**Handouts of Lecture 26 Professional Practices (IT)**

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

**Therac-25**

Soon after German physicist Wilhelm Roentgen discovered the X-ray in 1895, physicians began using radiation to treat cancer. Today, between 50 and 60 percent of cancer patients are treated with radiation, either to destroy cancer cells or relieve pain. Linear accelerators create high-energy electron beams to treat shallow tumors and X-ray beams to reach deeper tumors. The Therac-25 linear accelerator was notoriously unreliable. It was not unusual for the system to malfunction 40 times a day. We devote an entire section to telling the story of the Therac-25 because it is a striking example of the harm that can be caused when the safety of a system relies solely upon the quality of its embedded software. In a 20-month period between June 1985 and January 1987, the Therac-25 administered massive overdoses to six patients, causing the deaths of three of them. While 1987 may seem like the distant past to many of you, it does give us the advantage of 20/20 hindsight. The entire story has been thoroughly researched and documented. Failures of computerized systems continue to this day, but they have not yet been fully played out and analyzed.

**Genesis of the Therac-25**

Atomic Energy of Canada Limited (AECL) and the French corporation CGR cooperated in the 1970s to build two linear accelerators: the Therac-6 and the Therac-20. Both the Therac-6 and the Therac-20 were modernizations of older CGR linear accelerators. The distinguishing feature of the Therac series was the use of a DEC PDP 11 minicomputer as a “front end.” By adding the computer, the linear accelerators were easier to operate. The Therac-6 and the Therac-20 were actually capable of working independently of the PDP 11, and all of their safety features were built into the hardware. After producing the Therac-20, AECL and CGR went their separate ways. AECL moved ahead with the development and deployment of a next-generation linear accelerator called the Therac-25. Like the Therac-6 and the Therac-20, the Therac-25 made use of a PDP 11. Unlike its predecessor machines, however, AECL designed the PDP 11 to be an integral part of the device; the linear accelerator was incapable of operating without the computer. This design decision enabled AECL to reduce costs by replacing some of the hardware safety features of the Therac-20 with software safety features in the Therac-25.

AECL also decided to reuse some of the Therac-6 and Therac-20 software in the Therac-25. Code reuse saves time and money. Theoretically, “tried-and-true” software is more reliable than newly written code, but as we shall see, that assumption was invalid in this case. AECL shipped its first Therac-25 in 1983. In all, it delivered 11 systems in Canada and the United States. The Therac-25 was a large machine that was placed in its own room. Shielding in the walls, ceiling, and floor of the room prevented outsiders from being exposed to radiation. A television camera, microphone, and speaker in the room allowed the technician in an adjoining room to view and communicate with the patient undergoing treatment.

**Software Errors**

In the course of investigating the accidents, AECL discovered a variety of hardware and software problems with the Therac-25. Two of the software errors are examples of race conditions. In a race condition, two or more concurrent tasks share a variable, and the order in which they read or write the value of the variable can affect the behavior of the program. Race conditions are extremely difficult to identify and fix, because usually the two tasks do not interfere with each other and nothing goes wrong. Only in rare conditions will the tasks actually interfere with each other as they manipulate the variable, causing the error to occur. We describe both of these errors to give you some insight into how difficult they are to detect. The accidents at the ETCC occurred because of a race condition associated with the command screen.



One task was responsible for handling keyboard input and making changes to the command screen. A second task was responsible for monitoring the command screen for changes and moving the magnets into position. After the operator uses the first task to complete the prescription (1), the second task sees the cursor in the lower right-hand corner of the screen and begins the eight-second process of moving the magnets (2). Meanwhile, the operator sees her mistake. The first task responds to her keystrokes and lets her change the “X” to an “E” (3). She gets the cursor back to the lower right-hand corner before eight seconds are up (4). Now the second task finishes moving the magnets (5). It sees the cursor in the lower right-hand corner of the screen and incorrectly assumes the screen has not changed. The crucial substitution of electron beam for X-ray goes unnoticed. What makes this bug particularly treacherous is that it only occurs with faster, more experienced operators. Slower operators would not be able to complete the edit and get the cursor back to the lower right-hand corner of the screen in only eight seconds. If the cursor happened to be anywhere else on the screen when the magnets stopped moving, the software would check for a screen edit and there would be no overdose. It is ironic that the safety of the system actually decreased as the experience of the operator increased. Another race condition was responsible for the overdoses at the Yakima Valley Memorial Hospital.



It occurred when the machine was putting the electron beam gun back into position. A variable was supposed to be 0 if the gun was ready to fire. Any other value meant the gun was not ready. As long as the electron beam gun was out of position, one task kept incrementing that variable. Unfortunately, the variable could only store the values from 0 to 255. Incrementing it when it had the value 255 would result in the variable’s value rolling over to 0, like a car’s odometer. Nearly every time that the operator hit the SET button when the gun was out of position, the variable was not 0 and the gun did not fire (a). However, there was a very slight chance that the variable would have just rolled over when the operator hit the SET button (b). In this case the accelerator would emit a charge, even though the system was not ready

**Postmortem**

Let’s consider some of the mistakes AECL made in the design, development, and support of this system. When accidents were reported, AECL focused on identifying and fixing particular software bugs. This approach was too narrow. As Nancy Leveson and Clark Turner point out, “most accidents are system accidents; that is, they stem from complex interactions between various components and activities”. The entire system was broken, not just the software. A strategy of eliminating bugs assumes that at some point the last bug will be eradicated. But as Leveson and Turner write, “There is always another software bug”.

The real problem was that the system was not designed to be fail-safe. Good engineering practice dictates that a system should be designed so that no single point of failure leads to a catastrophe. By relying completely upon software for protection against overdoses, the Therac-25 designers ignored this fundamental engineering principle. Another flaw in the design of the Therac-25 was its lack of software or hardware devices to detect and report overdoses and shut down the accelerator immediately. Instead, the Therac-25 designers left it up to the patients to report when they had received overdoses.

There are also particular software lessons we can learn from the case of the Therac25. First, it is very difficult to find software errors in programs where multiple tasks execute at the same time and interact through shared variables. Second, the software design needs to be as simple as possible, and design decisions must be documented to aid in the maintenance of the system. Third, the code must be reasonably documented at the time it is written. Fourth, reusing code does not always increase the quality of the final product. AECL assumed that by reusing code from the Therac-6 and Therac-20, the software would be more reliable. After all, the code had been part of systems used by customers for years with no problems. This assumption turned out to be wrong. The earlier codes did contain errors, but these errors remained undetected because the earlier machines had hardware interlocks that prevented the computer’s erroneous commands from harming patients. The tragedy was compounded because AECL did not communicate fully with its customers. For example, AECL told the physicists in Washington and Texas that an overdose was impossible, even though AECL had already been sued by the patient in Georgia.

**Moral Responsibility of the Therac-25**

Team Should the developers and managers at AECL be held morally responsible for the deaths resulting from the use of the Therac-25 they produced? In order for a moral agent to be responsible for a harmful event, two conditions must hold:

Causal condition: the actions (or inactions) of the agent must have caused the harm.

 Mental condition: the actions (or inactions) must have been intended or willed by the agent.

In this case, the causal condition is easy to establish. The deaths resulted both from the action of AECL employees (creating the therapy machine that administered the overdose) and the inaction of AECL employees (failing to withdraw the machine from service or even inform other users of the machine that there had been overdoses). What about the second condition? Surely the engineers at AECL did not intend or try to create a machine that would administer lethal overdoses of radiation. However, philosophers also extend the mental condition to include unintended harm if the moral agent’s actions were the result of carelessness, recklessness, or negligence. The design team took a number of actions that fall into this category. It constructed a system without hardware interlocks to prevent overdoses or to keep the beam from being activated when the turntable was not in a correct position. The machine had no software or hardware devices to detect an accidental overdose. Management allowed software to be developed without adequate documentation. It presumed the correctness of reused code and failed to test it thoroughly. For these reasons the mental condition holds as well, and we conclude the Therac-25 team at AECL is morally responsible for the deaths caused by the Therac-25 radiation therapy machine.

**Postscript**

More than two decades after the Therac-25 accidents, computer errors related to radiation machines continue to maim and kill patients. In late 2006, Scott Jerome-Parks received three overdoses from a linear accelerator at a New York City Hospital that led to his death a few weeks later. He was only 43 years old. About the same time, 32-year-old breast cancer patient Alexandra Jn-Charles received 27 straight days of radiation overdoses at another New York hospital that led to her death. An investigation of radiation overdoses by the New York Times concluded that a variety of errors, including faulty software, were leading to crippling or fatal accidents.

**Computer Simulations**

Errors in computer simulations can result in poorly designed products, mediocre science, and bad policy decisions. In this section we review our growing reliance on computer simulations for designing products, understanding our world, and even predicting the future, and we describe ways in which computer modelers validate their simulations.

**Uses of Simulation**

Computer simulation plays a key role in contemporary science and engineering. There are many reasons why a scientist or engineer may not be able to perform a physical experiment. It may be too expensive or time consuming, or it may be unethical or impossible to perform. Computer simulations have been used to design nuclear weapons, search for oil, create pharmaceuticals, and design safer, more fuel-efficient cars. They have even been used to design consumer products such as disposable diapers. Some computer simulations model past events. For example, when astrophysicists derive theories about the evolution of the universe, they can test them through computer simulations. A computer simulation has demonstrated that a gas disk around a young star can fragment into giant gas planets such as Jupiter. A second use of computer simulations is to understand the world around us. One of the first important uses of computer simulations was to aid in the exploration for oil.

We rely on computer simulations to predict the path and speed of hurricanes. (Courtesy of NASA) Drilling a single well costs millions of dollars, and most drillings result in “dry wells” that produce no revenue. By using computer simulations, the process becomes much more predictable. Geologists lay out networks of microphones and set off explosive charges. Computers analyze the echoes received by the microphones to produce graphical representations of underground rock formations. Analyzing these formations helps petroleum engineers select the most promising sites to drill.

Computer simulations are also used to predict the future. Modern weather predictions are based on computer simulations. These predictions become particularly important when people are exposed to extreme weather conditions, such as floods, tornadoes, and hurricanes. Every computer simulation has an underlying mathematical model. Faster computers enable scientists and engineers to develop more sophisticated models. Over time, the quality of these models has improved.

**Validating Simulations**

A computer simulation may produce erroneous results for two fundamentally different reasons. The program may have a bug in it, or the model upon which the program is based may be flawed. ***Verification*** is the process of determining if the computer program correctly implements the model. ***Validation*** is the process of determining if the model is an accurate representation of the real system.

One way to validate a model is to make sure it duplicates the performance of the actual system. For example, automobile and truck manufacturers create computer models of their products. They use these models to see how well vehicles will perform in a variety of crash situations. Crashing an automobile on a computer is faster and much less expensive than crashing an actual car. To validate their models, manufacturers compare the results of crashing an actual vehicle with the results predicted by the computer model. Validating a model that predicts the future can introduce new difficulties. If we are predicting tomorrow’s weather, it is reasonable to validate the model by waiting until tomorrow and seeing how well the prediction held up. However, suppose you are a scientist using a global warming model to estimate what the climate will be like 50 years from now. You cannot validate this model by comparing its prediction with reality, because you cannot afford to wait 50 years to see if its prediction comes true. However, you can validate the model by using it to predict the present.



Figure illustrates how a model can predict the present. Suppose you want to see how well your model predicts events 25 years into the future. You have access to data going back 75 years. You let the model use data at least 25 years old, but you do not let the model see any data collected in the past 25 years. The job of predicting the present, given 25-year-old data, is presumably just as hard as the job of predicting 25 years into the future, given present data. The advantage of predicting the present is that you can use current data to validate the model.

A final way to validate a computer model is to see if it has credibility with experts and decision makers. Ultimately, a model is valuable only if it is believed by those who have the power to use its results to reach a conclusion or make a decision.

***Reference***

***Lecture 26 slides: Computer reliability (cont.)***

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