Complexity revisited: learning from failures

Frans Kaashoek and Robert Morris

Lec 26 --- Last one!

5/13/09

Credit: Jerry Saltzer

6.033 in one slide

Principles: End-to-end argument, Modularity, …

- Client/server
- RPC
- File abstraction
- Virtual memory
- Threads
- Coordination
- Protocol layering
- Routing protocols
- Reliable packet delivery
- Names
- Atomicity
- Transactions
- Replication
- Sign/Verify
- Encrypt/Decrypt
- Authorization

Case studies of successful systems: UNIX, X Windows, MapReduce, Ethernet, Internet, WWW, RAID, DNS, ….

hidden

- we showed principles, techniques, cases
- result of years of experience
- helpful -- yet far from sufficient!
- crucial org/mgmt techniques
- not 033 topic, but closely related
- illustrate via failure: memorable, educational

Today:
Why do systems fail anyway?

- Complexity has no hard edge
- Learning from failures: common problems
- Fighting back: avoiding the problems
- Final admonition

Too many objectives

- Ease of use
- Availability
- Scalability
- Flexibility
- Mobility
- Security
- Networked
- Maintainability
- Performance
- Cheap
- ….

But no systematic methods to synthesize systems to meet objectives

Many objectives

+

Few Methods

+

High d(technology)/dt

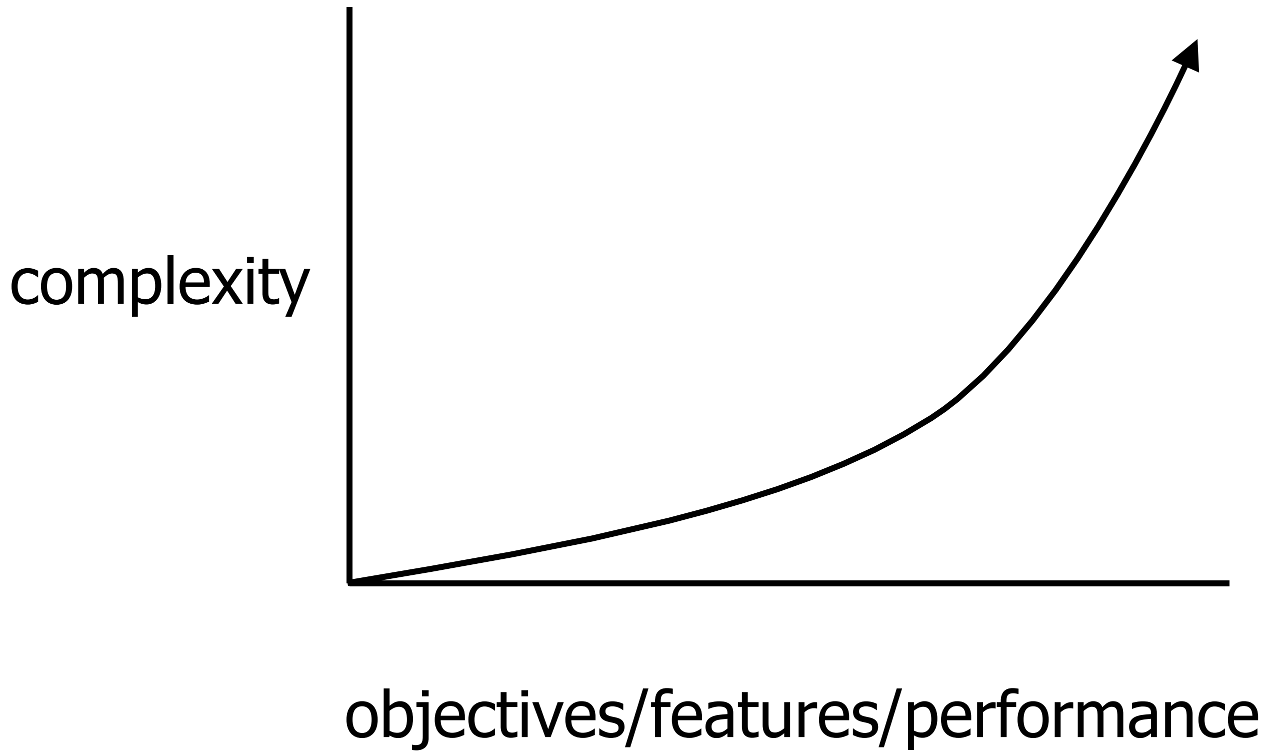
=

High risk of failure

The tarpit

[F. Brooks, Mythical Man Month]

Complexity: no hard edge



- When is it too much?

hidden

- this will happen to your projects
- you must notice in time!
- but how?
- Experience!

Learn from failure!

"The concept of failure is central to design process, and it is by thinking in terms of obviating failure that successful designs are achieved..."
[Henry Petroski]

hidden

- quote from neat book about failure
- engineering a very human undertaking
- all projects have problems, design flaws, bugs
  - progress comes by taking risks → failure
- good engineering about anticipating failure
  - understand the past, learn from it
  - and coping: keeping small failures small

Keep digging principle

- Complex systems systems fail for complex reasons
  - Find the cause …
  - Find a second cause …
  - Keep looking …
  - Find the mind-set.

[Petroski, Design Paradigms]

hidden

- NOT the real answer:
  - “there was a bug”
  - “the operator made an error”
- e.g. Therac-25 and ATM
  - lack of understanding of real problems
  - too little testing, training
  - no feedback into future versions
  - broken organization, management, oversight
- let’s look at some big failures

Pharaoh Sneferu's Pyramid project

Try 1: Meidum (52° angle)

Try 2: Dashur/Bent (52° to 43.5° angle)

Try 3: Red pyramid (right angle: 43°)

hidden

- early example of learning from failure at large scale
- sneferu built three pyramids!
- meidum pyramid
  - originally stepped, filled later, made it more “true”
  - BUT facing fell off during sneferu’s lifetime
- bent pyramid
  - angle change due to failure of meidum pyramid?
- red pyramid
  - starts at 43, less complex internally
  - successful prototype for later “true” pyramids
- ultimately didn’t meet big requirement: eternal rest

United Airlines/Univac

- Automated reservations, ticketing, flight scheduling, fuel delivery, kitchens, and general administration
- Started 1966, target 1968, scrapped 1970, spent $50M
- Second-system effect (First: SABRE)

(Burroughs/TWA repeat)

hidden

- AA’s SABRE (1964?) one of first big “on-line” systems
  - IBM had prior experience w/ SAGE air defense
  - SABRE tightly focused on seat reservation
  - SABRE gave AA a crushing advantage
- United/Univac had no comparable on-line experience
  - but wanted something vastly more capable than SABRE!

CONFIRM

- Hilton, Marriott, Budget, American Airlines
- Linked air + car + hotel reservations
- Started 1988, scrapped 1992, $125M
- Second system
- DB integration problems
- DB not crash recoverable
- Bad-news diode

[Communications of the ACM 1994]

hidden

- SABRE successful -> second system!
- DB integration problems
  - reservations vs yield mgmt (histories &c)
- DB not crash-recoverable
- persistent hiding of schedule slips
  - and 2x under-estimate of running costs
- big consortium, loose oversight

Advanced Automation System

- US Federal Aviation Administration
- To replace 1972 computerized system
- Real-time nation-wide route planning
- Started 1982, scrapped 1994 ($6B)
- Big ambitions
- Changing ideas about UI
- 12 years -> evolving requirements, tech
- 12 years -> culture of not finishing
- Big -> congressional meddling

London Ambulance Service

- Ambulance dispatching
- Started 1991, scrapped in 1992
  - 20 lives lost in 2 days
- No testing/overlap with old system
- Required big changes in procedure
- Users not consulted during design
- Unrealistic schedule (5 months)
- Perhaps first of kind, no experience

[Report of the Inquiry Into The London Ambulance Service 1993]

hidden

- a neat system: loc track, optimized dispatch
- not tested, little training, changed procedures
- congestion collapse on first day
  - inaccurate/old status / position
  - suboptimal amb chosen, two sent, &c
  - so lower capacity, longer delays
  - people called multiple times
  - repeat dispatches, even less efficient
  - no good plan for reverting to backup system
- but real issues were mgmt/planning, not tech
- 100% manual -> 100% auto in one leap

IBM Workplace OS

- One microkernel O/S for all IBM products
  - PDAs / desktop / servers / supercomputers
  - “personalities” for OS/2, AIX, OS/400, Windows
  - x86, new PowerPC, ARM
- Started in 1991, scrapped 1996 ($2B)
- factoring out common services too hard
- PPC needed new OS, new OS needed PPC
  - but PPC was late, buggy, and slow
- IBM division per personality, bad cooperation

[Fleisch HotOS 1997]

hidden

- ambitious / cool idea
- binary compatibility with existing windows &c apps
  - binary translation, APIs
- each aspect well within reach by itself
- common services too hard
  - e.g. pull virt mem out of Windows &c into service
  - too hard to get personalities to agree on services
- OS needed PPC: otherwise too slow
- PPC needed OS: otherwise incompatible
- maybe virtual machines were the right answer
- caused IBM to give up idea of building its own O/Ss

Many more

- Portland, Oregan, Water Bureau, 30M, 2002
- Washington D.C., Payroll system, 34M 2002
- Southwick air traffic control system $1.6B 2002
- Sobey's grocery inventory, 50M, 2002
- King's County financial mgmt system, 38M, 2000)
- Australian submarine control system, 100M, 1999
- California lottery system, 52M
- Hamburg police computer system, 70M, 1998
- Kuala Lumpur total airport management system, $200M, 1998
- UK Dept. of Employment tracking, $72M, 1994
- Bank of America Masternet accounting system, $83M, 1988,
- FBI virtual case, 2004.
- FBI Sentinel case management software, 2006.

Recurring problems

- Excessive generality and ambition
- Second-system effect
- Bad modularity
- Inexperience (or ignoring experienced advice)
- Bad-news diode
- Mythical Man Month

Fighting back: control novelty

- Only one big new idea at a time
- Re-use existing components
- Why it's hard to say "no"
  - Second-system effect
  - Technology is better
  - Idea worked in isolation
  - Marketing pressure
- Hire strong, knowledgeable management

Fighting back: adopt sweeping simplifications

- Processor, Memory, Communication
- Dedicated servers
- Best-effort network
- End-to-end error control
- Atomic transactions
- Authentication, confidentiality

Fighting back: design for iteration, iterate the design

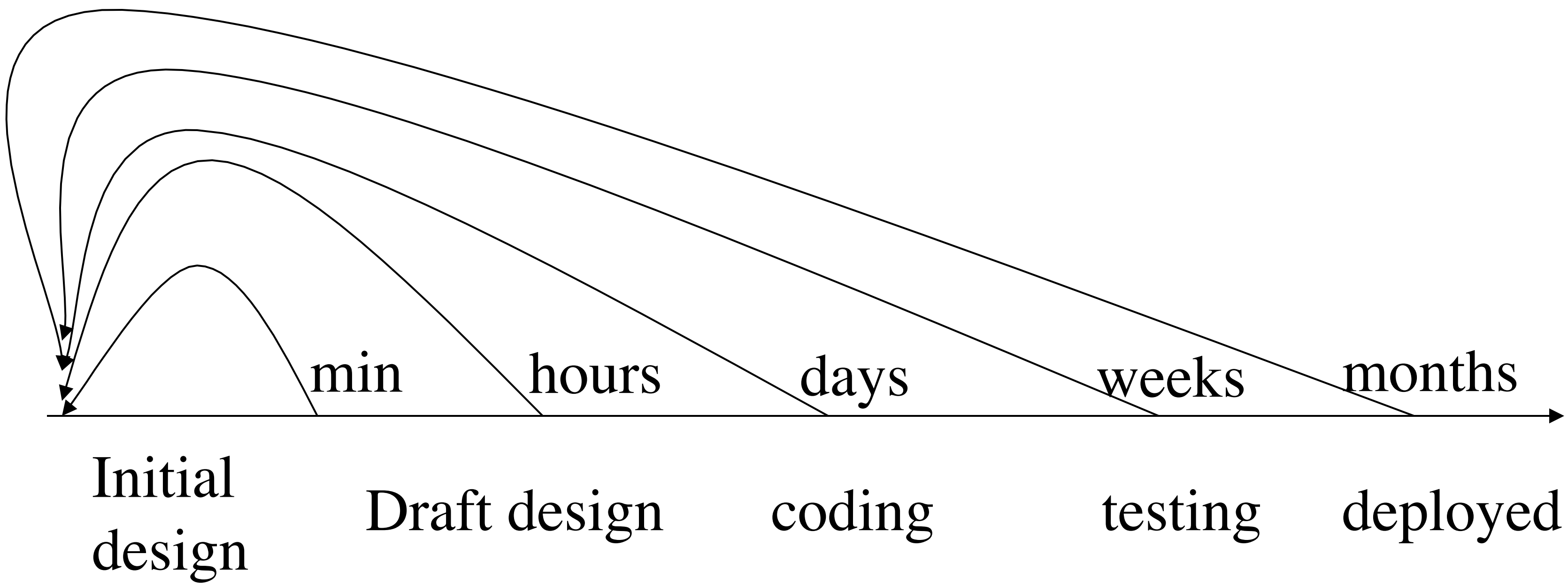
- Get something simple working soon
  - Find out what the real problems are
- Structure project to allow feedback
  - e.g. deploy in phases
- Series of small projects

"Every successful complex system is found to have evolved from a successful simple system" – John Gall

Fighting back: find bad ideas fast

- Question requirements
  - “And ferry itself across the Atlantic” [LHX light attack helicoper]
- Try ideas out, but don’t hesitate to scrap
- Have a design loop

The design loop



- Find flaws fast!

Fighting back: find flaws fast

- Plan and simulate
  - Boeing 777 CAD, F-16 flight sim
- Design reviews, coding reviews, regression tests, daily/hourly builds, performance measurements
- Design the feedback system:
  - Alpha and beta tests
  - Incentives, not penalties, for reporting errors

Fighting back: conceptual integrity

- One mind controls the design
  - Macintosh, Visicalc, UNIX, Linux
- Good abstractions/modules reduce $O(n^2)$ effects
  - In human organization as much as software
  - Small focused teams
- Good esthetics yields more successful systems
  - Parsimonious, Orthogonal, Elegant, Readable, …
- Best designers much better than average
  - Find and exploit them

Summary

- Principles that help avoid failure
  - Limit novelty
  - Adopt sweeping simplifications
  - Get something simple working soon
  - Iteratively add capability
  - Incentives for reporting errors
  - Descope early
  - Give control to (and keep it in) a small design team
- Strong outside pressures to violate these principles
  - Need strong knowledgeable managers

Admonition

Don’t design future failure case studies

Close the 6.033 design loop

https://sixweb.mit.edu/student/evaluate/6.033-s2009

Or https://sixweb.mit.edu