DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.033—Computer System Engineering May 12, 2004

Computer System Design Principles

Adopt sweeping simplifications
So you can see what you are doing.

Avoid excessive generality
If it is good for everything it is good for nothing (Hammer's law)

Design for iteration
You won't get it right the first time, so make it easy to change.

Diminishing returns principle
The more one improves a measure of goodness, the more effort the next improvement requires.

End-to-end argument
The application knows best.

Escalating complexity principle
Adding function adds complexity that is out of proportion

Golden rule of recoverability
Never modify the only copy!

Incommensurate scaling rule
Changing a parameter by a factor of ten requires a new design.

Keep digging
When something goes wrong, there are nearly always several reasons.

Open design principle
Let everyone comment on your design; you need all the help you can get.

Robustness principle
Be tolerant of inputs, strict on outputs.

It is easier to change a module than to change the modularity
Try hard to get the architecture (modularity, abstraction, hierarchy, and layering) right

Safety margin principle
Stay away from the edge of the cliff. But keep track of how far away it is.

Complexity Revisited

6.033 Lecture 26

May 12, 2004

Lecturer: Jerry Saltzer

Saltzer@mit.edu

http://mit.edu/Saltzer

Coping with Complexity

- Sources
- Learning from failure (and success)
- Fighting back
- Admonition

Too many objectives

Not enough systematic methods

Many objectives

+

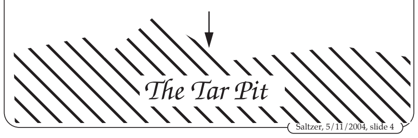
Few methods

+

High d(technology)/dt

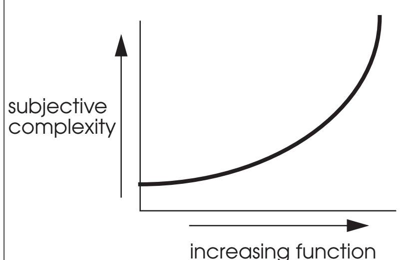
=

Very high risk



Principle of escalating complexity

No hard-edged barrier, it just gets worse…



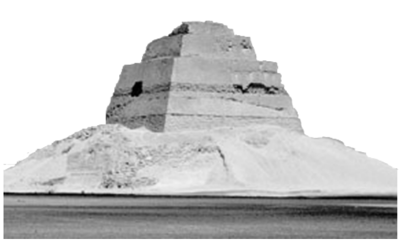
Learn from failure

Try to make original mistakes, rather than needlessly repeating the mistakes of others.

— Donald Rumsfeld

Learn from failure

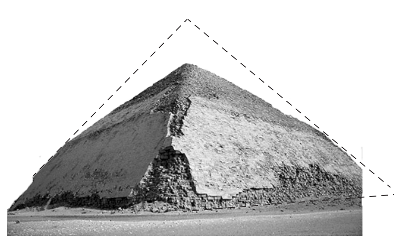
Pharaoh Sneferu's first try



Meidum pyramid
The outer layers collapsed

Learn from failure

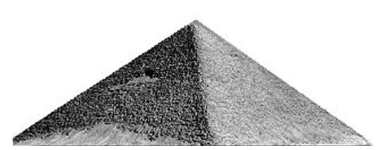
Pharaoh Sneferu's second try



Dashum (bent) pyramid
The plan changed midway, but interior chambers still collapsed.

Learn from failure

Pharaoh Sneferu's third try



Red pyramid
Success, design used in all later pyramids

Keep Digging Principle

Complex systems fail for complex reasons

Find the cause…

Find a second cause…

Keep looking…

Find the mind–set.

(see Petroski, *Design Paradigms*)

NYC: 2,963 traffic lights

Univac, based on experience in Baltimore and Toronto with 100 lights

started: 1965
scrapped: 1968
spent: $5.4M

- two years behind schedule
- changing specifications
- second-system effect:
  - new, untried sensors
  - new, untried software
  - new, untried algorithms
- incommensurate scaling at 30X

United Airlines/Univac

Automated reservations, ticketing, flight scheduling, fuel delivery, kitchens, and general administration

started: 1966, target 1968
scrapped: 1970
spent: $50M

- Second system: tried to automate everything, including the kitchen sink
- "Enhancement" concurrent with "stabilization"

(repeat: Burroughs/TWA)

CONFIRM

Hilton, Marriott, Budget, American Airlines

Hotel reservations linked with airline and car rental

started: 1988
scrapped: 1992
spent: $125M

- Second system
- Very dull tools (machine language)
- Bad-news diode
- See CACM October 1994, for details

SACSS(California) Statewide Automated Child–Support System

Started: 1991 ($99M)
Abandoned: 1998
cost: $300M

- "Lockheed and HWDC disagree on what the system contains and which part of it isn't working."
- "Departments should not deploy a system to additional users if it is not working."
- "...should be broken into smaller, more easily managed projects..."

Taurus

British Stock Exchange share settlement system

started: 1990
scrapped: 1993
spent: £400M = $600M

- "Massive complexity of the back-end systems…"
- All–or–nothing approach, nothing to show until everything works
- Shifting requirements
- Responsibility disconnected from control
- Bad–news diode in action
- Thorough report in Drummond, *Escalation in Decision–Making* (1996)

IBM Workplace OS for PPC

Mach 3.0 + binary compatibility with AIX, DOS, MacOS, OS/400 + new clock mgt + new RPC + new I/O + new CPU

started: 1991
scrapped: 1996
spent: $2B (est.)

- 400 staff on kernel, 1500 elsewhere
- "Sheer complexity of the class structure proved to be overwhelming"
- Big–endian/little–endian not solved
- Inflexibility of frozen class structure
- report in Fleisch, HOT-OS 1997

Tax systems modernization plan

U.S. Internal Revenue Service, to replace 27 aging systems

started: 1989 (est.: $7B)
scrapped: 1997
spent: $4B

- All–or–nothing massive upgrade
- Systems "do not work in real world"
- Government procurement regulations
- Still trying, little progress...

Advanced Automation System

U.S. Federal Aviation Administration

Replaces 1972 Air Route Traffic Control System

started: 1982
scrapped: 1994
spent: $6B

- Changing specifications
- Grandiose expectations
- Contract monitors viewed contractors as adversaries (might work for payroll)
- Congressional meddling

London Ambulance Service

Ambulance dispatching

started: 1991
scrapped: 1992
cost: 20 lives lost in 2 days of operation, $2.5M

- Unrealistic schedule (5 months)
- Overambitious objectives
- Unidentifiable project manager
- Low bidder had no experience
- Backup system not checked out
- No testing/overlap with old system
- Users not consulted during design

More, too many to list…

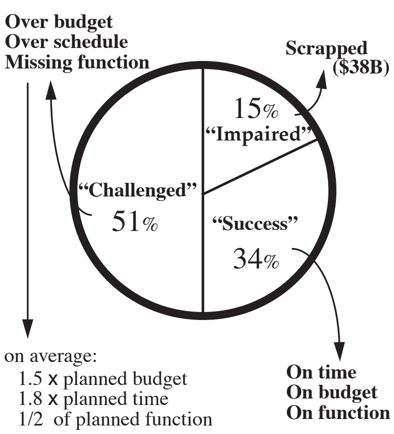
- Portland, Oregon Water Bureau $30M, 2002
- Washington, D.C. payroll system, $34M, 2002
- Southwick air traffic control system, 6 yr -> 12 yr; $600M -> $1.650M, 2002
- Sobey's grocery inventory management system, $50M (2000)
- King's County, CA, financial management system, $38M (2000
- Australian submarine combat control system, $100M, 1999
- California lottery system, $52M
- Hamburg police computer system, $70M, 1998
- Kuala Lumpur total airport management system, $200M (1998)
- U.K. Dep't of Employment tracking system, $72M, 1994
- Bank of America Masternet accounting system, $83M, 1988

Disasters still in progress

(not enough info to understand yet)

- FBI Virtual case file (30 April 2004)
- AT&T Customer service system (7 November 2003)
- British MOD Defense Stores Management System (5 November 2003)

2003 Standish Group study



Recurring problems

- Incommensurate scaling
- Second–system effect
- Excessive generality
- Mythical man-month
- Bad ideas get included
- Wrong modularity
- Bad-news diode

Why aren't abstraction, modularity, hierarchy, and layers enough?

- First, you must understand what you are doing.
- It is easy to create abstractions; it is hard to discover the ***right*** abstraction.
- It is hard to change the abstractions later.

(ditto for modularity, hierarchy, and layers)

Fighting back: Adopt sweeping simplifications

Some modular boundaries work better than others

By chapter…

1: Processors, memory, communication links
2: Dedicated servers
3: *N*–level memories, *N* = 2
4: Best–effort network
5: Delegate administration
6: Signing *and* sealing
7: Fail–fast, pair–and–compare
8: Don't overwrite, append

Fighting Back: Control Novelty

Sources of excessive novelty…

- Second–system effect
- Technology is better
- Idea worked in isolation
- Marketing pressure

Some novelty is necessary; the hard part is figuring out when to say **No**.

Design for Iteration, Iterate the Design

- Something simple working soon
- One new problem at a time
- Feedback is part of the design
- Find ways to find flaws early
- Use iteration-friendly design
- Bypass the bad-news diode
- (Learn from failure)

Fighting back: Find bad ideas fast

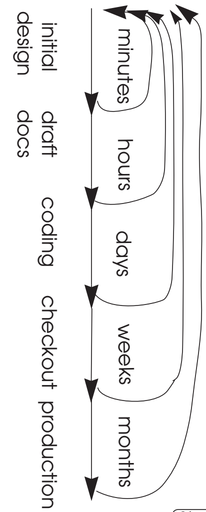
- Question the requirements

  "and ferry itself across the Atlantic" (LHX light attack helicopter)

- Try ideas out—but don't hesitate to scrap them
- Understand the design loop

Requires strong, knowledgeable management

The design Loop



Fighting back: Find flaws fast

- Plan, plan, plan
- Simulate, simulate, simulate
- Design reviews, coding reviews, regression tests, performance measurements
- Design the feedback system e.g., alpha test + beta test; incentives, not penalties, for reporting problems

Use iteration–friendly design methods

- Authentication logic (Ch 6)
- Alibis (space shuttle)
- Failure tolerance models (Ch 7)

General method:

— document all assumptions
— provide feedback paths
— when feedback arrives, review assumptions

Fighting back: Conceptual integrity

- One mind controls the design
  — *Reims cathedral*
  — *Macintosh*
  — *Visicalc (spread sheet)*
  — *Linux*
  — *X Window System*
- Good esthetics yields more successful systems
  — *Parsimony*
  — *Orthogonality*
  — *Elegance*

Obstacles

- Hard to find the right modularity
- Tension: need the best designers—but they are the hardest to manage
- *The Mythical Man–Month* (Brooks' law): Adding more people to a late project makes it later.
- "Our problem is different" COTS versus bespoke

Fighting back: Summary

- Use sweeping simplifications
- Control novelty
- Feedback is part of the design
- Find bad ideas fast
- Use iteration–friendly design methods
- Conceptual integrity

Admonition

Make sure that none of the systems ***you*** design can be used as disaster examples in future versions of this lecture.

6.033 Theme song

'Tis the gift to be simple, 'tis the gift to be free,
'Tis the gift to come down where we ought to be;
And when we find ourselves in the place just right,
'Twill be in the valley of love and delight.

When true simplicity is gained
To bow and to bend we shan't be ashamed;
To turn, turn will be our delight,
Till by turning, turning we come out right.

— *Simple Gifts,* traditional Shaker hymn