LECTURE 8: Software Reliability

(Some slides are from M. Quinn, Ethics for the Information Age, Pearson © 2013.)

Lecture Overview

• Introduction
• Data-entry or data-retrieval errors
• Software errors
• Notable software system failures
• Therac-25
• Software engineering
• Software warranties

Introduction

• Computer systems are sometimes unreliable
  – Erroneous information in databases
  – Misinterpretation of database information
  – Malfunction of embedded systems
• Effects of computer errors
  – Inconvenience
  – Bad business decisions
  – Fatalities
Data-Entry or Data-Retrieval Errors

Two Kinds of Data-related Failure

- A computerized system may fail because wrong data entered into it
- A computerized system may fail because people incorrectly interpret data they retrieve

Disfranchised Voters

- November 2000 general election
- Florida disqualified thousands of voters
- Reason: People identified as felons
- Cause: Incorrect records in voter database
- Consequence: May have affected election’s outcome
False Arrests

- Sheila Jackson Stossier mistaken for Shirley Jackson
  - Arrested and spent five days in detention
- Roberto Hernandez mistaken for another Roberto Hernandez
  - Arrested twice and spent 12 days in jail
- Terry Dean Rogan arrested after someone stole his identity
  - Arrested five times, three times at gun point

Accuracy of NCIC Records

- March 2003: Justice Dept. announces FBI not responsible for accuracy of NCIC information
- Exempts NCIC from some provisions of Privacy Act of 1974
- Should government take responsibility for data correctness?

Dept. of Justice Position

- Impractical for FBI to be responsible for data’s accuracy
- Much information provided by other law enforcement and intelligence agencies
- Agents should be able to use discretion
- If provisions of Privacy Act strictly followed, much less information would be in NCIC
- Result: fewer arrests
Position of Privacy Advocates

- Number of records is increasing
- More erroneous records → more false arrests
- Accuracy of NCIC records more important than ever

Utilitarian Analysis: Database of Stolen Vehicles

- > 1 million cars stolen every year
  - Owners suffer emotional, financial harm
  - Raises insurance rates for all
- Transporting stolen car across a state line
  - Before NCIC, greatly reduced chance of recovery
  - After NCIC, nationwide stolen car retrieval
- At least 50,000 recoveries annually due to NCIC
- Few stories of faulty information causing false arrests
- Benefit > harm → Creating database the right action

Software Errors
Errors When Data Are Correct

- Assume data correctly fed into computerized system
- System may still fail if there is an error in its programming

Software Errors Leading to System Malfunctions

- 2001 Qwest sent incorrect bills to cell phone customers – 1.4% of customers affected, charged $600/minute
- Faulty USDA beef price reports – Errors cost between $15-$20 million
- 1996 U.S. Postal Service error caused mail addressed to Patent and Trademark Office to be returned for 2 weeks
- 2009 New York City Housing authority overcharged renters
- 2010 About 450 California prison inmates with “high risk of violence” mistakenly released

Software Errors Leading to System Failures

- 1992 Ambulance dispatch system in London – 20 people died
- 1998 Chicago Board of Trade suspended trading
- 2003 Thailand’s minister trapped in BMW limousine
- 2003 Japan’s air traffic control system failed as well as backup system
- 2003 Los Angeles County + USC Medical Center new laboratory computer system, stopped all ambulances
- 2004 Comair’s Christmas Day shutdown cancelled 1,100 flights (Delta Air Lines)
- 2005 Boeing 777 lost control of auto-pilot
Ethical Analysis: E-Retailer Posts Wrong Price, Refuses to Deliver

- 2003 Amazon.com in Britain offered iPaq handheld computer for £7 instead of £275
- Orders flooded in
- Amazon.com shut down site, refused to deliver unless customers paid true price
- Was Amazon.com wrong to refuse to fill the orders?

Rule Utilitarian Analysis

- Imagine rule: A company must always honor the advertised price
- Consequences
  - Harms
    - More time spent proofreading advertisements
    - Companies would take out insurance policies
    - Higher costs → higher prices
    - All consumers would pay higher prices
  - Benefits
    - Few customers would benefit from errors
- Conclusion
  - Rule has more harms than benefits
  - Amazon.com did the right thing

Kantian Analysis

- Buyers knew 97.5% markdown was an error
- They attempted to take advantage of Amazon.com’s stockholders
- They were not acting in “good faith”
- Buyers did something wrong
Notable Software System Failures

Patriot Missile
1991
• Designed as anti-aircraft missile
• Used in 1991 Gulf War to intercept Scud missiles
• One battery failed to shoot at Scud that killed 28 soldiers
• Designed to operate only a few hours at a time
• Kept in operation > 100 hours
• Tiny truncation errors added up
• Clock error of 0.3433 seconds → tracking error of 687 meters

Ariane 5
1996
• Satellite launch vehicle
• 40 seconds into maiden flight, rocket self-destructed
  • $500 million of uninsured satellites lost
• Statement assigning floating-point value to integer raised exception
• Exception not caught and computer crashed
• Code reused from Ariane 4
  • Slower rocket
  • Smaller values being manipulated
  • Exception was impossible
<table>
<thead>
<tr>
<th>Event</th>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
</table>
| AT&T Long-Distance Network | 1990 | Significant service disruption - About half of telephone-routing switches crashed - 70 million calls not put through - 60,000 people lost all service - AT&T lost revenue and credibility  
| Cause | - Single line of code in error-recovery procedure - Most switches running same software - Crashes propagated through switching network |
| Robot Missions to Mars | 1999 | Mars Climate Orbiter - Disintegrated in Martian atmosphere - Lockheed Martin design used English units - Jet Propulsion Lab design used metric units  
| Mars Polar Lander | - Crashed into Martian surface - Engines shut off too soon - False signal from landing gear |
| Denver International Airport | 1993 | BAE built automated baggage handling system  
| Problems | - Airport designed before automated system chosen - Timeline too short - System complexity exceeded development team’s ability |
| Results | - Added conventional baggage system - 16-month delay in opening airport - Cost Denver $1 million a day |
Tokyo Stock Exchange
2005
• First day of trading for J-Com
• Mizuho Securities employee mistakenly entered order to sell 610,000 shares at 1 yen, instead of 1 share at 610,000 yen
• Employee overrides computer warning
• After sell order posted on exchange’s display board, Mizuho tried to cancel order several times; software bug caused attempts to fail
• Mizuho lost $225 million buying back shares

Direct Recording Electronic Voting Machines
• After problems with 2000 election, Congress passed Help America Vote Act of 2002
• HAVA provided money to states to replace punch card voting systems
• Many states used HAVA funds to purchase direct recording electronic (DRE) voting machines
• Brazil and India have run national elections using DRE voting machines exclusively
• In November 2006 1/3 of U.S. voters used DRE voting machines

Diebold Electronic Voting Machine
Issues with DRE Voting Machines

- Voting irregularities
  - Failure to record votes 2002
  - Overcounting votes 2003
  - Misrecording votes 2004 & 2006
- Lack of a paper audit trail
- Vulnerability to tampering
  - CS prof Herbert Thompson examined centralized Diebold machine: lacked authentication mechanism
  - Inserted 5 lines of code and switched 5k votes from one candidate to another
- Source code a trade secret, can’t be examined
- Possibility of widespread fraud through malicious programming

CASE STUDY
Therac-25

Why this case study?

- Example of system where safety relies solely upon the quality of its embedded software
- Harms are very serious
- Although old, it is very well documented
Describe the Therac-25

Therac-25

- Medical machine generating radiation from linear accelerator
- Radiation treatment commonly used for cancer patients
  - between 50-60% today
- AECL and CGR built Therac-6 and Therac-20
- Therac-25 built by AECL
  - PDP-11 an integral part of system
  - Hardware safety features replaced with software
  - Reused code from Therac-6 and Therac-20
- First Therac-25 shipped 11 systems in 1983
  - Patient in one room
  - Technician in adjoining room
- Very unreliable
  - 40 malfunctions/day
  - Resulted in 6 massive overdoses with three deaths
  - 1987 operations suspended

Chronology of Accidents and AECL Responses

- Marietta, Georgia (June 1985)
- Hamilton, Ontario (July 1985)
- First AECL investigation (July-Sept. 1985)
- Yakima, Washington (December 1985)
- Tyler, Texas (March 1986)
- Second AECL investigation (March 1986)
- Tyler, Texas (April 1986)
- Yakima, Washington (January 1987)
- FDA declares Therac-25 defective (February 1987)
What caused the safety failure?

The Socio-Technical System

<table>
<thead>
<tr>
<th>The Machine</th>
<th>Hospitals and Clinics</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Supporting Systems (video, audio, etc.)</td>
<td>• Doctors, Medical Physicians</td>
</tr>
<tr>
<td>• Hardware</td>
<td>• Management, User Groups</td>
</tr>
<tr>
<td>• Software Systems</td>
<td>• Operators, Reporting Procedures</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Atomic Energy Canada, Limited</th>
<th>Government Medical Device Regulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Management, Reporting Procedures, Design Team, Sales Staff, Support and Field Engineers</td>
<td>• Food and Drug Administration</td>
</tr>
<tr>
<td></td>
<td>• Canadian Radiation Protection Board</td>
</tr>
<tr>
<td></td>
<td>• Reporting Procedures</td>
</tr>
</tbody>
</table>

User Interface
Causes of Safety Failure

- Programming errors
- Poor human computer interaction design
- Inadequate safety engineering
- Lax culture of safety in the manufacturing organization
- Inadequate reporting structure at the company level and as required by the U.S. government

Software Errors

- Race condition: order in which two or more concurrent tasks access a shared variable can affect program’s behavior
- Two race conditions in Therac-25 software
  - Command screen editing
  - Movement of electron beam gun
- Extremely difficult to diagnose and debug race condition

Race Condition Revealed by Fast-typing Operators

[Diagram of race condition]
Code for Detent Task

```
Option:
   if meter/energy specified then
       begin
       calculate valid sink.
       repeat
       fetch parameters
       store in same parameters
       until all parameters set
       end repeat.
   if meter/energy changed then return.
   if data entry is complete then set Tphase to 3
   if data entry is not complete then
       if entry canceled then set Tphase to 3
       return.
```

Race Condition Caused by Counter Rolling Over to Zero

![Diagram of race condition caused by counter rolling over to zero]
Post Mortem
• AECL focused on fixing individual bugs
  – System bugs are interactions
• System not designed to be fail-safe
  – No single point of failure should create catastrophe
  – Need fail-safe solutions that are not software
• No hardware or software to report overdoses
• Software lessons
  – Difficult to debug programs with concurrent tasks
  – Design must be as simple as possible
  – Documentation crucial
  – Code reuse does not always lead to higher quality
• AECL did not communicate fully with customers
  – AECL told physicists in Washington & Texas overdose was impossible despite being sued by overdose patient in Georgia

Were the Therac-25 designers and producers morally responsible?

Moral Responsibility of the Therac-25 Team
• Conditions for moral responsibility
  – Causal condition: actions (or inactions) caused the harm
  – Mental condition
    • Actions (or inactions) intended or willed
    OR
    • Moral agent is careless, reckless, or negligent
• Therac-25 team morally responsible
  – They constructed the device that caused the harm
  – They were negligent
Responsibilities of the Programmer

• To make superiors aware of the dangers inherent in doing safety interlocks only in the software
  – Limits of software only
  – Redundancy
  – Over-ride testing
• Knowledge of professional software practices
  – Using unprotected memory
  – Improper initialization
  – Thoroughly test software for many possible conditions
  – Human-computer interaction as a system

Postcript

• Computer errors related to radiation machines continue to maim and kill patients
• Investigation by *The New York Times*
  • 3 overdoses from linear accelerator, died
  • 27 days radiation overdoses, died

Improving Software Reliability by Software Engineering Practices
Software Engineering Practices

- Specification using system requirements
- Development
  - CASE tools
  - Object-oriented systems have advantages
  - Modular design and implementation
- Validation (Testing)

Validation (Testing)

- Ensure software satisfies specification
- Ensure software meets user’s needs
- Challenges to testing software
  - Non-continuous responses to changes in input
  - Exhaustive testing impossible
  - Testing reveals bugs, but cannot prove none exist
- Test modules, then subsystems, then system
- Simulation
  - Validating software by prediction

Validation by Comparing Predicted and Actual Outcomes

Courtesy of Daimler AG
Validation by "Predicting the Present"

Software Quality Is Improving

Software Warranties
Shrinkwrap Non-Warranties

- Some say you accept software “as is”
- Some offer 90-day replacement or money-back guarantee
- None accept liability for harm caused by use of software

Are Software “Warranties” Enforceable?

- No: company still responsible despite warranty
  - Article 2 of Uniform Commercial Code
  - Magnuson-Moss Warranty Act
  - 1987 Step-Saver Data Systems v. Wyse Technology and The Software Link
- Yes: company not responsible
  - 1994 ProCD, Inc. v. Zeidenberg
  - 1993 Mortensen v. Timberline Software

Moral Responsibility of Software Manufacturers

- If vendors were responsible for harmful consequences of defects
  - Companies would test software more
  - They would purchase liability insurance
  - Software would cost more
  - Start-ups would be affected more than big companies
  - Less innovation in software industry
  - Software would be more reliable
- Making vendors responsible for harmful consequences of defects may be wrong, but…
- Consumers should not have to pay for bug fixes
Overall Conclusions

- Data can be incorrectly entered
- Software inherently has limitations
  - Design (model failure)
  - Software Implementation (bugs)
  - Software Testing (impossible to catch all bugs)
  - Operational and maintenance failures
- Design, implement, test and train for FAILURE