Risks of Computers: What do we Do?

We Have Problems. . .

- Software is buggy
- It takes too long to develop
- It's generally over budget
- What do we do?

Custom Systems

- Even the big vendors of mass market software have such problems
- A lot of organizations rely on custom systems
- Custom system designs are frequently worse

Conversions

- Different programs
- Different file formats
- Different processes
- Different things to back up
- Different inputs and outputs
- Repeat for each program in your system

We're Dealing with *Systems*

- The hard part isn't one program on one machine
- Generally, it's many different programs
- They all interact
- "You can't eat just one peanut"

The Bloomberg Crash

- The Bloomberg data terminal system—relied on by financial personnel around the world—was down for several hours on Friday
- "'Bloomberg connects 100 percent of the Street, and all that human intelligence is what makes markets hum'" (NY Times)
- The "problem was caused by 'a combination of hardware and software failures,' accompanied by failure in the company's 'multiple redundant systems.'"
- "[T]he trader said: 'Problems? Simple: No prices. Nothing. So you can't do anything at all.'"
- "[O]ther traders said they depended on Bloomberg chat so heavily that they weren't able to easily reach out to clients through other means."

Lessons

- Multiple components failed
- Almost certainly, there was a strange interaction or a sequence that wasn’t anticipated
- Total reliance on one platform
- No usable backup system

A Tale of Two Airlines. . .

- WestJet and JetBlue wanted to switch to new, more capable reservations systems
- They independently selected the same new system
- WestJet went first
- "Despite months of planning . . . its Web site crashed repeatedly and its call center was overwhelmed. It took months to resolve all the issues." (WSJ, 12 Apr 2010)

Why?

- Massive data conversion: 840,000 files for existing reservations
- Too much complex hand-processing to convert files
- Too many passengers with reservations, despite the preemptive cancelation of some flights
- For competitive reasons, they didn't announce the conversion until that day

Recovery

- Apology letters
- Flight credits
- Outsourced temporary call center
- A “three- to six-month recovery process”

JetBlue Went Second

- They picked a light-traffic weekend
- They kept their planes abnormally empty
- They developed and deployed a backup web server (it was needed twice)
- They hired 500 temporary agents to handle routine calls, while their own, experienced staff handled complex situations
- The extra agents stayed for two months—“one of the wisest investments we made”
- There were still a few problems

The Differences

- Lighter load (though JetBlue is a bigger airline)
- Backup systems
- Personnel who were trained for conversion glitches

What Wasn’t a Problem

- For the most part, this did not involve new software development
- They were switching to an existing, well-tested product
- (There was undoubtedly some custom software written, both for the conversion and for customization)
- JetBlue spent $40M—$25M in capital costs and $15M in one-time operating expenses
- The problem was *conversion*

Conversion Principles

- Conversions never go smoothly
- If possible, run old and new systems in parallel for a while
- Lighten load
- Automate as much of the data conversion as possible
- Train people for the effort
- Temporarily shed non-essential features

Conversions are *Still* Hard

- When United and Continental merged, they had serious computer problems 20 months after the merger, and two months after the new system went live
- “Delta, which acquired Northwest in 2008, integrated its various computer systems in stages, a process that went relatively smoothly. United elected to tackle a bunch of integration tasks all at once.”

Why is This a Societal Issue?

- When large-scale—or governmental—systems are converted, the public at large is converted
- It costs money and poor service—and we all pay
- Too often, the conversions are postponed or mismanaged

The FBI

- The FBI’s computer systems were *old*
- As of 2002, their network ran “bisync”, a protocol that was obsolete by 1980
- (They had to buy spare parts on eBay!)
- The Trilogy project, the attempt to upgrade, was an unmitigated disaster

What the FBI Did Wrong

- No user involvement—they didn’t know what they wanted the system to do
- Requirements change—post-9/11, they realized they needed much more support for intelligence work
- No prototyping
- Inadequate IT management ability
- Plans for a flash cut

The IRS Has Problems, Too

- Some of their systems are more than 40 years old, and are written in IBM mainframe assembler language
- They spent $3 *billion* trying—and failing—to upgrade their systems in the 1990s
- The problem: “inadequate management, ill-defined goals, repeated cost overruns, and failure to meet deadlines and expectations.” (CNET News)

Autonomous Systems (“Robots”)

- Autonomous systems are now getting a lot of attention
- They all run on software
- Many will interact with each other
- What is likely to happen?

Predictions

- Utterly certain: there will be bugs in the software
- Utterly certain: the devices will sometimes encounter situations that weren’t anticipated
- How about: a drone and a guy in a flying lawn chair (`http://en.wikipedia.org/wiki/Larry_Walters`)?
- Or a self-driving car fording a stream (`http://en.wikipedia.org/wiki/File:Ogle_County_IL_White_Pines_State_Park_Fords3.jpg`)?
- What if a strange situation triggers a software bug?

What About Safety?

- Often, the autonomous options will be safer *statistically*
- Robots don't drink, get distracted (though their CPUs can be overloaded), fall asleep, etc.
- (The Apollo 11 landing was almost aborted because of a CPU overload: `https://www.hq.nasa.gov/alsj/a11/a11.1201-fm.html`)
- However—it is quite likely that there will also be some accidents that a human would have avoided
- We can't point to specific accidents avoided—but we can point to specific crashes

What is the Right Tradeoff?

- How should this question be approached?
- The only people who deal with things like this statistically are insurance actuaries
- We can try to make the systems failsafe—but it's not likely we'll always succeed
- Should such systems be deployed? How should the consequent tragedies be handled?

Requirements for Successful Software

- Clear goals
- A clean architecture
- Very good management
- Enough time
- A deployment plan
- A plan to cope with partial or total failures
- *Don't* leave people out of your planning

Goals

- Meet the real needs of the organization
- Establish the goals up front, and don’t keep changing them
- Have the political fight up front, before you spend money on software

Architecure

- All that stuff you’ve learned about software engineering really helps
- Remember that needs—and hence software—will change over time
- Clean system designs will smooth that process, too

Deployment Plans

- *Assume* that things will go wrong
- Plan for this—how will you cope?
- If you're running old and new systems in parallel, what if they disagree?
- How do you share data between different versions?

Process Matters

- People are part of the system, too
- In understanding the functions and security of a system, it is important to understand what people do
- People are also much more flexible in coping with mistakes—*if* the process lets them
- In some sense, “process” is the way you program the people. . .

Large Software Systems

- Be afraid. Be very afraid.
- It *never* works well at first
- But there's "bad" and "worse"
- Expect this and plan for it