Coping with Complexity

SOSP 17

December 14, 1999

Speaker: Jerry Saltzer

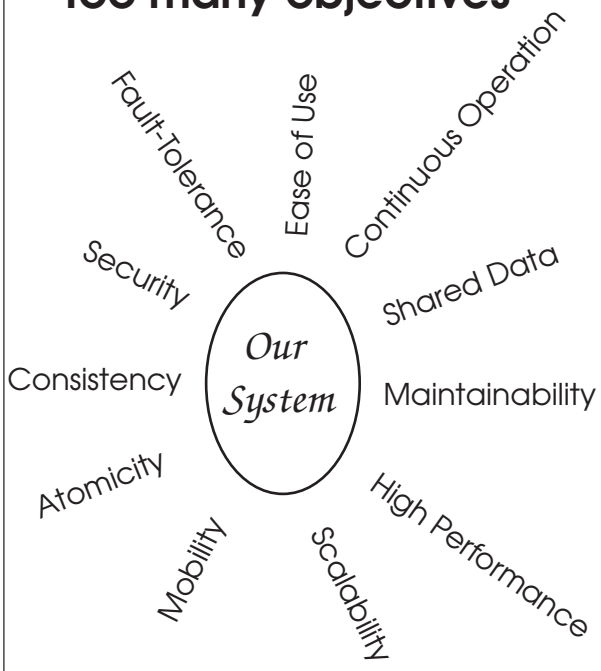
Saltzer@mit.edu

http://mit.edu/Saltzer

Coping with Complexity

- Sources
- Learning from disaster (and experience)
- Fighting back
- Admonition

Too many objectives



Not enough principles

Many objectives

+

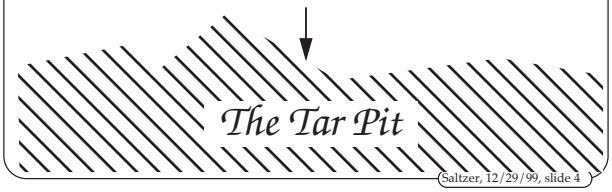
Few principles

+

High d(technology)/dt

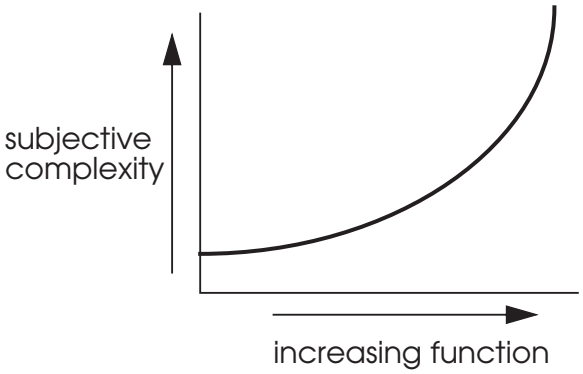
=

Very high risk



No Hard–edged barrier—

it just gets worse…



Learn from failure

(photo)

Pyramid at Meidum

(photo)

Bent Pyramid

Learn from failure

Complex systems fail for complex reasons

Find the cause
Find a second cause
Keep looking
Find the mind–set

(see Petroski, *Design Paradigms*)

NYC control of 10,000 traffic lights

Univac, based on experience in Baltimore and Toronto

started: late 1960's
scrapped: 2-3 years later
spent: ?

- second-system effect:
  - new radio control system
  - new software
  - new algorithms
- based on systems 100X smaller, incommensurate scaling

California Department of Motor Vehicles

Vehicle Registration, Driver's License

started: 1987
scrapped: 1994
spent: $44M

- underestimated cost by factor of 3
- slower than 1965 system
- governor fired the whistleblower
- DMV blames Tandem
- Tandem blames DMV

United Airlines/Univac

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

started: late 1960's
scrapped: early 1970's
spent: $50M

- second system: tried to automate everything, including the kitchen sink

(ditto: Burroughs/TWA)

CONFIRM

Hilton, Marriott, Budget, American Airlines

Hotel reservations with links to Wizard and Sabre

started: 1988
scrapped: 1992
spent: $125M

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

Advanced Logistics System

U.S. Air Force
Materiel and transport tracking

started: 1968
scrapped: 1975
spent: $250M

- second system effect

SACSS(California) State-wide Automated Child Support System

Started: 1991 ($99M)
"on hold": Sept. 1997
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 trading system

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

- "massive complexity of the back-end settlement systems…"
- delays and cost overruns

IBM Workplace OS for PPC

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

Started: 1991
Scrapped: 1996
Spent: $2B

- 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

Tax Systems Modernization

U.S. Internal Revenue Service, replaces 27 aging systems

Started: 1989 (est.: $7B)
Scrapped: 1997?
Spent: $4B

- all–or–nothing massive upgrade
- government procurement regulations

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
- congressional meddling

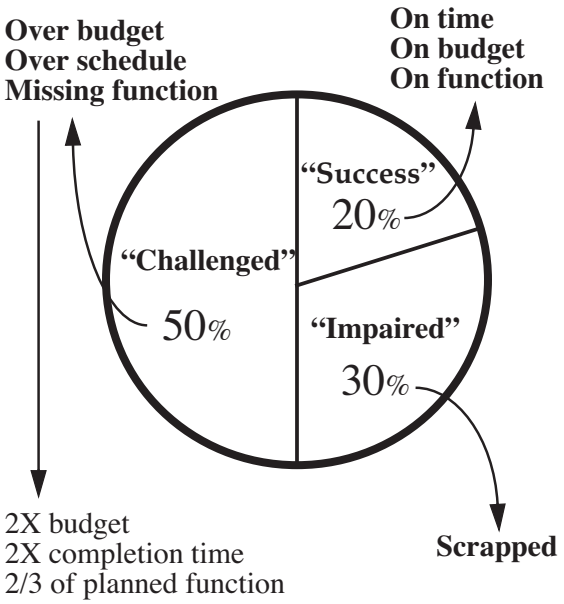
London Ambulance Service

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

1995 Standish Group Study



Recurring problems

- Incommensurate scaling

- Too many ideas

- Mythical man-month

- bad ideas included

- modularity is hard

- bad-news diode

Why aren't abstraction, modularity, hierarchy, and level definition enough?

- First, you must understand what you are doing.

- It is easy to create abstractions; it is hard to discover the ***right*** abstraction.

(ditto for modularity, hierarchy, level definition)

Fighting Back: Control Novelty

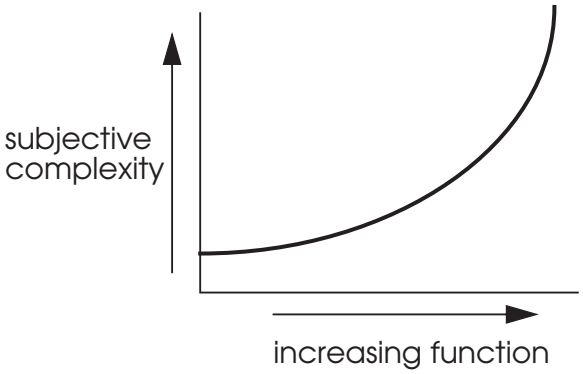
Sources of excessive novelty…

- 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**.

No Hard–edged barrier—

it just gets worse…



Fighting Back: Control Novelty

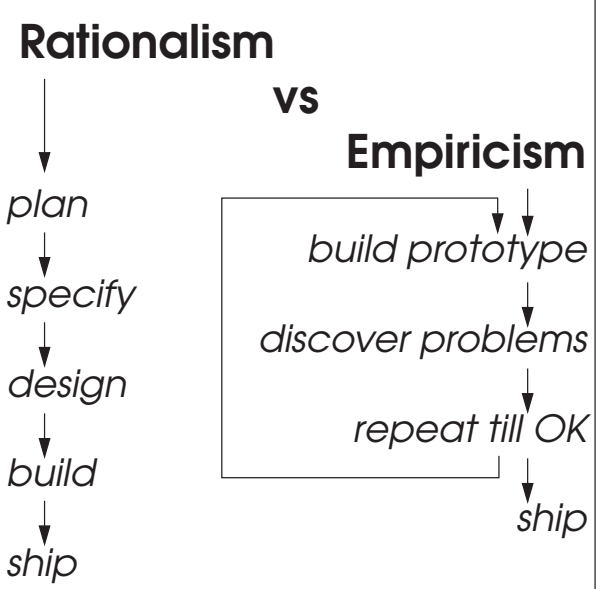
- Something simple working soon
- One new problem at a time

Fighting Back: Feedback

Design for Iteration,
Iterate the Design

- Something simple working soon
- One new problem at a time
- Find ways to find flaws early
- Use iteration-friendly design
- Bypass the bad-news diode
- General: Learn from failure

Brooks's version:



(stolen from Brooks, 1993)

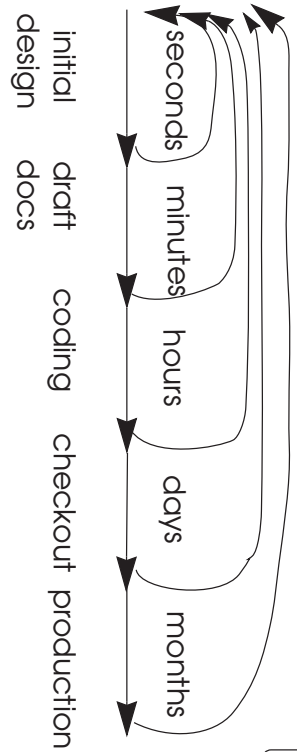
Fighting Back: Find bad ideas fast

- Understand the design loop
- Examine the requirements

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

- Try ideas out—but don't hesitate to scrap them

Requires strong, knowledgeable management



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, no–penalty reports, incentives & reinforcement

Use Iteration–friendly design methods

- Authentication logic (BAN)
- Alibis (space shuttle)
- Error classification (Lampson)

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*
  — *SunOS*
  — *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

Fighting Back: Summary

- Control novelty
- Install Feedback
- 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 talk.