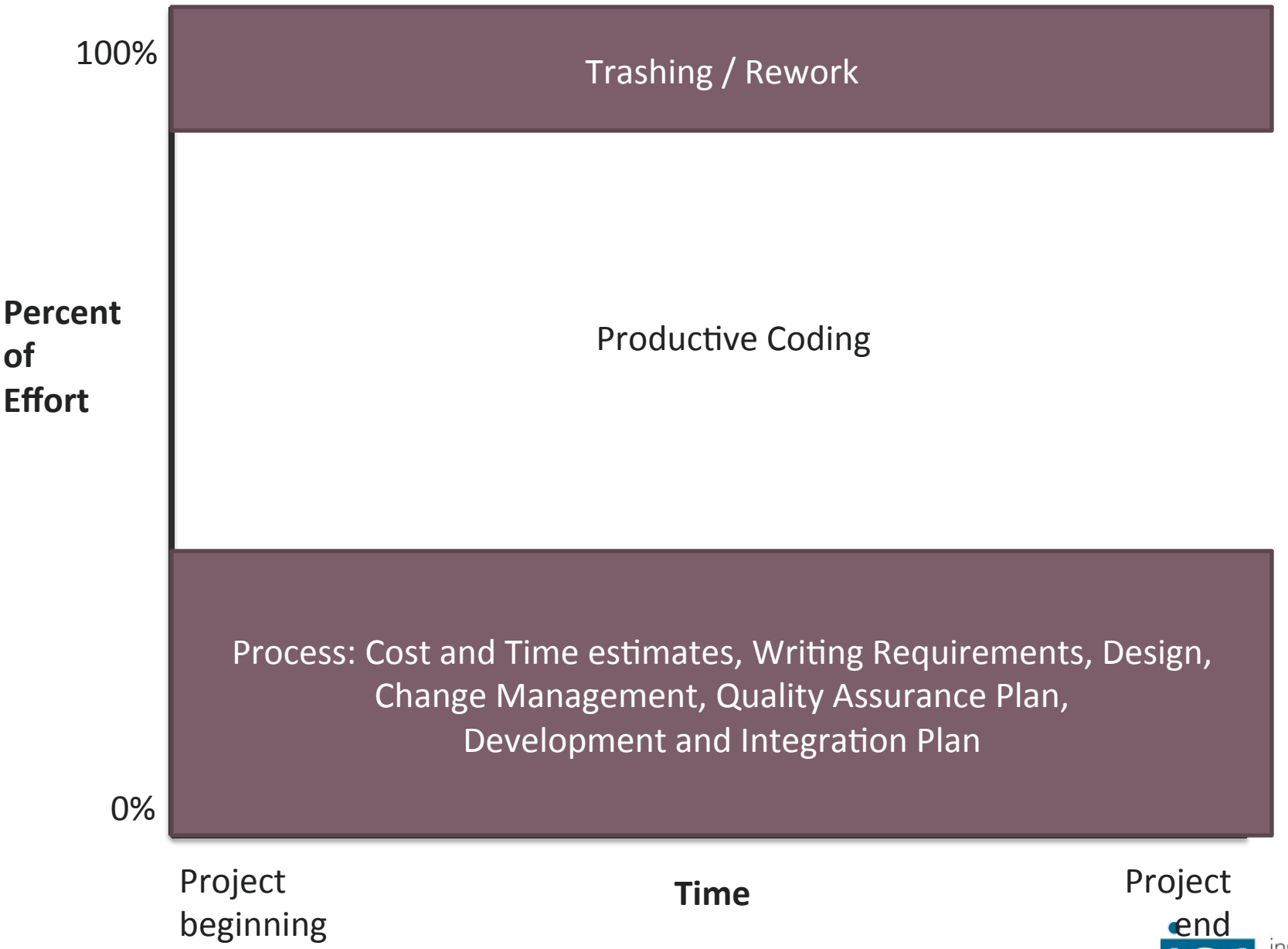
Sommerville, SE, ed. 8



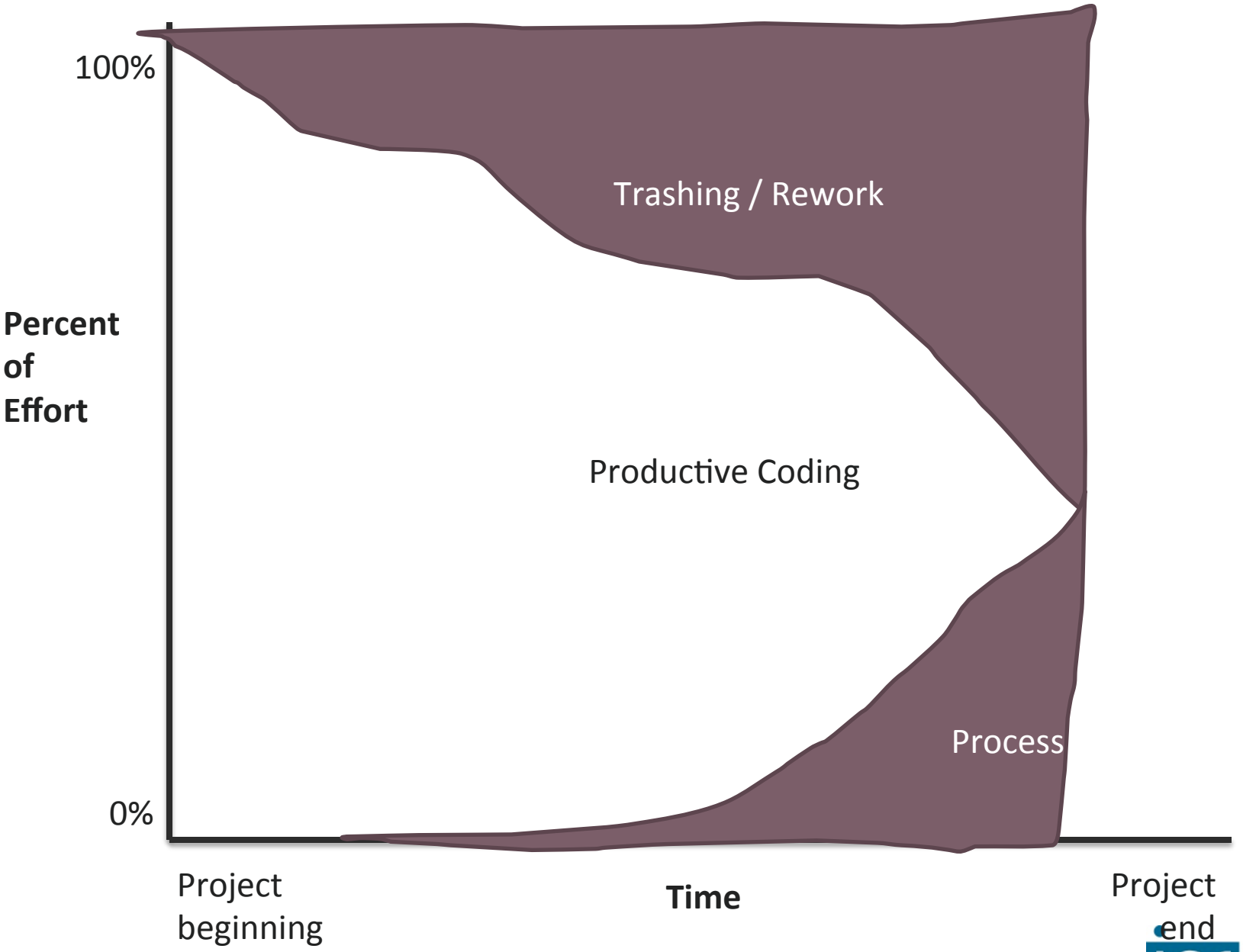
institute for SOFTWARE RESEARCH



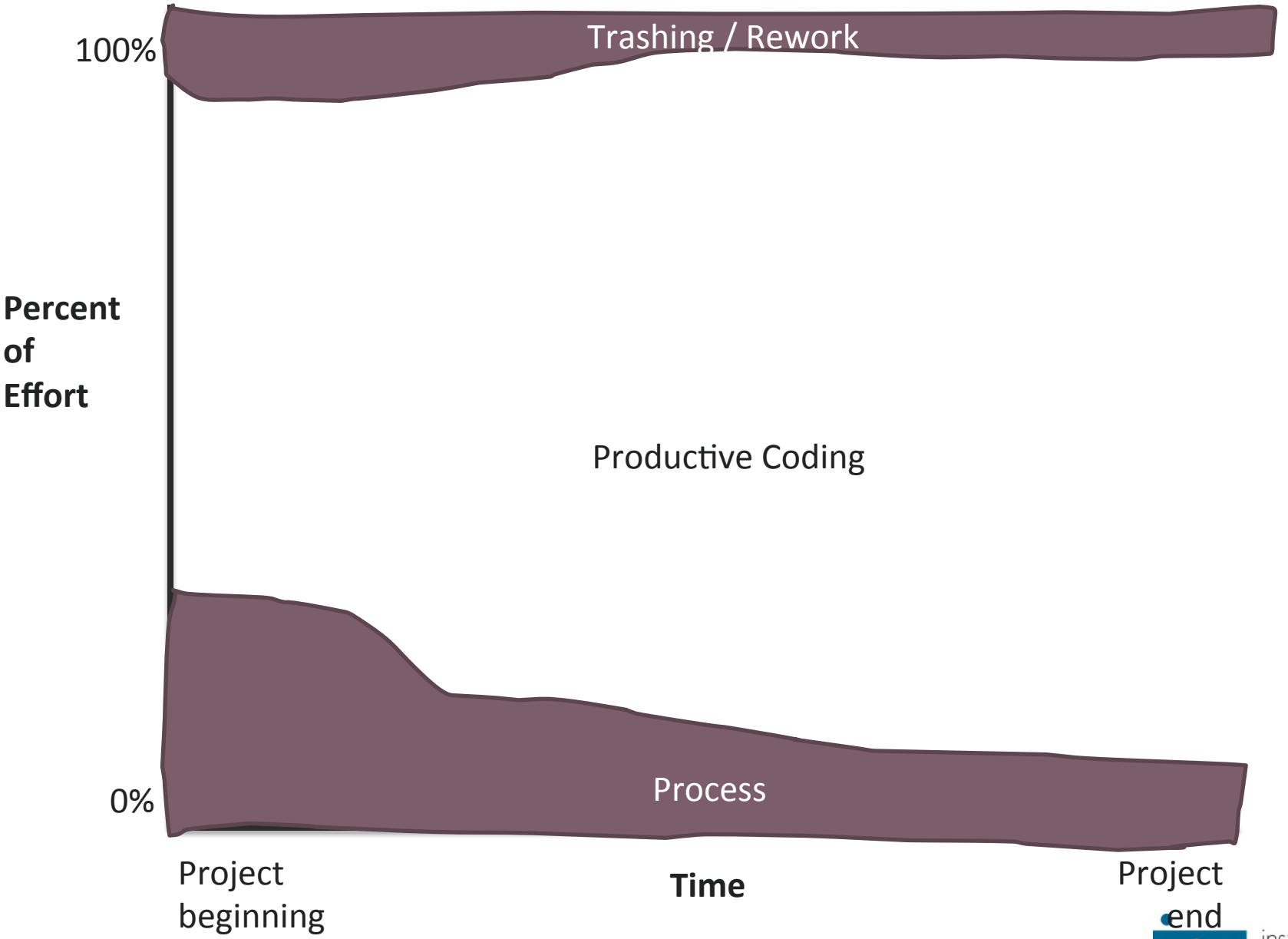
institute for SOFTWARE RESEARCH



institute for SOFTWARE RESEARCH



institute for SOFTWARE RESEARCH

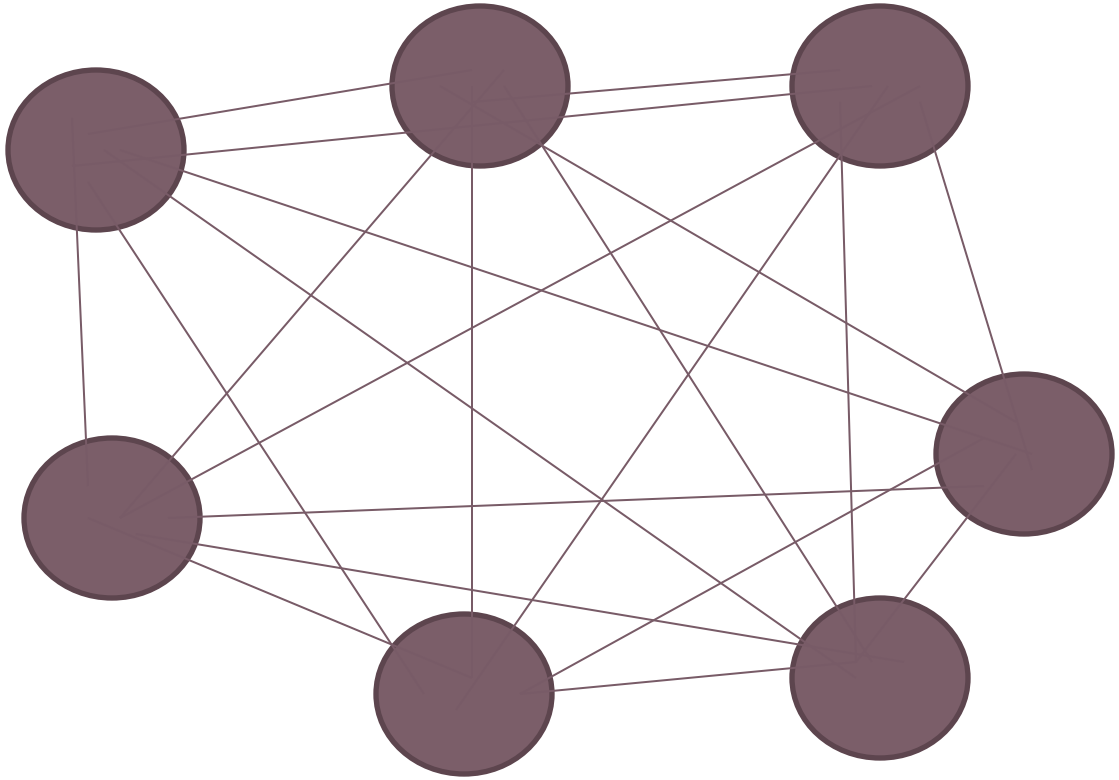


institute for SOFTWARE RESEARCH

Example process issues

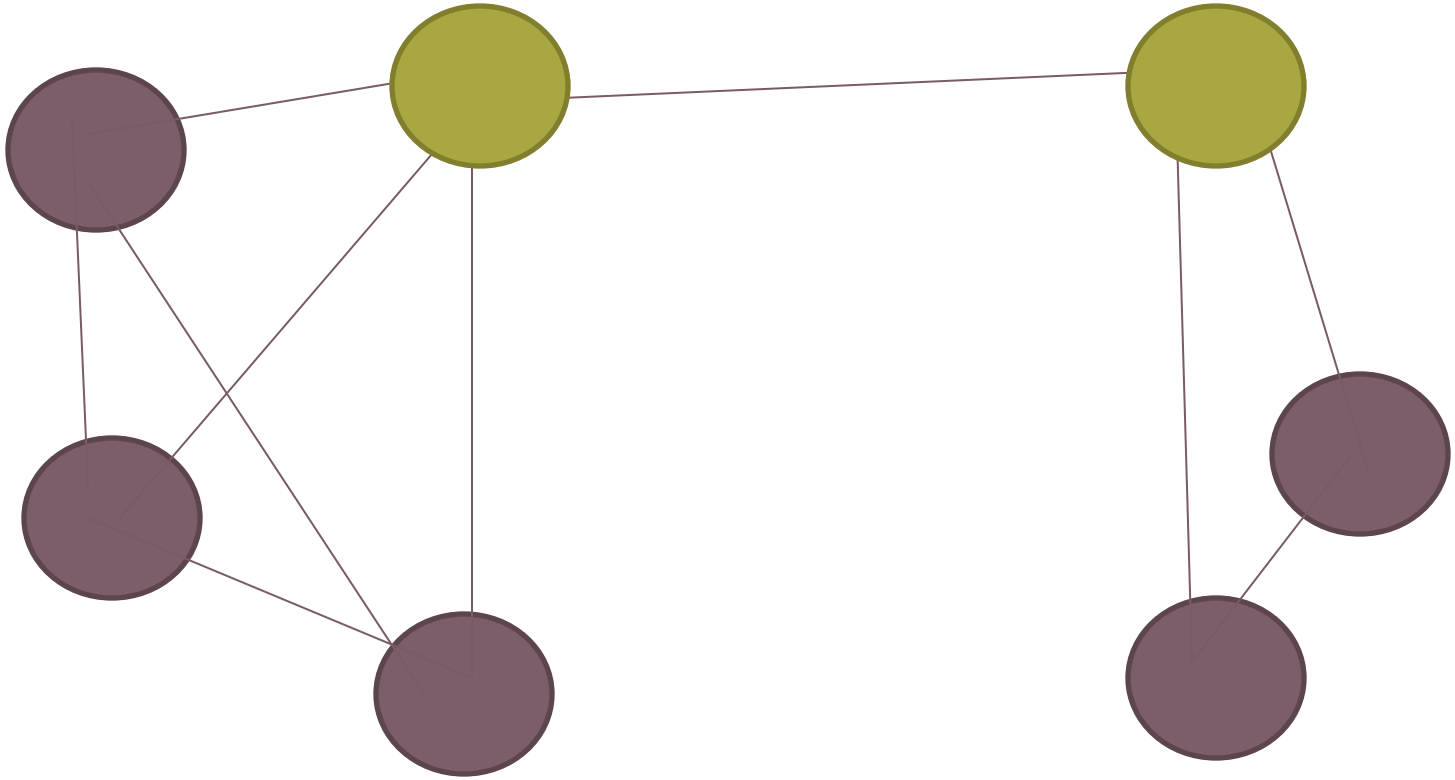
- Change Control: Mid-project informal agreement to changes suggested by customer or manager. Project scope expands 25-50%
- Quality Assurance: Late detection of requirements and design issues. Test-debug-reimplement cycle limits development of new features. Release with known defects.
- Defect Tracking: Bug reports collected informally, forgotten
- System Integration: Integration of independently developed components at the very end of the project. Interfaces out of sync.
- Source Code Control: Accidentally overwritten changes, lost work.
- Scheduling: When project is behind, developers are asked weekly for new estimates.

Process Costs



$n(n - 1) / 2$
communication links

Process Costs



Large teams (29 people) create around six times as many defects as small teams (3 people) and obviously burn through a lot more money. Yet, the large team appears to produce about the same mount of output in only an average of 12 days' less time. This is a truly astonishing finding, through it fits with my personal experience on projects over 35 years.

- Phillip Amour, 2006, CACM 49:9

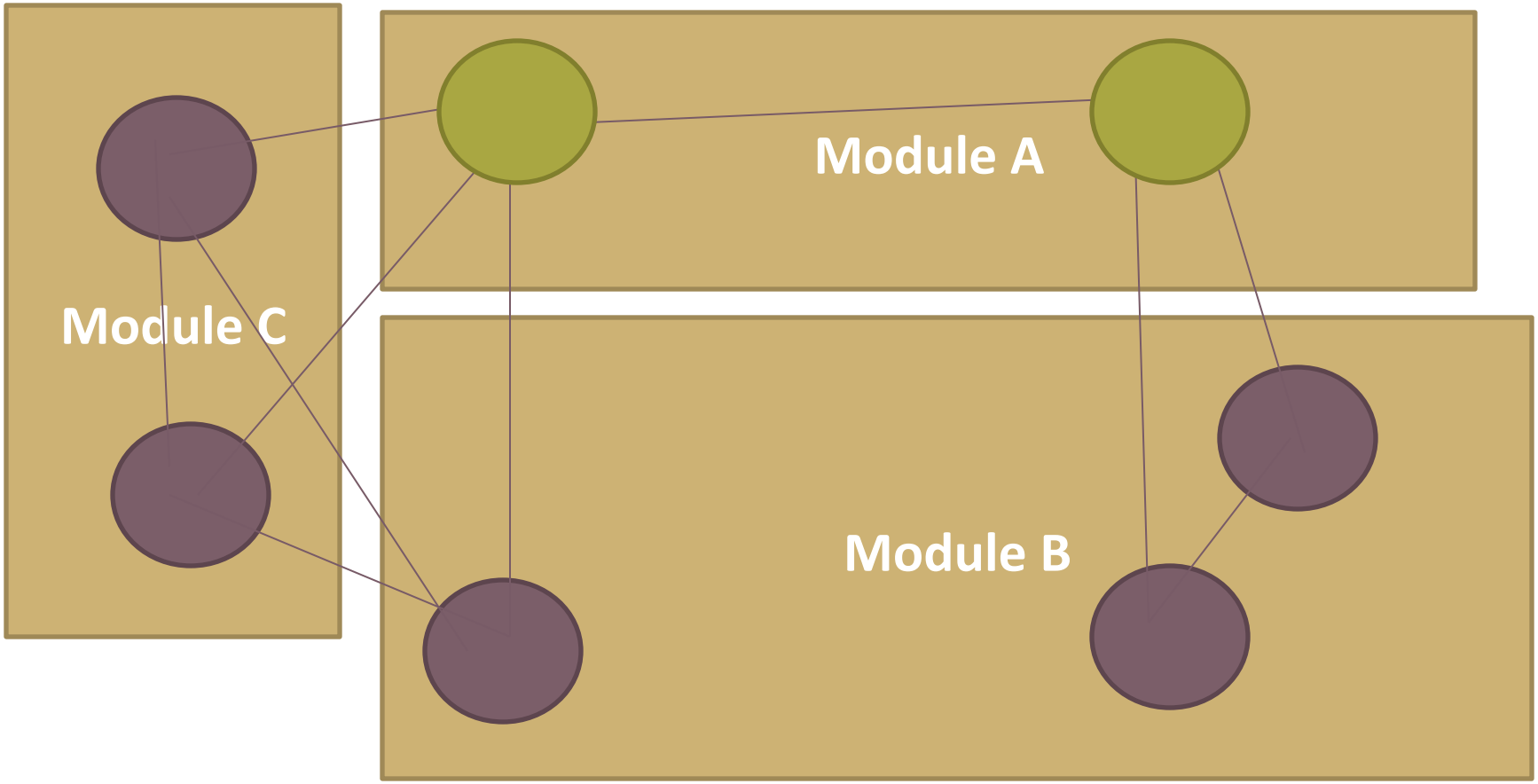
Conway's Law

"Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure."

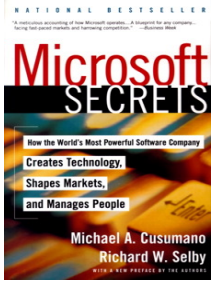
— *Mel Conway, 1967*

"If you have four groups working on a compiler, you'll get a 4-pass compiler."

Congruence



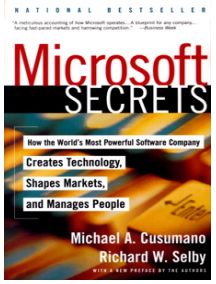
Microsoft's Small Team Practices



- Vision statement and milestones (2-4 month), no formal spec
- Feature selection, prioritized by market, assigned to milestones
- Modular architecture
  - Allows small federated teams (Conway's law)
- Small teams of overlapping functional specialists

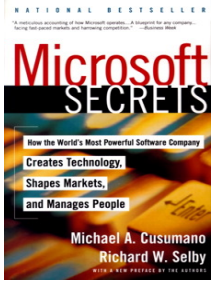
Windows 95: 200 developers and testers, one of 250 products

Microsoft's Small Team Practices



- Feature Team
  - 3-8 developers (design, develop)
  - 3-8 testers (validation, verification, usability, market analysis)
  - 1 program manager (vision, schedule communication; leader, facilitator) – working on several features
  - 1 product manager (marketing research, plan, betas)

Microsoft's Small Team Practices



- "Synchronize and stabilize"
- For each milestone
  - 6-10 weeks feature development and continuous testing
    - frequent merges, daily builds
  - 2-5 weeks integration and testing ("zero-bug release", external betas)
  - 2-5 weeks buffer

Agile Practices (e.g., Scrum)

- 7+/-2 team members, collocated
- self managing
- Scrum master (potentially shared among 2-3 teams)
- Product owner / customer representative

Planning

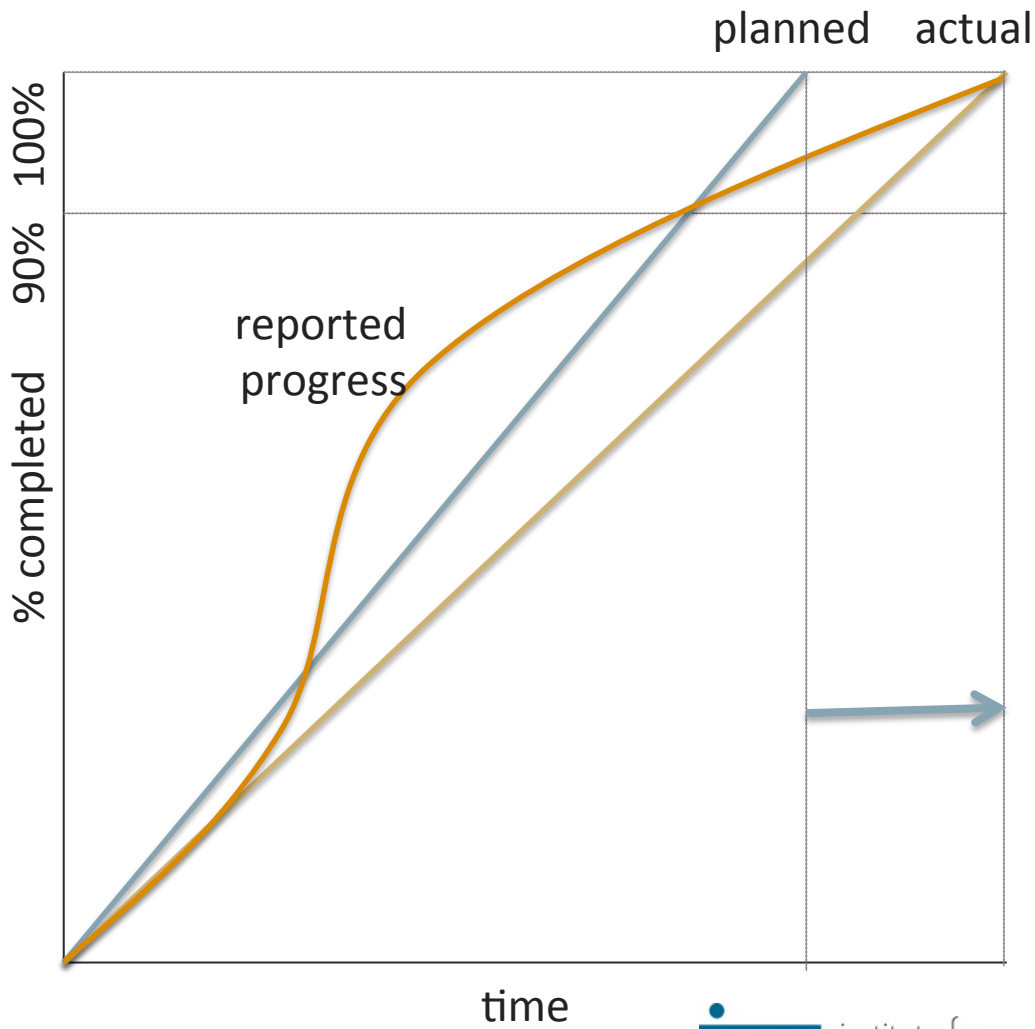
Measuring Progress?

- "I'm almost done with the X. Component A is almost fully implemented. Component B is finished except for the one stupid bug that sometimes crashes the server. I only need to find the one stupid bug, but that can probably be done in an afternoon?"

Almost Done Problem

- Last 10% of work -> 40% of time (or 20/80)

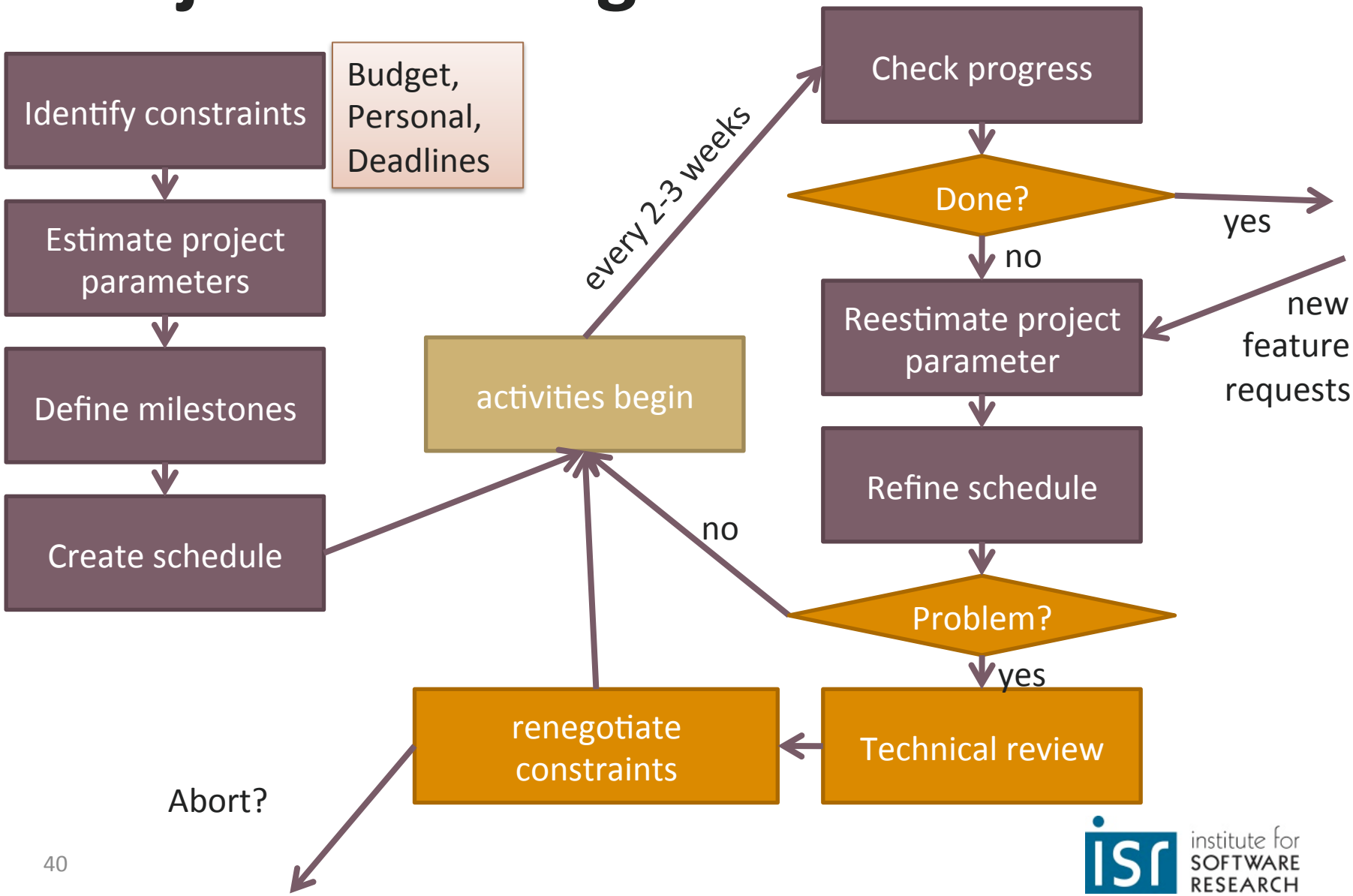
- Make progress measureable
- Avoid depending entirely on developer estimations



Measuring Progress?

- Developer judgment: x% done
- Lines of code?
- Functionality?
- Quality?

Project Planning



institute for SOFTWARE RESEARCH

Reasons for Missed Deadlines

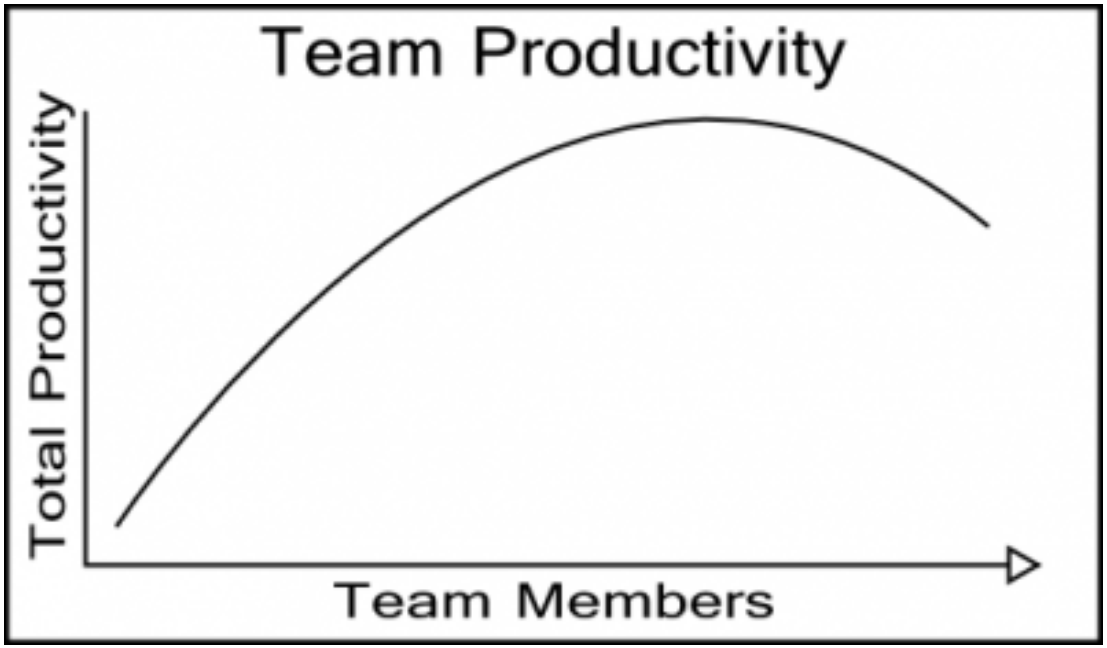
- Insufficient staff (illnesses, staff turnover, ...)
- Insufficient qualification
- Unanticipated difficulties
- Unrealistic time estimations
- Unanticipated dependencies
- Changing requirements, additional requirements
- Especially in student projects
  - Underestimated time for learning technologies
  - Uneven work distribution
  - Last-minute panic.

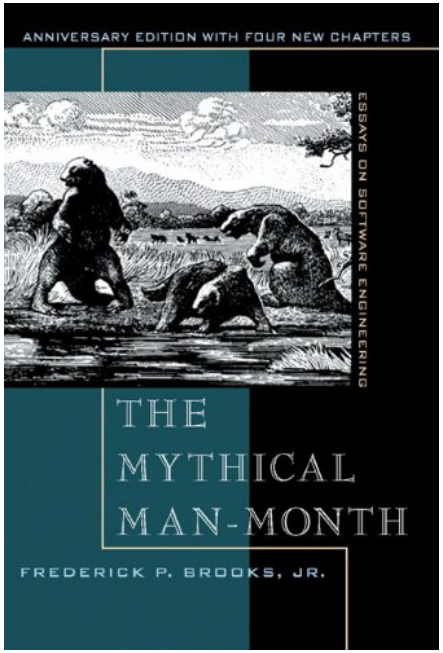
Recognize Scheduling Issues Early

- Monitoring and formal reporting necessary
  - Establish who, when, what
  - Compare planned/actual data
- Measurable milestones
- Outdated schedules no meaningful management mechanism

Team productivity

- Brook's law: Adding people to a late software project makes it later.





Estimating Effort

π

Task: Estimate Time

- A: Java version of the Monopoly boardgame with Pittsburgh street names
  - (you)
- B: Bank smartphone app
  - (you with team of 4 developers, one experienced with iPhone apps, one with background in security)

- Estimate in 8h days (20 work days in a month, 220 per year)

Company	Firm price without VAT (Euro)	Time schedule (days)	A&D in bids	Planned effort on A&D (%)	Emphasis on A&D
1	2630	14	Brief (2)		
2	4380		Brief (2)		
3	4880		Very brief (1)		
4	4970	28	Brief (2)	30	5.0
5	8750	18	Detailed (3)	7	3.7
6	9940		None (0)	40	4.0
7	11810		Brief (2)	0	2.0
8	11880	94	Detailed (3)	26	5.6
9	12190	77	Very detailed (4)	5	4.5
10	16630		Brief (2)	12	3.2
11	18130		Very brief (1)		
12	18510	91	Brief (2)	20	4.0
13	20000	30	Detailed (3)	28	5.8
14	20020		Very brief (1)	50	6.0
15	21090		Very brief (1)	44	5.4
16	25310		Very detailed (4)	11	5.1
17	33250	49	Detailed (3)	26	5.6
18	25810		Very brief (1)		
19	25940		Brief (2)	20	4.0
20	25980		Very detailed (4)	8	4.8
21	26880	45	Detailed (3)		
22	28700	77	Very detailed (4)	10	5.0
23	28950	42	Brief (2)	30	5.0
24	29000		Brief (2)		
25	33530		Brief (2)		
26	33880	77	Detailed (3)	10	4.0
27	33900		Detailed (3)	11	4.1
28	34500		Very brief (1)	36	4.6
29	38360	63	Detailed (3)	20	5.0
30	45380		Detailed (3)	10	4.0
31	52310		Brief (2)	27	4.7
32	56900		Detailed (3)	14	4.4
33	60750		Brief (2)	43	6.3
34	69060	49	Detailed (3)	23	5.3
35	69940		Detailed (3)	6	3.6

Anda, Bente CD, Dag IK Sjøberg, and Audris Mockus. "Variability and reproducibility in software engineering: A study of four companies that developed the same system." *IEEE Transactions on Software Engineering* 35.3 (2009): 407-429.

Development Process

	Company A	Company B	Company C	Company D
Nationality	Norwegian	Norwegian	Norwegian	International
Ownership	Private	By employees	By employees	Listed on exchanges
Location	Oslo	Oslo	Bergen	Oslo + 20 countries
Size	Appr. 100	Appr. 25	Appr. 8	Appr. 13,000 worldwide
Firm price	€20,000	€45,380	€8,750	€56,000
Agreed time schedule	55 days	73 days	41 days	62 days
Planned effort on A&D	28%	20%	7%	23%

Dimensions		Company A	Company B	Company C	Company D
Project	**Contractor-related costs**	90 hours	108 hours	155 hours	85 hours
	Actual lead time	87 days	90 days	79 days	65 days
	Schedule overrun	58%	23%	93%	5%
Product	**Reliability**	Good	Good	Poor	Fair
	Usability	Good	Fair	Fair	Good
	Maintainability	Good	Poor	Poor	Good

Risk and Uncertainty

Innovative vs Routine Projects

- Most software projects are innovative
  - Google, Amazon, Ebay, Netflix
  - Vehicles and robotics
  - Language processing, Graphics
- Routine (now, not 10 years ago)
  - E-commerce websites?
  - Many control systems?
  - Routine gets automated -> innovation cycle

Sources of Uncertainty

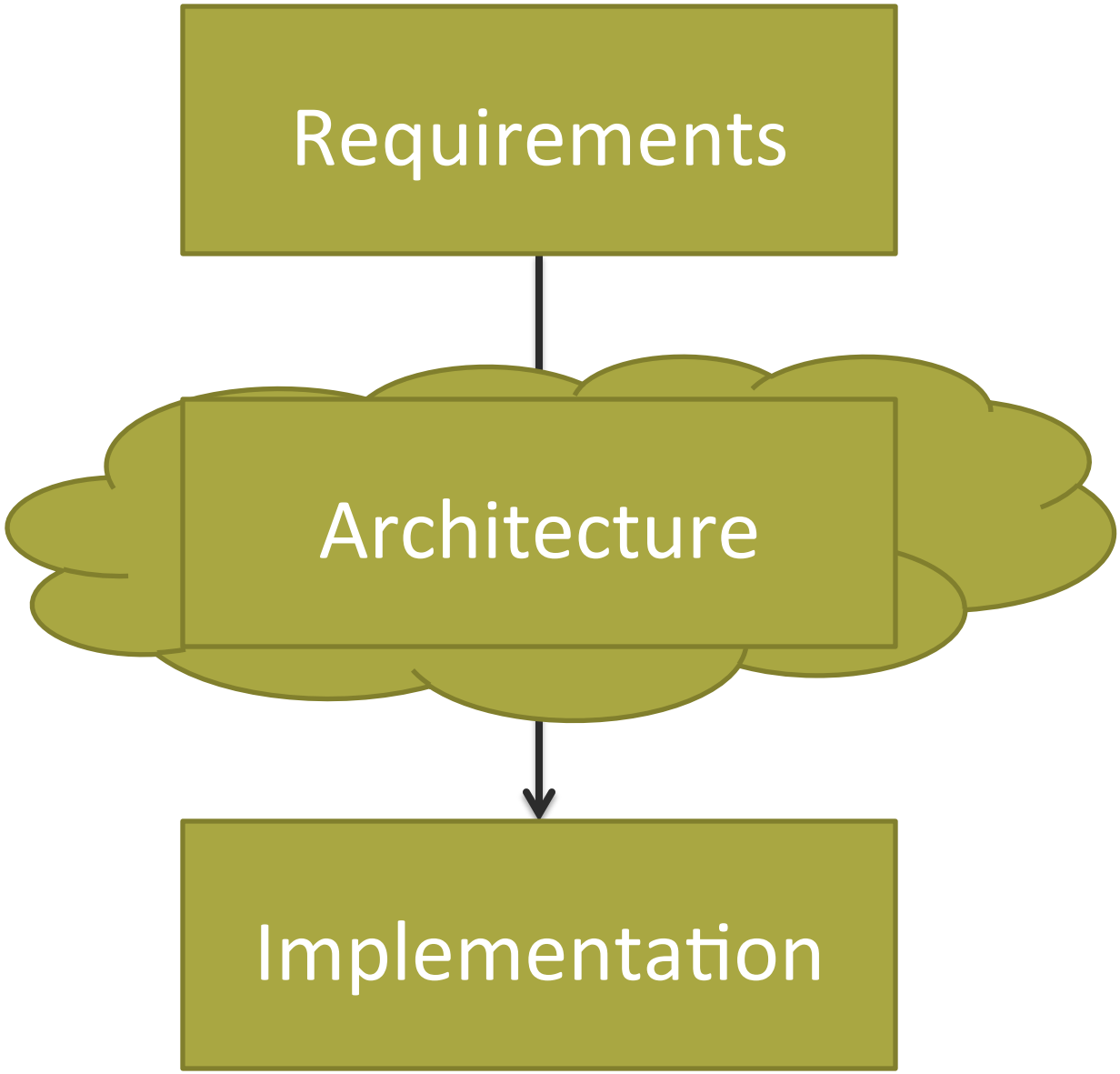
- Unpredictable operating environment
  - Cybersecurity threats, device drivers
  - Unanticipated usage scenarios
- Limited predictive power of models
  - Halting, abstract interpretation, testing
- Bounded rationality of humans
  - Designers, developers
  - Customers, users

Risk management

- Key task of a project manager
- Identify and evaluate risks early
- If necessary, plan mitigation strategies
- Document results of risk analysis in project plan

- Project risks: scheduling and resources
  - e.g., staff illness/turnover
- Product risks: Quality and functionality of the product
  - e.g. used component too slow
- Business risks:
  - e.g., competitor introduces similar product

Software Architecture



Software Architecture

"The software architecture of a computing system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both."

[Clements et al. 2010]

Beyond functional correctness

- Quality matters, eg.,
  - Availability
  - Modifiability, portability
  - Performance, scalability
  - Security
  - Testability
  - Usability
  - Cost to build, cost to operate

Design vs. Architecture

Design Questions

- How do I add a menu item in Eclipse?
- How can I make it easy to add menu items in Eclipse?
- What lock protects this data?
- How does Google rank pages?
- What encoder should I use for secure communication?
- What is the interface between objects?

Architectural Questions

- How do I extend Eclipse with a plugin?
- What threads exist and how do they coordinate?
- How does Google scale to billions of hits per day?
- Where should I put my firewalls?
- What is the interface between subsystems?

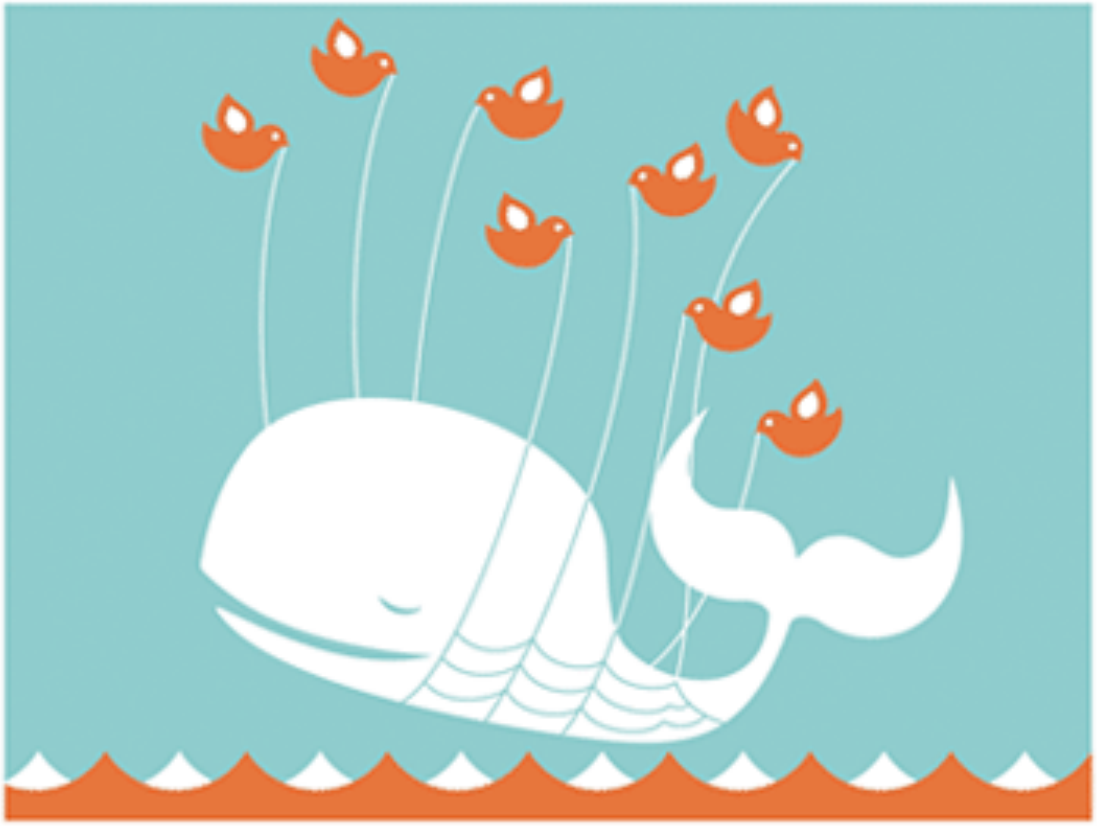
Case Study:
Architecture Changes at Twitter

twitter

Home Public Timeline Help

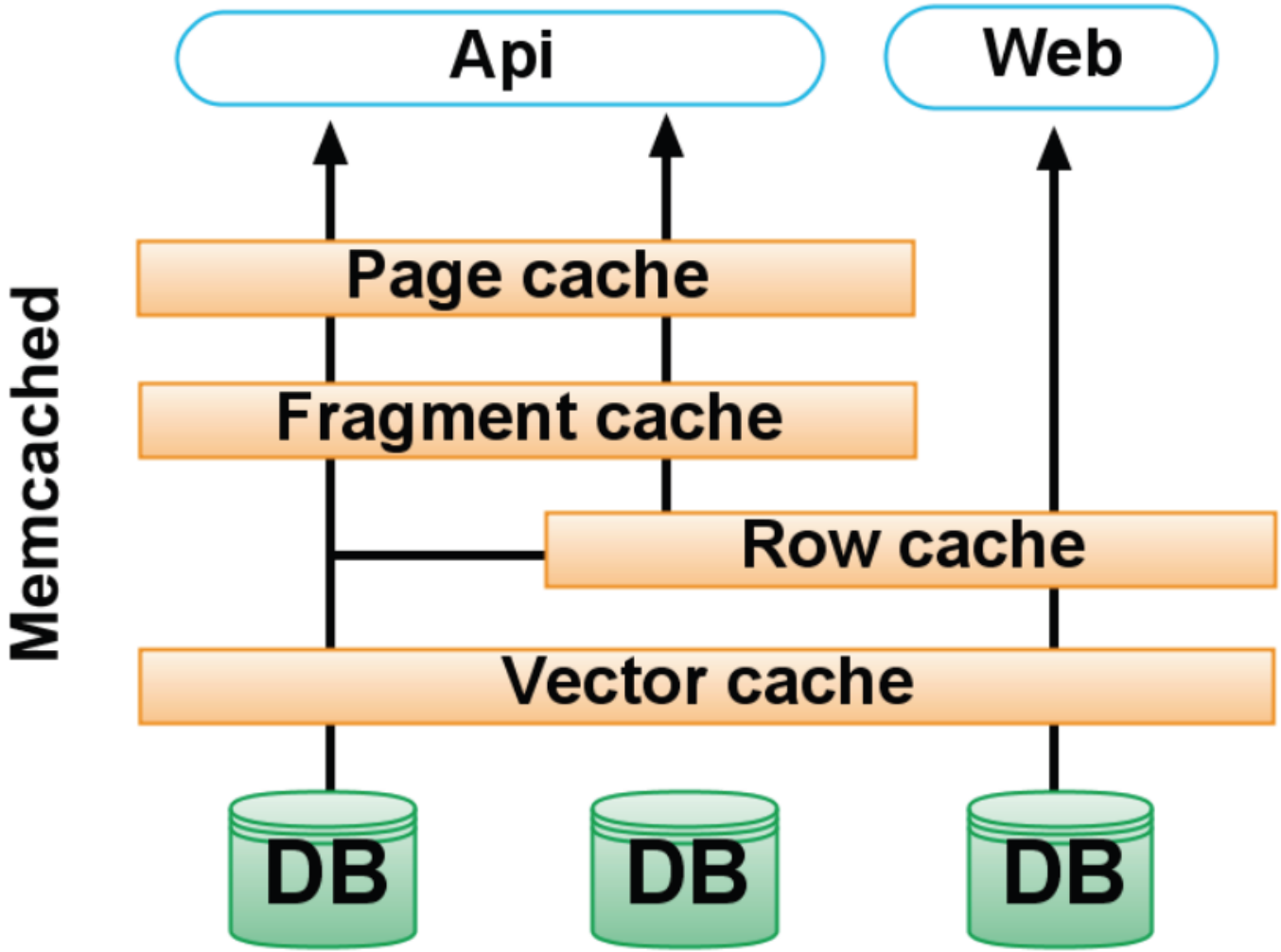
Twitter is over capacity.

Too many tweets! Please wait a moment and try again.



© 2009 Twitter About Us Contact Blog Status API Help Jobs TOS Privacy

Caching



Decision to Rearchitect Twitter

"After that experience, we determined we **needed to step back**. We then determined we needed to **re-architect** the site to support the continued growth of Twitter and to keep it running smoothly."

Redesign Goals

- Improve median latency; lower outliers
- Reduce number of machines 10x
- Isolate failures
- "We wanted cleaner boundaries with “related” logic being in one place"
  - encapsulation and modularity at the systems level (rather than at the class, module, or package level)
- Quicker release of new features
  - "run small and empowered engineering teams that could make local decisions and ship user-facing changes, independent of other teams"

performance

reliability

maintainability

modifiability

JVM vs Ruby VM

- Rails servers capabile of 200-300 requests / sec / host
- Experience with Scala on the JVM; level of trust
- Rewrite for JVM allowed 10-20k requests / sec / host

Programming Model

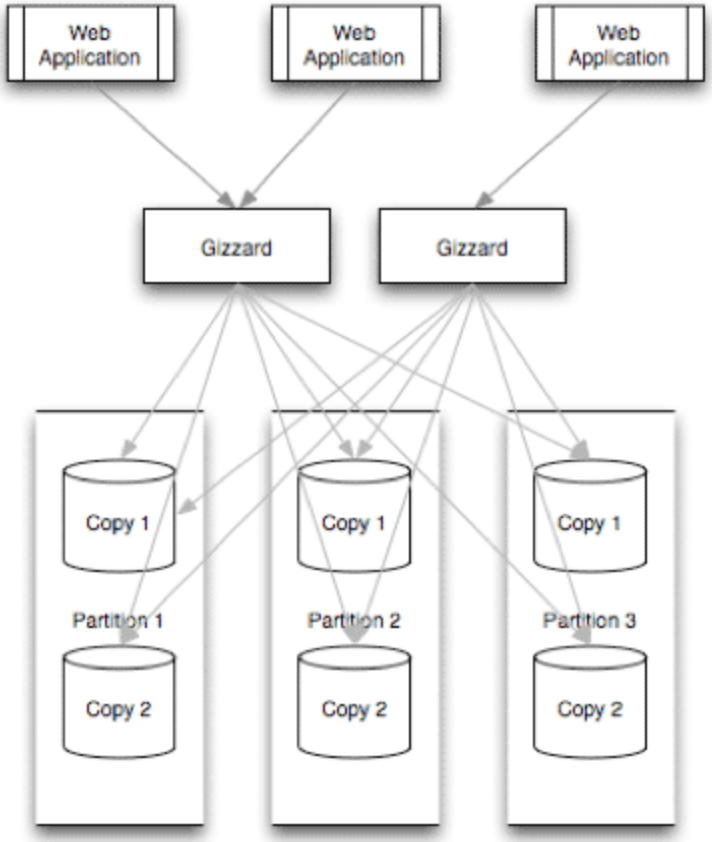
- Ruby model: Concurrency at process level; request queued to be handled by one process
- Twitter response aggregated from several services – additive response times
- *"As we started to decompose the system into services, each team took slightly different approaches. For example, the failure semantics from clients to services didn't interact well: we had no consistent back-pressure mechanism for servers to signal back to clients and we experienced "thundering herds" from clients aggressively retrying latent services."*
- Goal: Single and uniform way of thinking about concurrency
  - Implemented in a library for RPC (Finagle), connection pooling, failover strategies and load balancing

Independent Systems

- *" In our monolithic world, we either needed experts who understood the entire codebase or clear owners at the module or class level. Sadly, the codebase was getting too large to have global experts and, in practice, having clear owners at the module or class level wasn't working. Our codebase was becoming harder to maintain, and teams constantly spent time going on "archeology digs" to understand certain functionality. Or we'd organize "whale hunting expeditions" to try to understand large scale failures that occurred."*
- From monolithic system to multiple services
  - Agree on RPC interfaces, develop system internals independently
  - Self-contained teams

Storage

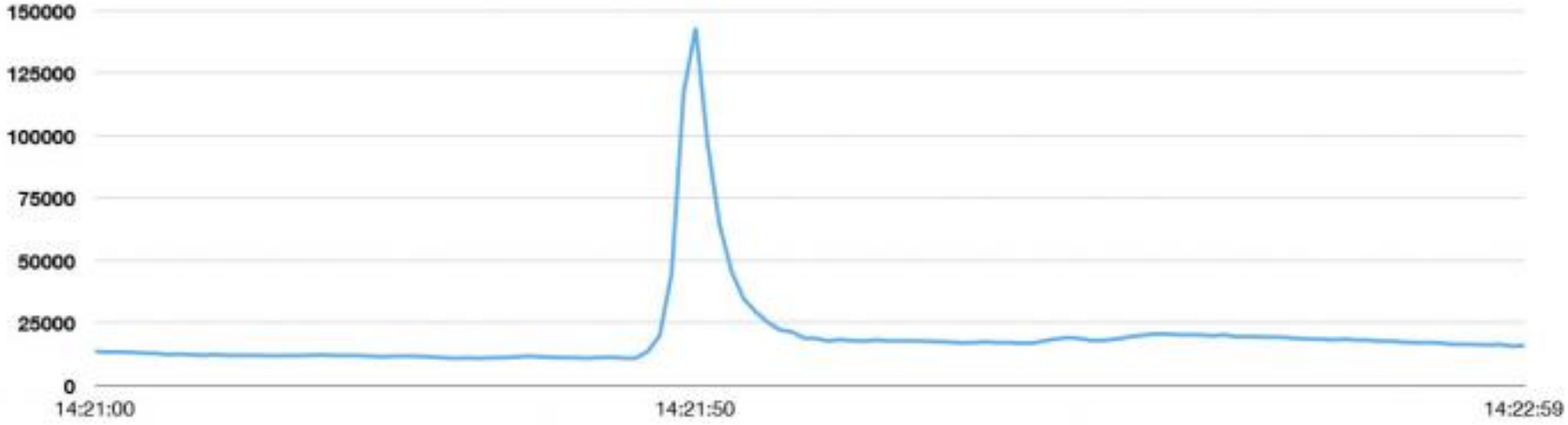
- Single-master MySQL database bottleneck despite more modular code
- Temporal clustering
  - Short-term solution
  - Skewed load balance
  - One machine + replications every 3 weeks
- Move to distributed database (Glizzard on MySQL) with "roughly sortable" ids
- Stability over features – using older MySQL version



Data-Driven Decisions

- Many small independent services, number growing
- Own dynamic analysis tool on top of RPC framework
- Framework to configure large numbers of machines
  - Including facility to expose feature to parts of users only


血湧き、
天空の城
ラピュタ



On Saturday, August 3 in Japan, people watched an airing of Castle in the Sky, and at one moment they took to Twitter so much that we hit a one-second peak of 143,199 Tweets per second.

Outcome: Rearchitecting Twitter

"This re-architecture has not only made the service more **resilient when traffic spikes** to record highs, but also provides a more **flexible** platform on which to **build more features faster**, including synchronizing direct messages across devices, Twitter cards that allow Tweets to become richer and contain more content, and a rich search experience that includes stories and users."

Key Insights: Twitter Case Study

- Architectural decisions affect entire systems, not only individual modules
- Abstract, different abstractions for different scenarios
- Reason about quality attributes early
- Make architectural decisions explicit

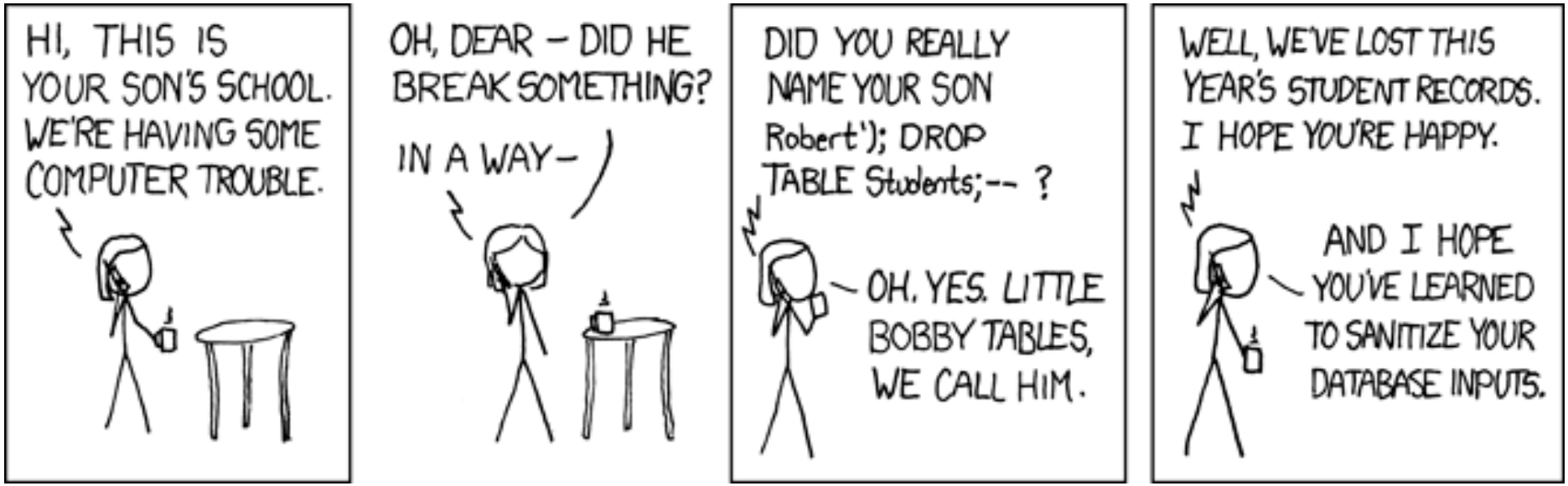
Was the original architect wrong?

How can I test my system with respect to desired quality attributes?

Example: Scalability

Which QA strategy is suitable?

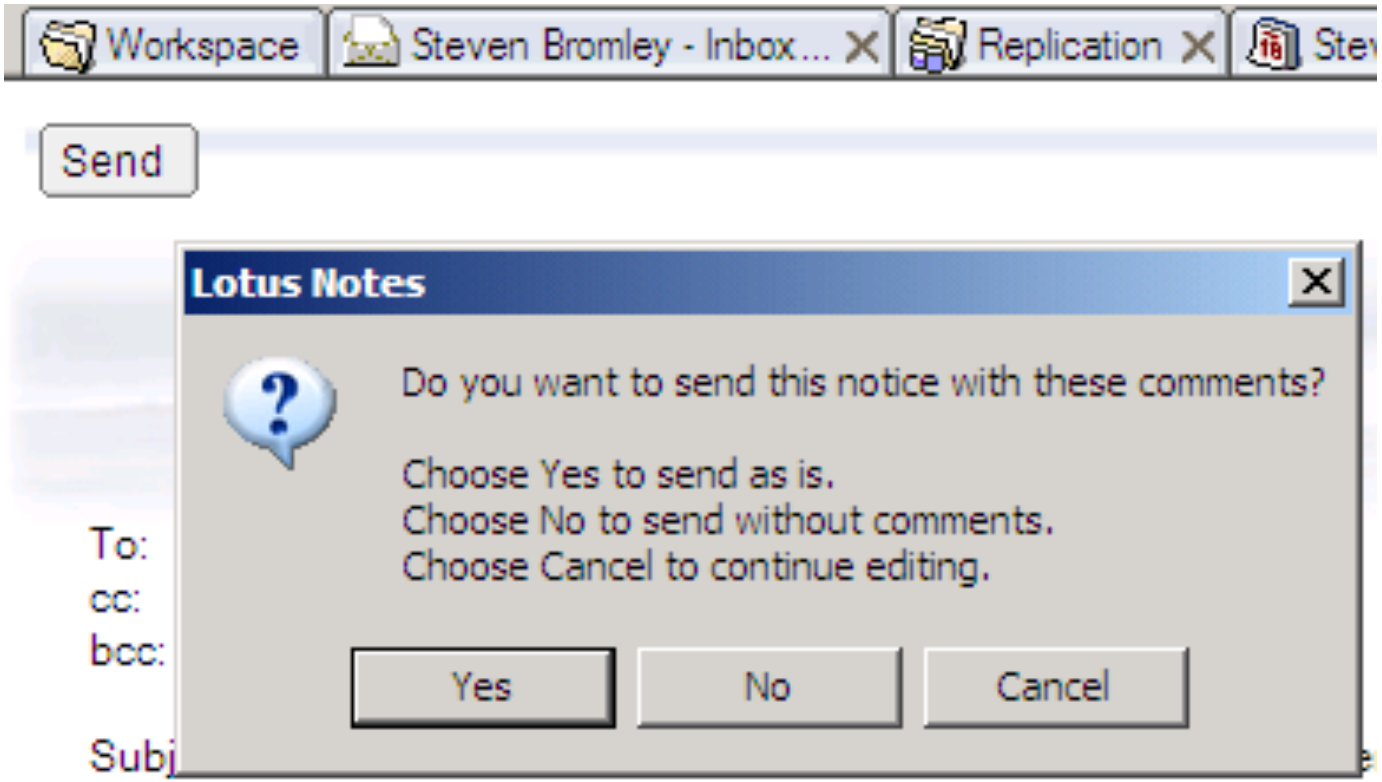
Example: SQL Injection Attacks



Which QA strategy is suitable?

http://xkcd.com/327/

Example: Usability



Which QA strategy is suitable?

institute for SOFTWARE RESEARCH

institute for SOFTWARE RESEARCH

QA Tradeoffs

- Understand limitations of QA approaches
  - e.g. testing vs static analysis, formal verification vs inspection, …
- Mix and match techniques
- Different techniques for different qualities
- …When am I done?

Quick aside on bug fixing and the tricky relationship between design, intent, implementation, and your cranky users…

Race conditions

- Races can occur when:
  - Multiple threads of control access shared data
  - Data gets corrupted when internal integrity assumptions are violated.
- How we protect against races
  - Use "lock" objects that enable access by one thread at a time
    - E.g., event dispatch
    - A language feature in Java, Ada95, etc.
  - Follow a thread discipline in which only one thread can access critical data (Common in GUI APIs e.g., graphical toolkit redraw)
- *Issue*: Basically the hardest bugs to find, fix, and protect against.
  - Why?

`java.io.BufferedInputStream`

- Buffering wrapper for unbuffered stream input: **read**, **close**, reset, skip, mark, etc.
- JDK < 1.2: Race condition between methods **read** and **close**: interleaved execution could cause **read** to throw `NullPointerException`
  - But not always; concurrency → non-deterministic!
- JDK1.2 fixes by synchronize-ing the methods, preventing **close** and **read** from interleaving.

(Aaron Greenhouse)

Reaction to bug fix

"This really sucks. Now just to convert to [JDK 1.2] I've got to rewrite code that has worked since JDK 1.02… It's pretty obvious that syncing close would break things."

Comment in Bug ID #4225348:

"Attempt to close while reading causes deadlock"

Why was everyone so mad?

- Java socket programming idiom that requires the ability to **close** mid-**read:** "Hung" socket stream: Use separate thread to **close** and interrupt "hung" **read** or **write**
- In other words: clients assumed **read** and **close** can interleave!
  - Bug fix *prevents* interleaving.
  - Intent inferred — is it correct?
- Design choices — What is/was the design intent?
  - Interleaving **intended** — Fix race while allowing interleaving
  - Interleaving **not intended** — Provide alternative idiom to get the same effect.
- **What should the Java designers have done? What's a good solution to this problem? Whose fault was it?**

Upshot

- Fix was *undone* in JDK1.3
  - Re-enabled socket idiom.
  - Compromises safety of the class by re-enabling the race condition
- **BufferedInputStream** was fixed to both prevent the race and allow socket idiom for JDK 1.5
- Issue #1 – Race condition in deployed production library code
- Issue #2 – Lack of documentation of design intent with respect to concurrency.
- **Moral: bugs are hard, and correctness depends on context and user expectations.**

Summary: take 15-313!

- Software Engineering in practice requires consideration of numerous issues---technical and social---above the level of individual class design/implementation.
- Do you think this is interesting? 15-313, Foundations of Software Engineering is offered in the Fall.
- And consider the undergraduate SE minor!