BIO
PRESENTATION

W5

10/18/2006 11:30:00 AM

SOFTWARE DISASTERS AND LESSONS LEARNED

Patricia McQuaid
Cal Poly State University

International Conference on
Software Testing Analysis and Review
October 16-20, 2006
Anaheim, CA USA

Patricia A. McQuaid

Patricia A. McQuaid, Ph.D., is a Professor of Information Systems at California Polytechnic State University, USA. She has taught in both the Colleges of Business and Engineering throughout her career and has worked in industry in the banking and manufacturing industries. Her research interests include software testing, software quality management, software project management, software process improvement, and complexity metrics.

She is the co-founder and Vice-President of the American Software Testing Qualifications Board (ASTQB). She has been the program chair for the Americas for the Second and Third World Congresses for Software Quality, held in Japan in 2000 and Germany in 2005.

She has a doctorate in Computer Science and Engineering, a masters degree in Business, an undergraduate degree in Accounting, and is a Certified Information Systems Auditor (CISA). Patricia is a member of IEEE, a Senior Member of the American Society for Quality (ASQ), and a member of the Project Management Institute (PMI). She is on the Editorial Board for the *Software Quality Professional* journal, and also participates on ASQ's Software Division Council. She was a contributing author to the *Fundamental Concepts for the Software Quality Engineer* (ASQ Quality Press) and is one of the authors of the forthcoming ASQ Software Quality Engineering Handbook (ASQ Quality Press).

Software Disasters and Lessons Learned…

Patricia McQuaid, Ph.D.
Professor of Information Systems
California Polytechnic State University
San Luis Obispo, CA

STAR West October 2006

Agenda for Disaster …

- **Therac-25**
- **Denver Airport Baggage Handling**
- **Mars Polar Lander**
- **Patriot Missile**

Therac-25

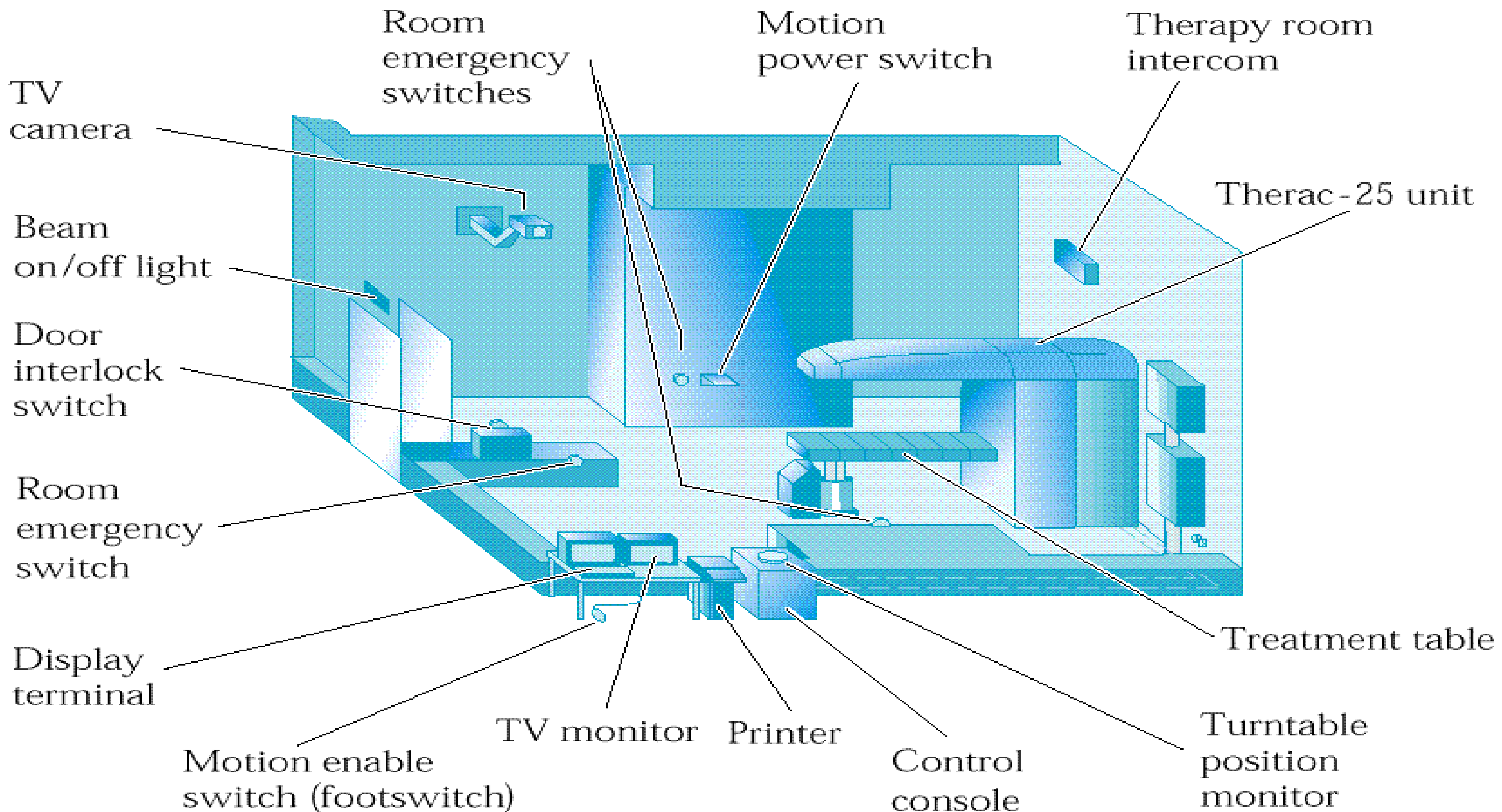
"One of the most devastating computer related engineering disasters to date"

Therac-25

- Radiation doses ➔ RADS
- Typical dosage 200 RADS
- Worst case dosage **20,000 RADS !!!**

6 people severely injured

Machine Design



Accident History Timeline

1. **June 1985, Georgia: Katherine Yarbrough, 61**
   - Overdosed during a follow-up radiation treatment after removal of a malignant breast tumor.
2. **July 1985, Canada: Frances Hill, 40**
   - Overdosed during treatment for cervical carcinoma.
   - "No dose" error message
   - **Dies** November 1985 of cancer.
3. **December 1985 , Washington**
   - A woman develops erythema on her hip

Accident History Timeline (continued)

4. **March 1986: Voyne Ray Cox**
   - Overdosed; next day, severe radiation sickness.
   - 'Malfuction 54'
   - **Dies** August 1986 – radiation burns.

5. **April 1986, Texas: Verdon Kidd**
   - Overdosed during treatments to his face (treatment to left ear).
   - 'Malfuction 54'
   - May 1986: **dies** as a result of acute radiation injury to the right temporal lobe of the brain and brain stem.

6. **January 1987, Washington : Glen A. Dodd, 65**
   - (same location in Washington as earlier woman) overdosed
   - April 1987: **dies** of complications from radiation burns to his chest.

What led to these problems?

- FDA’S “pre-market approval”
- Reliance on the software for safety – not yet proven
- No adequate software quality assurance program
- One programmer created the software
- Assumed that re-used software is safe
- AECL - unwilling to acknowledge problems

1. Poor coding practices
   - Race conditions, overflow problems
2. Grossly inadequate software testing
3. Lack of documentation
4. Lack of meaningful error codes
5. AECL's unwillingness or inability to resolve problems
6. FDA's policies for reporting known issues were poor

Poor User Interface

PATIENT NAME : TEST
TREATMENT MODE : FIX   BEAM TYPE: X   ENERGY (MeV): 25

	ACTUAL	PRESCRIBED	
UNIT RATE/MINUTE	0	200	
MONITOR UNITS	50 50	200	
TIME (MIN)	0.27	1.00	
GANTRY ROTATION (DEG)	0.0	0	VERIFIED
COLLIMATOR ROTATION (DEG)	359.2	359	VERIFIED
COLLIMATOR X (CM)	14.2	14.3	VERIFIED
COLLIMATOR Y (CM)	27.2	27.3	VERIFIED
WEDGE NUMBER	1	1	VERIFIED
ACCESSORY NUMBER	0	0	VERIFIED

DATE : 84-OCT-26   SYSTEM : BEAM READY   OP.MODE : TREAT   AUTO
TIME : 12:55:8   TREAT : TREAT PAUSE   : X-RAY   173777
OPR ID : T25V02-R03   REASON : OPERATOR   COMMAND:

Lessons Learned

- There is a tendency to believe that the cause of an accident had been determined. Investigate more.
- Keep audit trails and incident analysis procedures
  - Follow through on reported errors
- Follow the basic premises of software engineering
  - Complete documentation
  - Established software quality practices
  - Clean designs
  - Extensive testing at module, integration, and system level
- Do NOT assume reusing software is 100% safe

Fear and Loathing in Denver International

Background

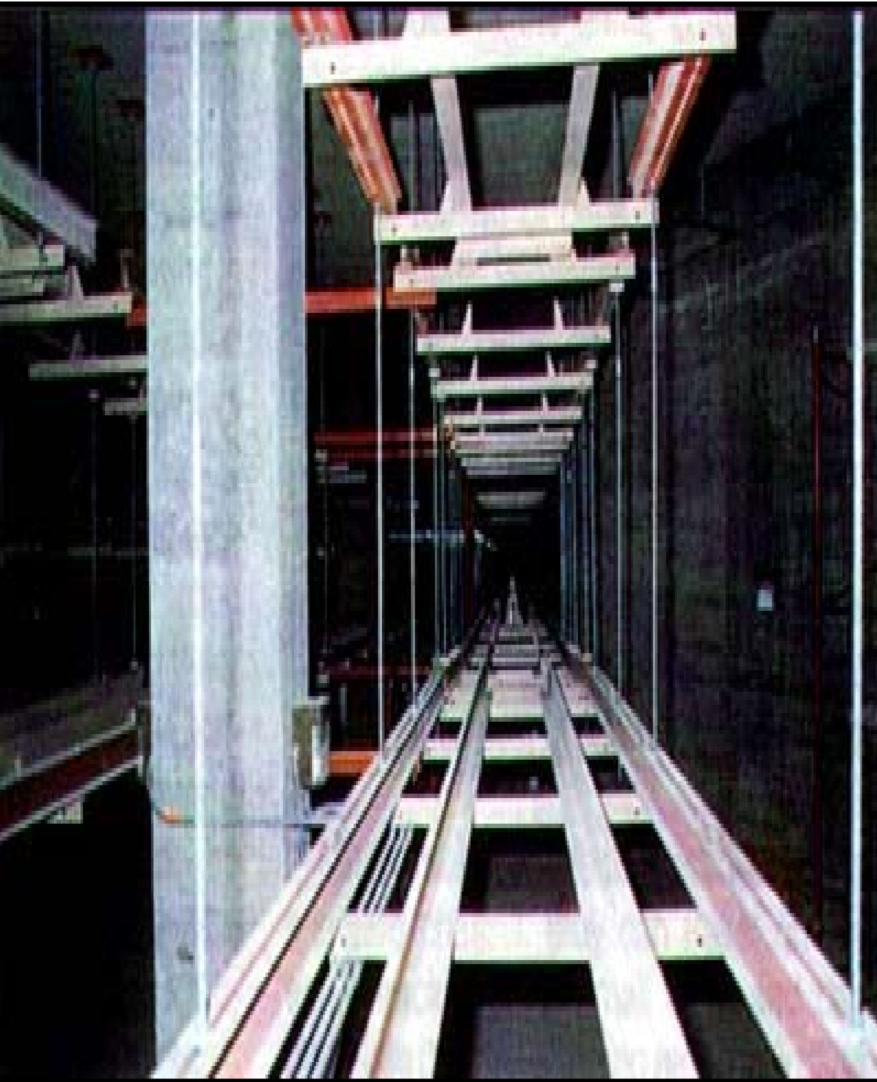
- Opened in 1995
- 8th most trafficked airport in the world
- Fully automated luggage system
- Able to track baggage entering, transferring, and leaving
- Supposed to be extremely fast – 24 mph (3x fast as conveyor systems)
- Uses Destination Coded Vehicles (telecars)
- The plan: 9 minutes to anywhere in the airport
- Planned cost: $193 million
- Actual cost: over $640 million
- Delay in opening the airport: 16 months

System Specifications

Copyright 2004 - TerraServer.com, AirPhoto USA & GlobeXplorer

Denver International Airport
Denver, Colorado

Project Planning



Construction

- Poor planning
- Poor designs
- Challenging construction

Timetable

- 3-4 year project to be completed in 2 years
- Airport opening delayed 16 months

•Coding

- Integrate code into United's existing Apollo reservation system

Cart Failures

- Routed to wrong locations
- Sent at the wrong time
- Carts crashed and jumped the tracks
- Agents entered information too quickly, causing bad data
- Baggage flung off telecarts
- Baggage could not be routed, went to manual sorting station
- Line balancing / blocking
- Baggage stacked up

Hardware Issues

- Computers ➔ insufficient
- Could not track carts
- Redesigned system
- Different interfaces caused system crashes
- Scanners
- Hard to read barcodes
- Poorly printed baggage tags
- Sorting problems, if dirty or blocked
- Scanners crashed into could no longer read
- Faulty latches dumped baggage

Project Costs

Total cost:
over

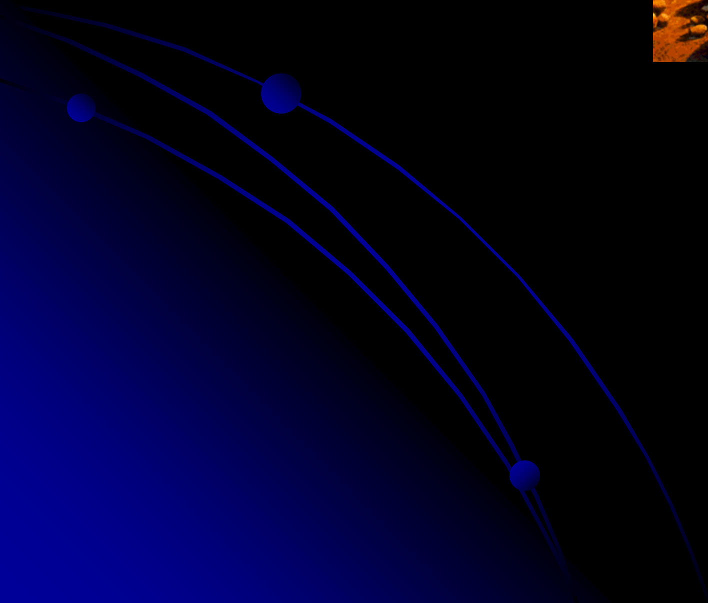
$640 million

A system with **ZERO** functionality…

Lessons Learned

- **Spend time up front on planning and design**
- **Employ risk assessment and risk-based testing**
- **Control scope creep**
- **Develop realistic timelines**
- **Incorporate a formal Change Management process**
- **Enlist top management support**
- **Testing - integration testing; systems testing**
- **Be cautious when moving into areas you have no expertise in**
- **Understand the limits of the technology**

Mars Polar Lander



Why have a Polar Lander?

- Answer the questions:

    Could Mars be hospitable?
    Is there bacteria in the sub-surface of the planet?
    Is it really water that we have seen evidence of or not?

- Follow up on prior missions

    Mars Climate Orbiter 1999
    Mars Surveyor Program 1998

- Plain ole' curiosity

    Do they really look like that? →

Timeline of Mission

- Launched Jan 3, 1999 from Cape Canaveral
- Gets to Mars atmosphere Dec 3, 1999 @ 12:01am PST
- 12:39am PST engineers wait for the signal that is to reach Earth any second…

What happened?

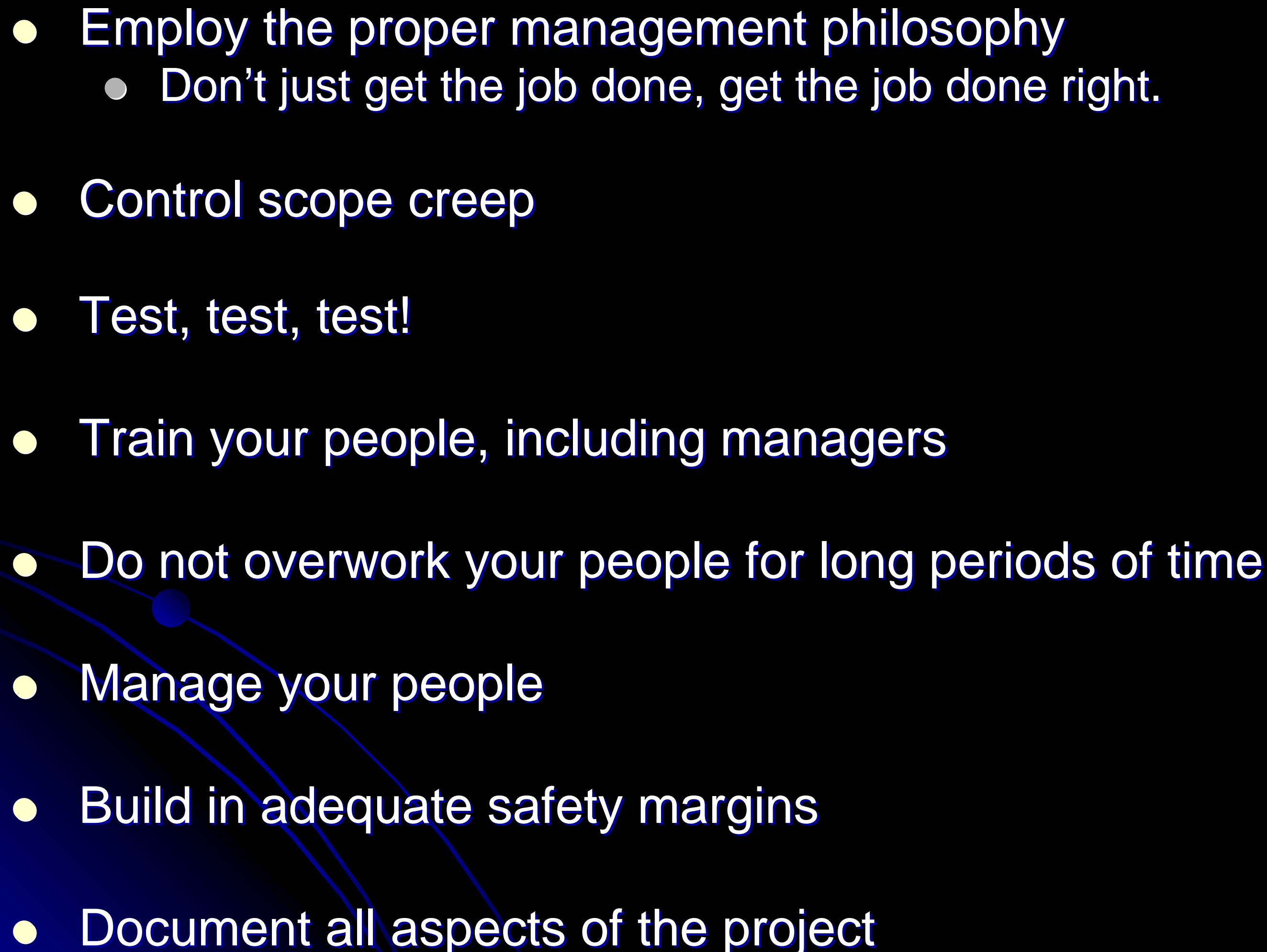
- We don’t know.
  - Unable to contact the $165 million Polar Lander.
  - Various reported “potential” sightings in March 2001, and May 2005.
  - Error in rocket thrusters - landing.
  - Conspiracy theory
    - Type of fuel used
    - Landing gear not tested properly

What went wrong?

- Coding and human errors
  - An error in one simple line of code
    - Shut down the engines too early ➔ crashed
  - Feature creep and inadequate testing
  - Teams were unfamiliar with the spacecraft
  - Some teams improperly trained
  - Some teams drastically overworked
  - Inadequate management
  - No external independent oversight of the teams
  - Software complexity (problem for aerospace industry)

Lessons Learned

- Employ the proper management philosophy
  - Don’t just get the job done, get the job done right.
- Control scope creep
- Test, test, test!
- Train your people, including managers
- Do not overwork your people for long periods of time
- Manage your people
- Build in adequate safety margins
- Document all aspects of the project



Patriot Missile Disaster

Missiles

- 7.4 feet long
- Powered by a single stage solid propellant rocket motor
- Weighs 2,200 pounds
- Range of 43 miles

Launcher

- Transports, points, and launches the missile
- 4 missiles per launcher
- Operated remotely via a link to the control station

Radar

- Carries out search, target detection, track, and identification
- Responsible for missile tracking, guidance, and electronic counter counter-measures
- Mounted on a trailer
- Controlled by a computer in the control area

Operation

1. Target is acquired by the radar.
2. Information is downloaded to the computer.
3. Missile is launched.
4. Computer guides missile to target.
5. A proximity fuse detonates the warhead.

Disaster

February 25, 1991

- Dhahran, Saudi Arabia
- SCUD hits Army barracks killing 28 people and injuring 97 others
- Patriot fails to track the incoming missile; did not even launch!

What Went Wrong?

- Designed as *anti-aircraft*, not *anti-missile*
- Expected: a mobile unit, not a fixed location
- Designed to operate for a few hours at a time
  - Had been running continuously for 4 days
  - After 8 continuous hours stored clock value, off by .0275 seconds=55 meter error
  - For 100 hours stored clock value is off by .3433 seconds = 687 meter error…

Timeline

- February 11
  - Israelis notify US of loss of accuracy problem
- February 16
  - Software fixes made to correct timing error
  - Update sent out to troops
- February 25
  - The attack!
- February 26
  - Update arrives in Saudi Arabia

Lessons Learned

- Robust testing needed for safety-critical software
  - Test the product for the environment it will be used in.
  - Test under varying conditions
- System redesigns – be careful…
  - Special care needed when redesigning a system for a new use
- Clear communication
  - Among the designers, developers, and operators
  - Dispatch fixes quickly!

The End!

Thank you for attending

Software Disasters and Lessons Learned

Patricia McQuaid, Ph.D.
Professor of Information Systems
California Polytechnic State University
San Luis Obispo, CA 93407

Abstract

Software defects come in many forms - from those that cause a brief inconvenience, to those that cause fatalities. It is important to study software disasters, to alert developers and testers to be ever vigilant, and to understand that huge catastrophes can arise from what seem like small problems. This paper examines such failures as the Therac-25, Denver airport baggage handling, the Mars Polar Lander, and the Patriot missile. The focus is on factors that led to these problems, an analysis of the problems, and then the lessons to be learned that relate to software engineering, safety engineering, government and corporate regulations, and oversight by users of the systems.

Introduction

"Those who cannot remember the past are condemned to repeat it", said George Santayana, a Spanish-born philosopher who lived from 1863 to 1952 [1].

Software defects come in many forms, from those that cause a brief inconvenience to those that cause fatalities, with a wide range of consequences in between. This paper focuses on four cases: the Therac-25 radiation therapy machine, the Denver airport baggage handling system, the Mars Polar Lander, and the Patriot missile. Background is provided, factors that led to these problems are discussed, the problems analyzed, and then the lessons learned from these disasters are discussed, in the hopes that we learn from them and do not repeat these mistakes.

Therac-25 Medical Accelerator

The first disaster discussed in this paper deals with the Therac-25, a computerized radiation therapy machine that dispensed radiation to patients. The Therac-25 is one of the most devastating computer related engineering disasters to date. It was one of the first dual-mode medical linear acceleration machines developed to treat cancer, but due to poor engineering, it led to the death or serious injury of six people. In 1986, two cancer patients in Texas received fatal radiation overdoses from the Therac-25. [2]

In an attempt to improve software functionality, the Atomic Energy Commission Limited (AECL) and a French company called CGR developed the Therac-25. There were earlier versions of the machine, and the Therac-25 reused some of the design features of the Therac-6 and the Therac-20. According to Leveson, the Therac-25 was "notably more compact, more versatile, and arguably easier to use" than the earlier versions [3]. However, the Therac-25 software also had more responsibility for maintaining safety than the software in the previous machines. While this new machine was supposed to provide advantages, the presence of numerous flaws in the software led to massive radiation overdoses, resulting in the deaths of three people [3].

According to its definition, "medical linear accelerators accelerate electrons to create high-energy beams that can destroy tumors with minimal impact on the surrounding healthy tissue [3]." As a dual-mode

machine, the Therac-25 was able to deliver both electron and photon treatments. The electron treatment was used to radiate surface areas of the body to kill cancer cells, while the photon treatment, also known as x-ray treatment, delivered cancer killing radiation deeper in the body.

The Therac-25 incorporated the most recent computer control equipment, which was to have several important benefits, one of which was to use a double pass accelerator, to allow a more powerful accelerator to be fitted into a small space, at less cost. The operator set up time was shorter, giving them more time to speak with patients and treat more patients in a day. Another “benefit” of the computerized controls was to monitor the machine for safety. With this extensive use of computer control, the hardware based safety mechanisms that were on the predecessors of the Therac-25, were eliminated and transferred completely to the software, which one will see was not a sound idea [4].

The Atomic Energy Commission Limited (AECL), along with a French company called CGR together built these medical linear acceleration machines. The Therac-25's X-rays were generated by smashing high-power electrons into a metal target positioned between the electron gun and the patient. The older Therac-20's electromechanical safety interlocks were replaced with software control, because software was perceived to be more reliable [5].

What the engineers did not know was that the programmer who developed the operating system used by both the Therac-20 and the Therac-25 had no formal training. Because of a subtle bug called a “race condition”, a fast typist could accidentally configure the Therac-25 so the electron beam would fire in high-power mode, but with the metal X-ray target out of position [5].

The Accidents. The first of these accidents occurred on June 3, 1985 involving a woman who was receiving follow-up treatment on a malignant breast tumor. During the treatment, she felt an incredible force of heat and the following week the area that was treated began to breakdown and to lose layers of skin. She also had a matching burn on her back and her shoulder had become immobile. Physicists concluded that she received one or two doses in the 20,000 rad (radiation absorbed dose) range, which was well over the prescribed 200 rad dosage. When AECL was contacted, they denied the possibility of an overdose occurring. This accident was not reported to the FDA until after the accidents in 1986.

The second accident occurred in July 1985 in Hamilton, Ontario. After the Therac-25 was activated, it shut down after five minutes and showed an error that said “no dose”. The operator repeated the process four more times. The operators had become accustomed to frequent malfunctions that had no problematic consequences for the patient. While the operator thought no dosage was given, in reality several doses were applied and the patient was hospitalized three days later for radiation overexposure. The FDA and the Canadian Radiation Protection Board were notified and the AECL issued a voluntary recall, while the FDA audited the modifications made to it. AECL redesigned a switch they believed caused the failure and announced that it was 10,000 times safer after the redesign.

The third accident occurred in December 1985 in Yakima, Washington where the Therac-25 had already been redesigned in response to the previous accident. After several treatments, the woman’s skin began to redden. The hospital called AECL and they said that “after careful consideration, we are of the opinion that this damage could not have been produced by any malfunction of the Therac-25 or by any operator error [3].” However, upon investigation the Yakima staff found evidence for radiation overexposure due to her symptoms of a chronic skin ulcer and dead tissue.

The fourth accident occurred in March 1986 in Tyler, Texas where a man died due to complications from the radiation overdose. In this, the message “Malfuction 54” kept appearing, indicating only 6 rads were given, so the operator proceeded with the treatment. That day the video monitor happened to be unplugged and the monitor was broken, so the operator had no way of knowing what was happening

inside, since the operator was operating the controls in another room. After the first burn, while he was trying to get up off the table, he received another dose – in the wrong location since he was moving. He then pounded on the door to get the operator's attention. Engineers were called upon, but they couldn't reproduce the problem so it was put back into use in April. The man's condition included vocal cord paralysis, paralysis of his left arm and both legs, and a lesion on his left lung, which eventually caused his death.

The fifth accident occurred in April 1986 at the same location as the previous one and produced the same "Malfunction 54" error. The same technician who treated the patient in the fourth accident prepared this patient for treatment. This technician was very experienced at this procedure and was a very fast typist. So, as with the former patient, when she typed something incorrectly, she quickly corrected the error. The same "Malfunction 54" error showed up and she knew there was trouble. She immediately contacted the hospital's physicist, and he took the machine out of service. After much perseverance on the parts of the physicist and the technician, they determined that the malfunction occurred only if the Therac-25 operator rapidly corrected a mistake. AECL filed a report with the FDA and began work on fixing the software bug. The FDA also required AECL to change the machine to clarify the meaning of malfunction error messages and shutdown the treatment after a large radiation pulse. Over the next three weeks, however, the patient fell into a coma, suffered neurological damage and died.

The sixth and final accident occurred in January 1987, again in Yakima, Washington. AECL engineers estimated that the patient received between 8,000 and 10,000 rads instead of the prescribed 86 rads after the system shut down and the operator continued with the treatment. The patient died due to complications from radiation overdose.

Contributing factors. Numerous factors were responsible for the failure of users and AECL to discover and correct the problem, and for the ultimate failure of the system. This is partly what makes this case so interesting; there were problems that spanned a wide range of causes.

Previous models of the machine were mostly hardware based. Before this, computer control was not widely in use and hardware mechanisms were in place in order to prevent catastrophic failures from occurring. With the Therac-25, the hardware controls and interlocks which had previously been used to prevent failure were removed. In the Therac-25, software control was almost solely responsible for mitigating errors and ensuring safe operation. Moreover, the same pieces of code which had controlled the earlier versions were modified and adapted to control the Therac-25. The controlling software was modified to incorporate safety mechanisms, presumably to replace more expensive hardware controls that were still in the Therac-20. To this day, not much is known about the sole programmer who ultimately created the software, other than he had minimal formal training in writing software.

Another factor was AECL's inability or unwillingness to resolve the problems when they occurred, even in the most serious patient death instances. It was a common practice for their engineering and other departments to dismiss claims of machine malfunction as user error, medical problems with the patient beyond AECL's control, and other circumstances wherein the blame would not fall on AECL. This caused users to try and locate other problems which could have caused the unexplained accidents.

Next, AECL's software quality practices were terrible, as demonstrated by their numerous Corrective Action Plan (CAP) submissions. When the FDA finally was made aware of the problem, they demanded of AECL that the numerous problems be fixed. Whenever AECL tried to provide a solution for a software problem, it either failed to fix the ultimate problem, or changed something very simple in the code, which ultimately could introduce other problems. They could not provide adequate testing plans, nor barely even provide any documentation to support the software they created [3]. For instance, the 64 "Malfunction"

codes were referenced only by their number, with no meaningful description of the error provided to the console.

FDA interaction at first was poor mainly due to the limited reporting requirements imposed on users. While medical accelerator manufacturers were required to report known issues or defects with their products, users were not, resulting in the governmental agency failing to get involved at the most crucial point following the first accident. Had users been required to report suspected malfunctions, the failures may well have been prevented. The AECL did not deem the accidents to be any fault on their part, and thus did not notify the FDA.

Finally, the defects in the software code were what ultimately attributed to the failures themselves. Poor user interface controls caused prescription and dose rate information to be entered improperly [3]. Other poor coding practices caused failures to materialize, such as the turntable to not be in the proper position when the beam was turned on. For one patient, this factor ultimately caused the radiation burns since the beam was applied in full force to the victim's body, without being first deflected and defused to emit a much lower dosage.

Another major problem was due to race conditions. They are brought on by shared variables when two threads or processes try to access or set a variable at the same time, without some sort of intervening synchronization. In the case of the Therac-25, if an operator entered all information regarding dosage to the console, arriving at the bottom of the screen, some of the software routines would automatically start even though the operator did not issue the command to accept those variables. If an operator then went back in the fields to fix an input error, it would not be sensed by the machine and would therefore not be used, using the erroneous value instead. This attributed to abnormally high dosage rates due to software malfunction.

An overflow occurs when a variable reaches the maximum value its memory space can store. For the Therac-25, a one byte variable named Class3 was used and incremented during the software checking phase. Since a one byte variable can only hold 255 values in total, on every 256th pass through, the value would revert to zero. A function checked this variable and if it was set to 0, the function would not check for a collimator error condition, a very serious problem. Thus, there was a 1 in 256 chance every pass through the program that the collimator would not be checked, resulting in a wider electron beam which caused the severe radiation burns experienced by a victim at one of the treatment sites [3].

Lessons Learned. The Therac-25 accidents were tragic and provide a learning tool to prevent any future disasters of this magnitude. All of the human, technical, and organizational factors must be considered when looking to find the cause of a problem. The main contributing factors of the Therac-25 accident included:

- "management inadequacies and lack of procedures for following through on all reported incidents
- overconfidence in the software and removal of hardware interlocks
- presumably less-than-acceptable software-engineering practices
- unrealistic risk assessments and overconfidence in the results of these assessments" [3].

When an accident arises, a thorough investigation must be conducted to see what sparked it. We cannot assume that the problems were caused by one aspect alone because parts of a system are interrelated.

Another lesson is that companies should have audit trails and incident-analysis procedures that are applied whenever it appears that a problem may be surfacing. Hazard logging and problems should be recorded as a part of quality control. A company also should not over rely on the numerical outputs of the safety analyses. Management must remain skeptical when making decisions.

The Therac-25 accidents also reemphasize the basics of software engineering which include complete documentation, established software quality assurance practices and standards, clean designs, and extensive testing and formal analysis at the module and software level. Changes in any design changes must also be documented so people do not reverse them in the future.

Manufacturers must not assume that reusing software is 100% safe; parts of the code in the Therac-20 that were re-used were found later to have had defects the entire time. But since there were still hardware safety controls and interlocks in place on that model, the defects went undetected. The software is a part of the whole system and cannot be tested on its own. Regression tests and system tests must be conducted to ensure safety. Designers also need to take time in designing user interfaces with safety in consideration.

According to Leveson [3], "Most accidents involving complex technology are caused by a combination of organizational, managerial, technical, and, sometimes, sociological or political factors. Preventing accidents requires paying attention to *all* the root causes, not just the precipitating event in a particular circumstance. Fixing each individual software flaw as it was found did not solve the device's safety problems. Virtually all complex software will behave in an unexpected or undesired fashion under some conditions -- there will always be another bug. Instead, accidents must be understood with respect to the complex factors involved. In addition, changes need to be made to eliminate or reduce the underlying causes and contributing factors that increase the likelihood of accidents or loss resulting from them."

Denver International Airport Baggage Handling System

Denver International Airport was to open a new airport in 1995, to replace the aging Stapleton International Airport. It was to have many new features that pilots and travelers alike would praise, in particular, a fully automated baggage system. At $193 million dollars, it was the new airports' crowning jewel [1]. It was to drastically lower operating costs while at the same time improve baggage transportation speeds and lower the amount of lost luggage.

Created and installed by BAE Automated Systems, this technological marvel would mean the airport needed no baggage handlers. It operated on a system of sensors, scales, and scanners all operated by a massive mainframe. Theft of valuables, lost luggage, and basically all human error would cease to be a problem in traveling through Denver. The airport and its main carrier, United Airlines, would no longer have the expenses of paying luggage handlers.

Unfortunately, the system didn't work like it was supposed to. Due to delays in getting the 22 mile long system to work, BAE delayed the opening of the airport by 16 months with an average cost of $1.1 million per day. Baggage was falling off the tracks and getting torn apart by automated handlers. Eventually a conventional system had to be installed at a cost of $70 million when it became clear the original system wasn't going to work. This cost, plus the delay in opening the airport, and some additional costs in installing the system added up to a staggering $640 million for what amounted to a regular baggage system [1].

This system has become a poster child for the poor project management that has seemed to pervade most major IT projects. A delay that was almost as long as the original project time and a budget that more than doubled the target all without a working end system.

The Plan. The baggage system of Denver International Airport was to be the pinnacle of modern airport design. It was designed to be a fully automated system that took care of baggage from offloading to retrieval by the owner at baggage claim. The system would be able to track baggage coming in, baggage being transferred, and a baggage's final destination at baggage claim. It was to handle the baggage of all

the airlines. The speed, efficiency, and reliability of the new system were also touted to be the most advanced baggage handling system in the world. Although the system was to be the world's most complex baggage technology, it was not the first of its kind. San Francisco International Airport, Rhein-Main International Airport, and Franz Joseph Strauss Airport all have similar systems but on a much smaller scale and of less complexity [3].

The high speed of the baggage system is one of the main areas of improvement. The system utilizes Destination-Coded Vehicles (DCV) or telecars capable of rolling on 22 miles of underground tracks at 24mph or 3 times as fast as the current conveyor belts in use. The plan was for a piece of baggage to be able to move between any concourse in the airport within 9 minutes. This would shave off minutes in turnaround time between each arriving and departing flight [3].

The system was integrated with several high-tech components. "It calls for 300 486-class computers distributed in eight control rooms, a Raima Corp. database running on a Netframe Systems fault-tolerant NF250 server, a high-speed fiber-optic ethernet network, 14 million feet of wiring, 56 laser arrays, 400 frequency readers, 22 miles of track, 6 miles of conveyor belts, 3,100 standard telecars, 450 oversized telecars, 10,000 motors, and 92 PLCs to control motors and track switches" [3]. With a system of this size and complexity, thorough planning and testing needed to take place to ensure reliable performance, once implemented.

Baggage Handling. The plan for the handling of baggage was also quite ingenious. At check-in, barcodes and identifiable photocells were attached to all baggage before being sent through the system. The barcodes and photocells contained information such as the bag's owner, flight number, final destination, etc. Once the baggage was sent on its way, the allocation system was to find an unused telecar for transportation. The telecar was to make its way towards the conveyor belt and engage the loading position. The baggage was to be quickly loaded into the moving telecar which was to assume the upright transport position, and then make its way to its destination. The photocell was to be read via photo-electric sensors and radio transmitters, with the tracking system guiding the telecar to its destination. The telecars were to move via linear induction motors along the tracks which would propel the car forward as the car passed over the motors. The tracking computers were also responsible for re-routing baggage and maintaining efficient transportation between the 3,550 telecars [3]. The system was to be a marvel to behold. Unfortunately, due to inadequate design and software testing procedures, the system did not work as planned. The final product was full of system glitches and baggage management errors.

Problems. The Denver International Airport's automated baggage system encountered so many problems that it delayed the opening of the airport for 16 months [4]. The scope of the project was far too complicated. It was too large of a technological leap from the manual sorting of luggage done in the past. With the system plagued with software glitches, carts being misrouted, and baggage being lost or damaged, the Denver airport baggage system was a complete failure.

Although the vast size and complication of the project was huge, BAE had to deal with constructing a baggage system while the airport was already being constructed. This led to designs that were not necessarily recommended, but were given no other option. BAE vice president stated that this was a 3-4 year project squeezed into 2 years. Adding to the time constraint and increasing costs, Denver officials often made changes to its timetable without consulting BAE, often causing delays. Airline companies such as United, would make changes or requests concerning where and how they wanted their baggage systems implemented, which also led to delays [5]. The project manager of the new airport made a poor decision concerning time management. He/she seemed to put the baggage system at the bottom of importance and did not realize the full size of the project.

When testing began on March of 1994, it was evident that there was still a lot of work to be done. The baggage system destroyed luggage and threw suitcases off of the carts. Carts jumped off tracks, crashed into each other, mysteriously disappeared, became jammed, and routed luggage to the wrong areas. Making the testing phase even more difficult was the lack of communication. Engineers inside the tunnels and around the airport dealt with dead spots which led to more frustration. BAE officials blamed software glitches and mechanical failures, beginning months of frustration [5].

The software that ran the system could not handle the airports 3,550 carts along 22 miles of track. The faulty software led carts to wrong areas, being sent too early or too late, and crashed into one another. The BAE president stated that writing code for the whole system was extremely complicated. They had to integrate the system with many of the airlines own software. For example, United uses a system called United Apollo, a reservation system that BAE had to integrate with its own software. This required BAE to translate their code so that the same language is used throughout the system which is painstaking and time consuming. While the code was being written, it grew more complicated and less manageable [5]. When code grows to an enormous size, it becomes harder to locate errors and to make corrections.

The timing of the carts caused much of the damage and misplaced luggage. When agents generated on-line tickets too quickly, it caused the reservation system to upload "junk" data to the baggage system. It caused the carts to dump luggage to a manual sorting station instead of its proper station. The speeds of the carts had to be regulated to fix the problem. There were also "line-balancing" issues where bags were waiting to be put on carts. Carts that were needed ended up being sent elsewhere [6]. So the system was not running efficiently.

The equipment that was chosen also led to some problems. The PCs chosen were not capable of handling the complex system. It became overwhelmed with the task of tracking all the carts. This resulted in having to redesigning the system so that it could handle the load. All the different types of software working with each other on the same computer caused system crashes. A major part of the system required scanners to read the barcode on the luggage so that the computers could tell where to unload the cart. The problem with this was that the barcodes were hard to read. Barcodes that were dirty, blocked, or bent were not picked up, leading the carts to the manual sorting stage. Sometimes the scanners would get hit, misplacing it, which also led the carts to the manual area. Faulty latches caused the carts to dump luggage on the track and jamming carts [5]. So besides software, hardware also caused problems.

The Results. Although some of Denver's problems were solved, the baggage system has never worked the way it was planned. It was only able to work with baggage on departing flights and not with baggage from arriving flights [2]. The airport's only solution was to downsize the whole system. Only parts of the airport used the automated baggage system while the rest relied on manual sorting. In 2005, the baggage system was abandoned entirely and a conventional baggage handling system was implemented at Denver International Airport [6].

The revolutionary new baggage system was a complete failure. The goals of the project were grand and innovative but the scope of the project was simply too large for the time allocated. The designers also had too much faith in their system and therefore failed to locate all the defects within their system. The project itself cost the Denver International Airport $250 million in startup, $100 million for additional construction, and another $341 million in trying to get the system to work properly [6]. After a decade of trouble and controversy, Denver finally decided to cut their losses and retire the automated baggage system. Even though this was a major failure in the field of project management, it was a hard lesson learned by the IT community. It takes failure to reach success, and in this case, it proved to be one of the major debacles which will hopefully relay the importance of effective project management and thorough software testing to future IT project teams.

Mars Polar Lander

The Mars Polar Lander was the second portion of NASA's Mars Surveyor '98 Program. Launched on January 3, 1999 from Cape Canaveral, it was to work in coordination with the Mars Climate Orbiter to study "Martian weather, climate and soil in search of water and evidence of long-term climate changes and other interesting weather effects" [1].

The initial problems with the project started in September 1999 when the Climate Orbiter disintegrated while trying to fall into Martian orbit. It was later determined that this was due to a miscommunication between Jet Propulsion Labs and Lockheed Martin Astronautics over the use of metric (Newton) and English (pound) forces to measure the thruster firing strength [1]. Upon the discovery of the problem's source, members at JPL went back and corrected the issues and "spent weeks reviewing all navigation instructions sent to the lander to make sure it would not be afflicted by an error like the one that doomed the orbiter" [2].

The Problem. On December 3, 1999, the Polar Lander began its decent towards the Martin surface after a slight course adjustment for refinement purposes, and entered radio silence as it entered the Martian atmosphere [2]. Five minutes after the Polar Lander landed on the Martian surface, it was supposed to have begun to transmit radio signals to Earth, taking approximately 30 minutes to travel the 157 million mile distance between the planets [3]. Those signals never made it to Earth, and despite repeated attempts by the people at JPL through April 2000 [2], contact was never established with the Polar Lander, resulting in another thorough investigation to determine the errors that faulted this program.

The Causes and Lessons Learned. It was later determined that an error in one simple line of code resulted in the loss of the Polar Lander. The landing rockets "were supposed to continue firing until one of the craft's landing legs touched the surface. Apparently the onboard software mistook the jolt of landing-leg deployment for ground contact and shut down the engines, causing [the Polar Lander] to fall from a presumed height of 40 meters (130 feet)" [4] to its demise on the Martian surface. This theory has received more support just recently when the recent discovery of what may be the Polar Lander's wreckage on the surface; as images recently reanalyzed by scientists show what may be both the parachute and the remains of the Polar Lander. From this, scientists can conclude that the Polar Lander's "descent proceeded more or less successfully through atmospheric entry and parachute jettison. It was only a few short moments before touchdown that disaster struck" [4], and a $165 million investment was lost millions of miles away.

According to software safety experts at the Massachusetts Institute of Technology (MIT), even though the cause of Mars Polar Lander's destruction was based on an error that was traced to a single bad line of software code, "'that trouble spot is just a symptom of a much larger problem -- software systems are getting so complex, they are becoming unmanageable'" [5]. Feature creep, where features are added, yet not tested rigorously due to budget restraints, compounds the problems. As the Polar Lander software developed more and more add-ons and features while being tested less and less, it became doomed before it was even loaded onto the Polar Lander itself and launched towards Mars [5].

Amidst the coding and human errors, there were also several engineering design and testing flaws discovered in the Polar Lander in post-wreck reports. There were reports that NASA altered tests, but these allegations have not been proven. One dealt with unstable fuel used and the other with improper testing of the landing legs [6].

One review, led by a former Lockheed Martin executive, found the Mars exploration program to be lacking in experienced managers, a test and verification program and adequate safety margins. NASA's Mars Polar Lander Failure Review Board, chaired by a former Jet Propulsion Laboratory flight operations

chief, also released its report. The report said the Polar Lander probably failed due to a premature shutdown of its descent engine, causing the $165 million spacecraft to smash into the surface of Mars. It concluded more training, more management and better oversight could have caught the problem [7].

The Patriot Missile

The Patriot is an army surface-to-air, mobile, air defense system [1]. Originating in the mid-1960's, it was designed to operate in Europe against Soviet medium-to-high-altitude aircraft and cruise missiles traveling at speeds up to about MACH 2. It was then modified in the 1980's to serve as a defense against incoming short range ballistic missiles, such as the SCUD missile used by Iraq during the first gulf war. It had several upgrades since then, in 2002 when the missile itself was upgraded to include onboard radar. The system was designed to be mobile and operate for only a few hours at a time. This was in order to avoid detection. It got its first real combat test in the first gulf war and was deployed by US forces during Operation Iraqi Freedom.

Specifications and Operations. The missile is 7.4 feet long and is powered by a single stage solid propellant rocket motor that runs at mach 3 speeds. The missile itself weighs 2200 pounds and its range is 43 miles. The Patriot is armed with a 200 pound high-explosive warhead detonated by a proximity fuse that causes shrapnel to destroy the intended target [3]. The plan is for the patriot missile to fly straight toward the incoming missile and then explode at the point of nearest approach. The explosion will either destroy the incoming missile with shrapnel, or knock the incoming missile off course so it misses its target.

The Engagement Control Station (ECS) is the only manned station in a patriot battery. The ECS communicates with the launcher, with other Patriot batteries, and with higher command headquarters. It controls all the launchers in the battery. The ECS is manned by three operators, who have two consoles and a communications station with three radio relay terminals [6]. So, the weapon control computer is linked directly to the launchers as well as the radar.

The phased array radar carries out search, target detection, track and identification, missile tracking and guidance and electronic counter-countermeasures (ECCM) functions. The radar is mounted on a trailer and is automatically controlled by the digital weapons control computer in the Engagement Control Station, via a cable link. The radar system has a range of up to 100km, capacity to track up to 100 targets and can provide missile guidance data for up to nine missiles. The radar antenna has a 63 mile (100 kilometer) range [6].

The system has 3 modes: automatic, semi-automatic, and manual. It relies heavily on its automatic mode. An incoming missile is flying at approximately one mile every second (Mach 5) and could be 50 miles (80.5 kilometers) away when the Patriot's radar locks onto it. Automatic detection and launching becomes a crucial feature, because there is not a lot of time to react and respond once the missile is detected and a human being could not possibly see it or identify it at that distance. The system therefore depends on its radar and the weapon control computer.

The process of finding a target, launching a missile, and destroying a target is straightforward. First, the Patriot missile system uses its ground-based radar to find, identify and track the targets. Once it finds a target, it scans it more intensely and communicates with the ECS. When the operator or computer decides that it has an incoming foe, the ECS calculates an initial heading for the Patriot missile. It chooses the Patriot missile it will launch, downloads the initial guidance information to that missile, and launches it. Within three seconds the missile is traveling at Mach 5 and is headed in the general direction of the target. After launch, the Patriot missile is acquired by the radar. Then the Patriot's computer guides the

missile toward the incoming target [7]. As briefly mentioned earlier, when it gets close to the target its proximity fuse detonates the high explosive warhead and the target is either destroyed by the shrapnel or knocked off course. After firing its missiles, a re-supply truck with a crane pulls up next to the launcher to load it with new missiles.

The newer PAC-3 missile, first used in 2002, contained its own radar transmitter and computer, allowing it to guide itself. Once launched, it turns on its radar, finds the target and aims for a direct hit. Unlike the PAC-2, which explodes and relies on the shrapnel to destroy the target or knock it off course, these PAC-3 missiles are designed to actually hit the incoming target and explode so that the incoming missile is completely destroyed. This feature makes it more effective against chemical and biological warheads because they are destroyed well away from the target. Described as a bullet hitting a bullet these “bullets” close in on each other at speeds up to Mach 10. At that speed there is no room for error -- if the missile miscalculates by even 1/100th of a second, it will be off by more than 100 feet (30.5 meters) [7] .

Disaster in Dhahran. On February 25, 1991 a Patriot missile defense system operating at Dhahran, Saudi Arabia, during Operation Desert Storm failed to track and intercept an incoming Scud. In fact, no Patriot missile was launched to intercept the Scud that day where it subsequently hit an Army barracks killing twenty-eight people and injuring ninety-seven [1].

The Patriot problems likely stemmed from one fundamental aspect of its design: the Patriot was originally designed as an anti-aircraft, not anti-missile, defense system. With this limited purpose in mind, Raytheon designed the system with certain constraints. One such constraint was that the designers did not expect the Patriot system to operate for more than a few hours at a time--it was expected to be used in a mobile unit rather than at a fixed location. At the time of the Scud attack on Dhahran, the Patriot battery had been running continuously for four days--more than 100 hours. They calculated that after only 8 hours of continuous operation, the Patriot's stored clock value would be off by 0.0275 seconds, causing an error in range gate calculation of approximately 55 meters. At the time of the Dhahran attack, the Patriot battery in that area had been operating continuously for more than 100 hours--its stored clock value was 0.3433 seconds off, causing the range gate to be shifted 687 meters, a large enough distance that the Patriot was looking for the target in the wrong place. Consequently, the target did not appear where the Patriot incorrectly calculated it should. Therefore the Patriot classified the incoming Scud as a false alarm and ignored it--with disastrous results [2].

On February 11, 1991, after determining the effect of the error over time, the Israelis notified the U.S. Patriot project office of the problem. Once they were notified, the programming team set to work solving the problem. Within a few days, the Patriot project office made a software fix correcting the timing error, and sent it out to the troops on February 16, 1991. Sadly, at the time of the Dhahran attack, the software update had yet to arrive in Dhahran. That update, which arrived in Dhahran the day after the attack, might have saved the lives of those in the barracks [2].

This problem of unsuccessfully intercepting incoming missiles was wide spread. The Patriot missile was designed in the late 1970's as an antiaircraft weapon. However, it was modified in the 1980's to serve as a defense against incoming short range ballistic missiles. Until the Gulf War the Patriot had not been tested in combat. Because of this, the Army determined that the Patriot succeeded in intercepting Scud missiles in only perhaps 10-24 of more than 80 attempts. Determining the true success rate of the Patriot is difficult. First, "success" is defined in several ways: destruction, damage to, and deflection of a Scud missile may variously be interpreted as successes depending on who makes the assessment. Secondly, the criteria used for "proof" of a "kill" varied--in some cases Army soldiers made little or no investigation and assumed a kill, in other cases they observed hard evidence of a Scud's destruction [2].

Lessons Learned. A 10 month investigation by the House Government Operations subcommittee on Legislation and National Security concluded that there was little evidence to prove that the Patriot hit more than a few Scuds. Testimony before the House Committee on Government Operations raised serious doubts about the Patriot's performance [3]. One significant lesson learned from this disaster is that robust testing is needed for safety-critical software. We need to test the product for the environment it will be used in, and under varying conditions. When redesigning systems for a new use, we need to be very careful to ensure the new design is safe and effective. Also, clear communication among the designers, developers, and operators is needed to ensure safe operation. When software needs to be fixed, it must be done quickly and deployed to the site immediately.

Introduction and Therac-25 Medical Accelerator Sources

Many thanks to Irene Songco, Josh Kretchman, and Marvel Tay.

[1] http://www.quotationspage.com/quote/27300.html The Life of Reason, Volume 1, 1905

[2] http://www.byte.com/art/9512/sec6/art1.htm

[3] Leveson, Turner, "An Investigation of the Therac-25 Accidents", IEEE Computer, Vol. 26, No. 7, July 1993, pp.18-41.

[4] http://www.computingcases.org/case_materials/therac/case_history/Case%20History.html

[5] http://wired.com/news/technology/bugs/0,2924,69355,00.html?tw=wn_tophead_1

Denver Airport Baggage System Sources

Many thanks to Timothy Aing, Nathan Grassman, Scott Kwan, and Andrew Stroud

[1] http://www.cis.gsu.edu/~mmoore/CIS3300/handouts/SciAmSept1994.html

[2] http://www.computerworld.com/managementtopics/management/project/story/0,10801,102436,00.html

[3] http://www.csc.calpoly.edu/~dstearns/SchlohProject/function.html

[4] http://ardent.mit.edu/airports/ASP_papers/Bag%20System%20at%20Denver.PDF

[5] http://www.csc.calpoly.edu/~dstearns/SchlohProject/problems.html

[6] http://www.msnbc.msn.com/id/8135924/

Mars Polar Lander Sources

Many thanks to Martin Chia, Ryan Feist, and Jeff Rose.

[1] http://www.spacetoday.org/SolSys/Mars/MarsExploration/MarsSurveyor98.html

[2] http://partners.nytimes.com/library/national/science/120499sci-nasa-mars.html

[3] http://www.astronautix.com/craft/marander.htm

[4] http://skyandtelescope.com/news/article_1509_1.asp

[5] http://www.space.com/businesstechnology/technology/mpl_software_crash_000331.html

[6] http://www.space.com/cgi-bin/email/gate.cgi?lk=T1&date=000328&go=/science/solarsystem/mpl_report_000328.html

[7] http://www.space.com/scienceastronomy/solarsystem/nasa_report_synopsis_000328.html

Patriot Missile Disaster Sources

Many thanks to Erin Bivens, Bryan Rasmussen, and Michael VonFrausing-Borch.

[1] http://www.fas.org/spp/starwars/gao/im92026.htm

[2] http://shelley.toich.net/projects/CS201/patriot.html

[3] http://www.cdi.org/issues/bmd/Patriot.html

[4] http://klabs.org/richcontent/Reports/Failure_Reports/patriot/patriot_gao_145960.pdf

[5] http://cs.furman.edu/digitaldomain/themes/risks/risks_numeric.htm

[6] http://www.army-technology.com/projects/patriot/

[7] http://science.howstuffworks.com/patriot-missile1.htm

[8] http://www.cbsnews.com/stories/2004/02/19/60minutes/main601241.shtml

[9] http://www.pcworld.com/news/article/0,aid,110035,00.asp