Risks of Computers

“Anything that can go wrong, will.”

As in so many other things, computers intensify the effect. . .

Why

- Speed
- Complexity
- Access
- Arrogance
- Excessive trust

Speed

- Today, things can go wrong at multigigahertz speeds
- Multicore makes them go wrong even faster
- Often too fast for human review or intervention

Complexity

- We generally do not have full understanding of our systems
- There are often unforeseen interactions
- It is rarely possible to test thoroughly

Access

- Many people have remote access to online machines (i.e., most of them), far more than would have access to corresponding non-computerized systems
- Often, the access is not intended by the designers. . .
- There are often inadquate logs

Arrogance

- Designers and programmers think they can build arbitrarily complex—but correct—systems
- They also think they can do it on time and under budget
- Purchasers believe them

Trust

- People *trust* computer output
- “Garbage in, gospel out”
- (“Data in, garbage out”?)

Interactions

- Programs interact internally
- Systems of programs interact with each other
- Programs interact with users
- Programs interact with the environment
- All of these interactions interact with each other!

Users

- Users do *strange* things, things unanticipated by the designer
- There may be surrounding features unknown to the designer, different styles of operation, etc.
- There are often subtle timing issues

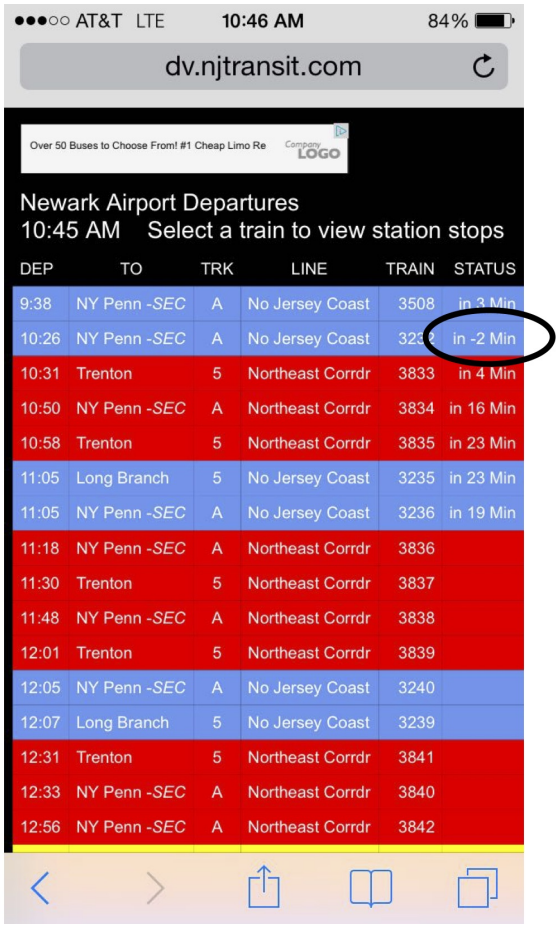
The Environment

- Differences in OS version, configuration, etc., are problematic
- Physical interactions—RF energy, cosmic rays, alpha particles, voltage fluctuations—can lead to failures
- Unanticipated constraints—RAM size, CPU speed, free disk space, etc.—can cause trouble

Error Handling

- Programmers often do a poor job handling errors
- “That can’t happen”. . .
- Is it detected? What is the response? To panic? To die a horrible death? To recover?
- How do you test, especially if it’s a hardware failure indication
- Sometimes, misbehaving hardware *really* misbehaves

NJ Transit Status Display



Bugs Happen

- It's hard to test for all possible failure conditions
- Many problems are caused by combinations of bugs
- Complex systems fail for complex reasons

Example: 2003 Northeast Blackout

- Multiple causes!
- The operators didn't fully understand their system
- The monitoring computers failed
- Power lines failed—and as some failed, other had to carry more of the load, so they heated up and sagged.
- It was a warm day and the wind died, so there was less cooling; this made them sag more—until one touched a tree
- Poor real-time data caused a cascade. . .

The Computer Failure

- The primary alarm server failed, because the alarm application failed and/or because of too much data queued for remote terminals
- The system properly failed over to the backup server
- But—the alarm application moved its data to the backup server, so it crashed, too. . .

Reliability is Hard

- Sometimes, critical systems are engineered for robustness
- Adding such features adds complexity
- This in turn can cause other kinds of failures

Example: the Space Shuttle

- The shuttle had four identical computers for hardware reliability, plus another running different code
- The four were designed to have no single point of failure—which meant that they couldn't share *any* hardware
- A voting circuit matched the outputs of the primary computers; the crew could manually switch to the backup computers
- But—a common clock would violate the "no single point of failure rule"...

What Time is It?

- In a hard real-time system like space shuttle avionics, *something* will always happen very soon
- Take the value of the first element in the timer queue as “now”
- However, there must be a special case for system initialization, when the queue is empty
- A change to a bus initialization routine meant that $1/67$ of the time, the queue wouldn’t be empty during certain crucial boot-time processing
- This in turn made it impossible for the back-up computer to synchronize with the four primaries
- They scrubbed the very first launch, before a world-wide live TV audience, due to a software glitch

The Basic Reliability Mechanism Did Work

- On the first “drop test” of the Enterprise, one computer did fail
- Vibration broke a solder joint
  (`http://www.universetoday.com/100050/the-lessons-we-learned-from-space-shuttle-enterprise/`)
- The other three computers did their job
- (Could their hardware have failed the same way?)

Example: the Phone System

- In January 1990, a processor on an AT&T phone switch failed
- During recovery, the switch took itself out of service
- When it came back up, it announced its status, which triggered a bug in *neighboring switches'* processors
- If those processors received two call attempts within 1/100th of a second, they'd crash, causing a switch to the backup processor
- If the backup received two quick call attempts, *it* would crash
- When those processors rebooted, they'd announce that to their neighbors. . .
- The root cause: a misplaced **`break`** statement
- The failure was a *systems* failure

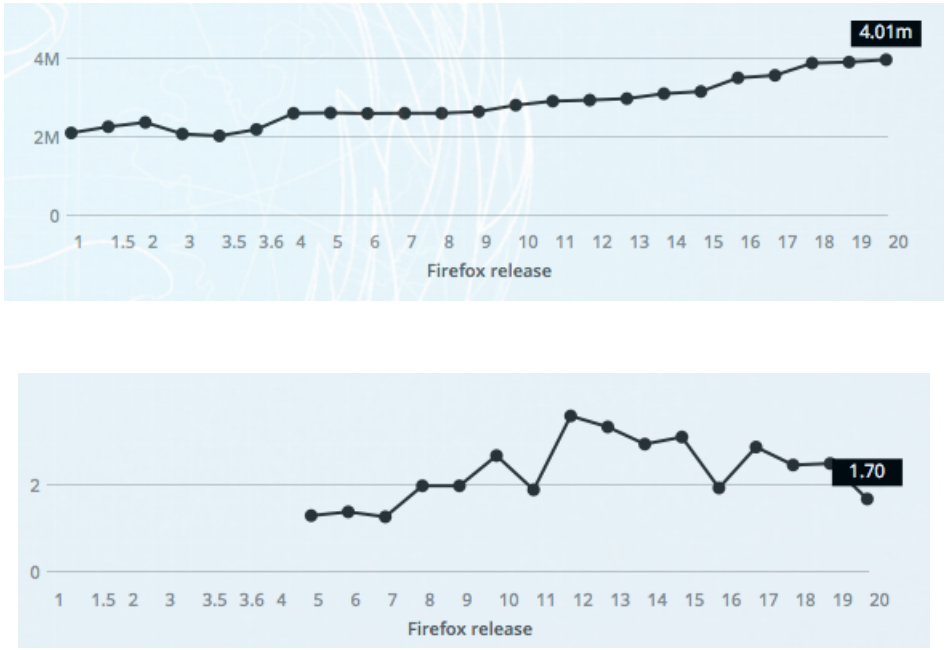
Programs and Systems

- Two levels of problems, code and systems
- Code is buggy
- Systems are also buggy!

Buggy Code

- There seems to be an irreducible minimum bug rate in code
- Our code is *always* buggy

Firefox, April 2013



(From `http://www.almossawi.com/firefox/`)

4M lines of code, 1.7 bugs/Kloc $\Rightarrow$ about 6800 bugs—
and because of the methodology, that's an underestimate.

Systems are More Complex

- Brooks (*The Mythical Man-Month*) distinguishes between programs, program products, program systems, and program system products
- He estimates that the transition to "system" or "product" costs $3\times$ and that "system product" is therefore $9\times$
- Why?

System Complexity

- Many different component, each of which is a complex program
- The components have to interact—will that work?
- ☞] Is the interface between the components precisely and correctly specified?
- The different comnoents likely share common subprocedures, which means that these have to have correct, precise interfaces *and* they must meet the needs of each component of the whole system

☞ Fixing a shared piece to repair a bug in one component could break a different one

N-Version Programming

- Common assumption: have different programmers write independent versions; overall reliability should increase
- But—bugs are correlated
- Sometimes, the specification is faulty—estimates range from 25–50%
- (“Flaws” versus “bugs”)
- Other times, common misperceptions or misunderstandings will cause different programmers to make the same mistake

Thinking Alike

- Many things are taught in standardized ways (often because of popular texts or current technical fads)
- ☞ Kernighan and Plauger's *Elements of Programming Stle* relied on examples of bad code from other books
- Cultural biases: left-to-right versus right-to-left

Achieving Reliability

- All that said, there are some very reliable systems
- Indeed, the space shuttle's software has been widely praised
- The phone system almost always works (though of course not always)
- How?

The Phone System

- A 1996 study showed four roughly-equal causes of phone switch outage: hardware error, software error, operator error, and miscellaneous
- The hardware was already ultrareliable—how did they get the software failure rate that low?
- A lot of hard work and good design—which is reflected in the wording of the question

Switch Design

- Primary goal: keep the *switch* working at all times
- No single call is important
- If anything appears wrong with a call, blow it away
- The caller will mutter, but retry—and the state in the phone switch will be so different that the retry will succeed
- Plus—lots of error-checking, roll-back, restart, etc.
- All of this is *hideously* expensive

The Problem

- We are building—and relying on—increasingly complex systems
- We do not always understand the interactions
- The very best systems are very, very expensive—and even they fail on occasion

Societal Implications

- We can't trust computers too much
- On the other hand, we can't trust them too little, either
- It's easy to point at places where computer failures have caused harm
- It's hard to point to specific cases where they've helped—but statistically, they undoubtedly have
- Where is the balance point?
- It is difficult, a priori, to tell

Self-Driving Cars

- Humans are terrible drivers
- They get distracted, doze off, drink, etc.
- Computers will probably be better
- But—it's all but certain that on occasion, buggy software will cause a fatal crash
- That will be visible, but the lives saved will not be
- What will happen? What should happen?

What Do We Do?

- We know that certain things help: good management practices, sound development environment environments, etc.
- We know that other things hurt: a poor notion of what the system should do, fuzzy specs, many changes during development, etc.
- Usability, in a particular environment, also matters

Drug Dispensing in Hospitals

- (I *assume* you've all read the article. . .)
- The older, manual system was very error-prone, and probably killed people
- The problem, though, is very complex: there are *very* many options, and no "do what I mean" checkbox
- The failure described in the article was a *systems* failure

Systems Failures

- The problem is not caused by any one failure
- Rather, it took a cascade of decisions and failures:
  - A non-standard dosing regimen, causing an alert
  - A menu selection error by the doctor
  - Bad working conditions for the pharmacist
  - "Alert fatigue"
  - An inexperienced nurse, working in an unfamiliar ward
  - A research environment with many non-standard protocols
  - Two children in the family being hospitalized at the same time
- Fixing any of those would have helped

Lack of Backup Mechanisms

- What should happen if the computer system fails?
- This is hard to design for and hard to test; by design, failures are rare events
- Can the pilots fly the plane without the computers?
- Can the doctors do the surgery without the surgical robots?
- A few years ago, Amtrak's master computer system failed—and they couldn't update their web site to say so, because the failed system was how they programmed the web site
- But—the conductors could cope, even when passengers couldn't get their tickets from booths or kiosks

What is Your Plan B?

- You need to understand how to cope with the partial or complete failure of your computer systems
- The more reliant you are on them, the more critical this is
- The more closely linked your computers are, the more likely a failure will cascade