**Handouts of Lecture 25 Professional Practices (IT)**

**Lecture Title: Computer Reliability (Cont.)**

**Analysis: E-retailer Posts Wrong Price, Refuses to Deliver**

Amazon.com shut down its British Web site on March 13, 2003, after a software error led it to offer iPAQ handheld computers for 7 pounds instead of the correct price of about 275 pounds. Before Amazon.com shut down the site, electronic bargain hunters had flocked to Amazon.com’s Web site, some of them ordering as many as 10 iPAQs.

Amazon said that customers who ordered at the mistaken price should not expect delivery unless they paid the difference between the advertised price and the actual price. An Amazon.com spokesperson said, “In our Pricing and Availability Policy, we state that where an item’s correct price is higher than our stated price, we contact the customer before dispatching. Customers will be offered the opportunity either to cancel their order or to place new orders for the item at the correct price”. Was Amazon.com wrong to refuse to fill the orders of the people who bought iPAQs for 7 pounds?

Let’s analyze the problem from a rule utilitarian point of view. We can imagine a moral rule of the form, “A person or organization wishing to sell a product must always honor the advertised price.” What would happen if this rule were universally followed? More time and effort would be spent proofreading advertisements, whether printed or electronic. Organizations responsible for publishing the advertisements in newspapers, magazines, and Web sites would also take more care to ensure no errors were introduced. There is a good chance companies would take out insurance policies to guard against the catastrophic losses that could result from a typo. To pay for these additional costs, the prices of the products sold by these companies would be higher. The proposed rule would harm every consumer who ended up paying more for products. The rule would benefit the few consumers who took advantage of misprints to get good deals on certain goods. We conclude the proposed moral rule has more harms than benefits, and Amazon.com did the right thing by refusing to ship the iPAQs.

We could argue, from a Kantian point of view, that the knowledgeable consumers who ordered the iPAQs did something wrong. The correct price was 275 pounds; the advertised price was 7 pounds. While electronic products may go on sale, retailers simply do not drop the price of their goods by 97.5 percent, even when they are being put on clearance. If consumers understood the advertised price was an error, then they were taking advantage of Amazon.com’s stockholders by ordering the iPAQ before the error was corrected. They were not acting “in good faith.”

**Notable Software System Failures**

An embedded system is a computer used as a component of a larger system. You can find microprocessor based embedded systems in microwave ovens, thermostats, automobiles, traffic lights, and a myriad of other modern devices. Because computers need software to execute, every embedded system has a software component. Software is playing an ever-larger role in system functionality. There are several reasons why hardware controllers are being replaced by microprocessors controlled by software. Software controllers are faster. They can perform more sophisticated functions, taking more input data into account. They cost less, use less energy, and do not wear out. Unfortunately, while hardware controllers have a reputation for high reliability, the same cannot be said for their software replacements.

Most embedded systems are also real-time systems: computers that process data from sensors as events occur. The microprocessor that controls the air bags in a modern automobile is a real-time system, because it must instantly react to readings from its sensors and deploy the air bags at the time of a collision. The microprocessor in a cell phone is another example of a real-time system that converts electrical signals into radio waves and vice versa.

**Patriot Missile**

The Patriot missile system was originally designed by the US Army to shoot down airplanes. In the 1991 Gulf War, the Army put the Patriot missile system to work defending against Scud missiles launched at Israel and Saudi Arabia. At the end of the Gulf War, the Army claimed the Patriot missile defense system had been 95 percent effective at destroying incoming Scud missiles.

Later analyses showed that perhaps as few as 9 percent of the Scuds were actually destroyed by Patriot missiles. As it turns out, many Scuds simply fell apart as they approached their targets—their destruction had nothing at all to do with the Patriot missiles launched at them. The most significant failure of the Patriot missile system occurred during the night of February 25, 1991, when a Scud missile fired from Iraq hit a US Army barracks in Dhahran, Saudi Arabia, killing 28 soldiers.

The Patriot missile battery defending the area never even fired at the incoming Scud. Mississippi congressman Howard Wolpe asked the General Accounting Office (GAO) to investigate this incident. The GAO report traced the failure of the Patriot system to a software error. The missile battery did detect the incoming Scud missile as it came over the horizon. However, in order to prevent the system from responding to false alarms, the computer was programmed to check multiple times for the presence of the missile. The computer predicted the flight path of the incoming missile, directed the radar to focus in on that area, and scanned a segment of the radar signal, called a range gate, for the target. In this case the program scanned the wrong range gate. Since it did not detect the Scud, it did not fire the Patriot missile

Why did the program scan the wrong range gate? The tracking system relied upon getting signals from the system clock. These values were stored in a floating-point variable with insufficient precision, resulting in a small mathematical error called a truncation. The longer the system ran, the more these truncation errors added up. The Patriot missile system was designed to operate for only a few hours at a time. However, the system at Dhahran had been in continuous operation for 100 hours. The accumulation of errors led to a difference between the actual time and the computed time of about 0.3433 seconds. Because missiles travel at high speeds, the 0.3433-second error led to a tracking error of 687 meters (about half a mile). That was enough of an error to prevent the missile battery from locating the Scud in the range gate area.

**Ariane 5**

The Ariane 5 was a satellite launch vehicle designed by the French space agency, the Centre National d’Etudes Spatiales, and the European Space Agency. About 40 seconds into its maiden flight on June 4, 1996, a software error caused the nozzles on the solid boosters and the main rocket engine to swivel to extreme positions. As a result, the rocket veered sharply off course. When the links between the solid boosters and the core stage ruptured, the launch vehicle self-destructed. The rocket carried satellites worth $500 million, which were not insured.

A board of inquiry traced the software error to a piece of code that converts a 64-bit floating-point value into a 16-bit signed integer. The value to be converted exceeded the maximum value that could be stored in the integer variable, causing an exception to be raised. Unfortunately, there was no exception-handling mechanism for this particular exception, so the onboard computers crashed. The faulty piece of code had been part of the software for the Ariane 4. The 64- bit floating-point value represented the horizontal bias of the launch vehicle, which is related to its horizontal velocity. When the software module was designed, engineers determined that it would be impossible for the horizontal bias to be so large that it could not be stored in a 16-bit signed integer.

There was no need for an error handler, because an error could not occur. This code was moved “as is” into the software for the Ariane 5. That proved to be an extremely costly mistake, because the Ariane 5 was faster than the Ariane 4. The original assumptions made by the designers of the software no longer held true.

**AT&T Long-Distance Network**

On the afternoon of January 15, 1990, AT&T’s long-distance network suffered a significant disruption of service. About half of the computerized telephone-routing switches crashed, and the remainder of the switches could not handle all of the traffic. As a result of this failure, about 70 million long-distance telephone calls could not be put through, and about 60,000 people lost all telephone service. AT&T lost tens of millions of dollars in revenue. It also lost some of its credibility as a reliable provider of long-distance service.

Investigation by AT&T engineers revealed that the network crash was brought about by a single faulty line of code in an error recovery procedure. The system was designed so that if a server discovered it was in an error state, it would reboot itself, a crude but effective way of “wiping the slate clean.” After a switch rebooted itself, it would send an “OK” message to other switches, letting them know it was back online. The software bug manifested itself when a very busy switch received an “OK” message. Under certain circumstances, handling the “OK” message would cause the busy switch to enter an error state and reboot.

On the afternoon of January 15, 1990, a System 7 switch in New York City detected an error condition and rebooted itself. When it came back online, it broadcast an “OK” message. All the switches receiving the “OK” messages handled them correctly, except three very busy switches in St. Louis, Detroit, and Atlanta. These switches detected an error condition and rebooted. When they came back up, all of them broadcast “OK” messages across the network, causing other switches to fail in an ever expanding wave.

Every switch failure compounded the problem in two ways. When the switch went down, it pushed more long-distance traffic onto the other switches, making them busier. When the switch came back up, it broadcast “OK” messages to these busier switches, causing some of them to fail. Some switches rebooted repeatedly under the barrage of “OK” messages. Within 10 minutes, half the switches in the AT&T network had failed.

The crash could have been much worse, but AT&T had converted only 80 of its network switches to the System 7 software. It had left System 6 software running on 34 of the switches “just in case.” The System 6 switches did not have the software bug and did not crash.

**Robot Missions to Mars**

NASA designed the $125-million Mars Climate Orbiter to facilitate communications between Earth and automated probes on the surface of Mars, including the Mars Polar Lander. Ironically, the spacecraft was lost because of a miscommunication between two support teams on Earth.

The Lockheed Martin flight operations team in Colorado designed its software to use English units. Its program output thrust in terms of foot-pounds. The navigation team at the Jet Propulsion Laboratory in California designed its software to use metric units. Its program expected thrust to be input in terms of newtons. One foot-pound equals 4.45 newtons.

On September 23, 1999, the Mars Climate Orbiter neared the Red Planet. When it was time for the spacecraft to fire its engine to enter orbit, the Colorado team supplied thrust information to the California team, which relayed it to the spacecraft. Because of the units mismatch, the navigation team specified 4.45 times too much thrust. The spacecraft flew too close to the surface of Mars and burned up in its atmosphere. A few months later NASA’s Martian program suffered a second catastrophe. The Mars Polar Lander, produced at a cost of $165 million, was supposed to land on the south pole of Mars and provide data that would help scientists understand how the Martian climate has changed over time.

On December 3, 1999, NASA lost contact with the Mars Polar Lander. NASA engineers suspect that the system’s software got a false signal from the landing gear and shut down the engines 100 feet above the planet’s surface.

Tony Spear was project manager of the Mars Pathfinder mission. He said, “It is just as hard to do Mars missions now as it was in the mid-70s. I’m a big believer that software hasn’t gone anywhere. Software is the number-one problem”.

Several years after Spear made this observation, NASA successfully landed two Mars Exploration Rovers on the Red Planet. The rovers, named Opportunity and Spirit, were launched from Earth in June and July of 2003, successfully landing on Mars in January 2004. Mission planners had hoped that each rover would complete a three month mission, looking for clues that the Martian surface once had enough water to sustain life. The rovers greatly exceeded this goal. The Spirit rover operated successfully for more than five years. Opportunity found evidence of a former saltwater lake and was still operational ten years after its launch.

**Denver International Airport**

As airline passenger traffic strained the capacity of Stapleton International Airport, the City and County of Denver planned the construction of a much larger airport. Stapleton International Airport had earned a reputation for slow baggage handling, and the project planners wanted to ensure the new airport would not suffer from the same problem. They announced an ambitious plan to create a one-of-a-kind, state-of-the-art automated baggage handling system for the Denver International Airport (DIA).

The airport authorities signed a $193 million contract with BAE Automated Systems to design and build the automated baggage-handling system, which consisted of thousands of baggage carts traveling roller-coaster-style on 21 miles of metal tracks. According to the design, agents would label a piece of luggage and put it on a conveyor belt. Computers would route each bag along one or more belts until they reached a cart-loading point, where each bag would be loaded into its own tub-like cart. Scanners would read the destination information from the suitcase label, and computers would then route each cart along the tracks at 20 miles per hour to the correct unloading point, where each bag would be unloaded onto a conveyor belt and carried to its final destination. To monitor the movement of the bags, the system used 56 bar code scanners and 5,000 electric eyes.

There were problems from the outset of the project. The airport design was already completed before the baggage-handling system was chosen. As a result, the underground tunnels were small and had sharp turns, making it difficult to shoehorn in an automated baggage system. And given its ambitious goals, the project timeline was too short. However, the most important problem with the automated baggage handler was that the complexity of the system exceeded the ability of the development team to understand it.

Here are a few of the problems BAE encountered:

Luggage carts were misrouted and failed to arrive at their destinations.

Computers lost track of where the carts were.

Bar code printers didn’t print tags clearly enough to be read by scanners.

Luggage had to be correctly positioned on conveyors in order to load properly.

Bumpers on the carts interfered with the electric photocells.

Workers painted over electric eyes or knocked photo sensors out of alignment.

Light luggage was thrown off rapidly moving carts.

Luggage was shredded by automated baggage handlers.

The design did not consider the problem of fairly balancing the number of available carts among all the locations needing them. BAE attempted to solve these problems one at a time by trial and error, but the system was too complicated to yield to this problem-solving approach. BAE should have been looking at the big picture, trying to find where the specifications for the system were wrong or unattainable.

DIA was supposed to open on October 31, 1993. The opening was delayed repeatedly because the baggage-handling system was not yet operational. Eventually, the mayor of Denver announced the city would spend $50 million to build a conventional luggage-handling system using tugs and carts. (This conventional system actually ended up costing $71 million.) On February 28, 1995, flights to and from the new airport began. However, concourse A was not open at all. Concourse C opened with 11 airlines using a traditional baggage system.

The BAE automated system, far over budget at $311 million, was used only by United Airlines in concourse B to handle outgoing baggage originating in Denver. United used a traditional system for the rest of its baggage in concourse B. The failure of BAE to deliver a working system on time resulted in a 16-month delay in the opening of DIA. This delay cost Denver $1 million a day in interest on bonds and operating costs. As a result, DIA began charging all the airlines a flight fee of about $20 per passenger, the highest airport fee in the nation. Airlines passed along this cost to consumers by raising ticket prices of flights going through Denver.

***Reference***

***Lecture 25 slides: COMPUTER RELIABILITY***

***Gao, Y. (2012). Ethics for the Information Age by Michael J. Quinn. World Libraries, 20(1).***