May 15, 2006

The Long, Dismal History of Software Project Failure

From the IEEE article Why Software Fails:

> Last October, for instance, the giant British food retailer J Sainsbury had to write off its US $526 million investment in an automated supply-chain management system. Merchandise was stuck in the company's depots and warehouses and was not getting through to many of its stores. Sainsbury was forced to hire about 3000 additional clerks to stock its shelves manually.
>
> This is only one of the latest in a long, dismal history of [software] projects gone awry. Most IT experts agree that such failures occur far more often than they should. What's more, the failures are universally unprejudiced: they happen in every country; to large companies and small; in commercial, nonprofit, and governmental organizations; and without regard to status or reputation. The business and societal costs of these failures -- in terms of wasted taxpayer and shareholder dollars as well as investments that can't be made -- are now well into the billions of dollars a year.
>
> The problem only gets worse as IT grows ubiquitous. This year, organizations and governments will spend an estimated $1 trillion on IT hardware, software, and services worldwide. Of the IT projects that are initiated, from 5 to 15 percent will be abandoned before or shortly after delivery as hopelessly inadequate. Many others will arrive late and over budget or require massive reworking. Few IT projects, in other words, truly succeed.

From Rapid Development:

> If Las Vegas sounds too tame for you, software might just be the right gamble. Software projects include a glut of risks that would give Vegas oddsmakers nightmares. The odds of a large project finishing on time are close to zero. The odds of a large project being canceled are an even-money bet (Jones 1991).
>
> In 1998, Peat Marwick found that about 35 percent of 600 firms surveyed had at least one runaway software project (Rothfeder 1988). The damage done by runaway software projects makes the Las Vegas prize fights look as tame as having high tea with the queen. Allstate set out in 1982 to automate all of its office operations. They set a 5-year timetable and an $8

million budget. Six years and $15 million later, Allstate set a new deadline and readjusted its sights on a new budget of $100 million. In 1988, Westpac Banking Corporation decided to redefine its information systems. It set out on a 5-year, $85 million project. Three years later, after spending $150 million with little to show for it, Westpac cut its losses, canceled the project, and eliminated 500 development jobs (Glass 1992). Even Vegas prize fights don't get this bloody.

The history of software development is a tremendous success. Just look around you for evidence of that. But that success has a long, dark shadow that we don't talk about very much: it's littered with colossal failures. What's particularly disturbing is that the colossal failures keep recurring year after year. The names and dollar amounts may change, but the story is otherwise the same. Two recent examples are the Canadian gun registry and the FBI's Virtual Case File system.

If you're looking for more examples of colossal software project failure, you don't have to look very far:

- Software Hall of Shame (from IEEE article Why Software Fails)
- History's Worst Software Bugs (Wired)
- Software Horror Stories (Nachum Deshowitz, Tel Aviv University)
- Forum on Computer Risks (ACM moderated mailing list)
- Failure Rate (collection of failure rate statistics from IT surveys)

You'd think that the software development industry would have matured over the last ten years. And it has:

The 10th edition of the annual CHAOS report from The Standish Group, which researches the reasons for IT project failure in the United States, indicates that project success rates have increased to 34 percent of all projects. That's more than a 100-percent improvement from the success rate found in the first study in 1994.

Asked for the chief reasons project success rates have improved, Standish Chairman Jim Johnson says, "The primary reason is the projects have gotten a lot smaller. Doing projects with iterative processing as opposed to the waterfall method, which called for all project requirements to be defined up front, is a major step forward."

The Standish Group has studied over 40,000 projects in 10 years to reach the findings.

Project failures have declined to 15 percent of all projects, a vast improvement over the 31-percent failure rate reported in 1994. Projects meeting the "challenged" description -- meaning that they are over time, over budget and/or lacking critical features and requirements -- total 51 percent of all projects in the current survey.

Failing is OK. Failing can even be desirable. But you must learn from your failures, and that requires concerted postmortem introspection and analysis. I'd like to think that a large part of the statistical improvement cited above is attributable to sharp project managers and savvy developers who **studied the first CHAOS report**. Once you know what the common pitfalls are, it's easier to avoid them.

Why Software Fails

We waste billions of dollars each year on entirely preventable mistakes

By ROBERT N. CHARETTE / SEPTEMBER 2005

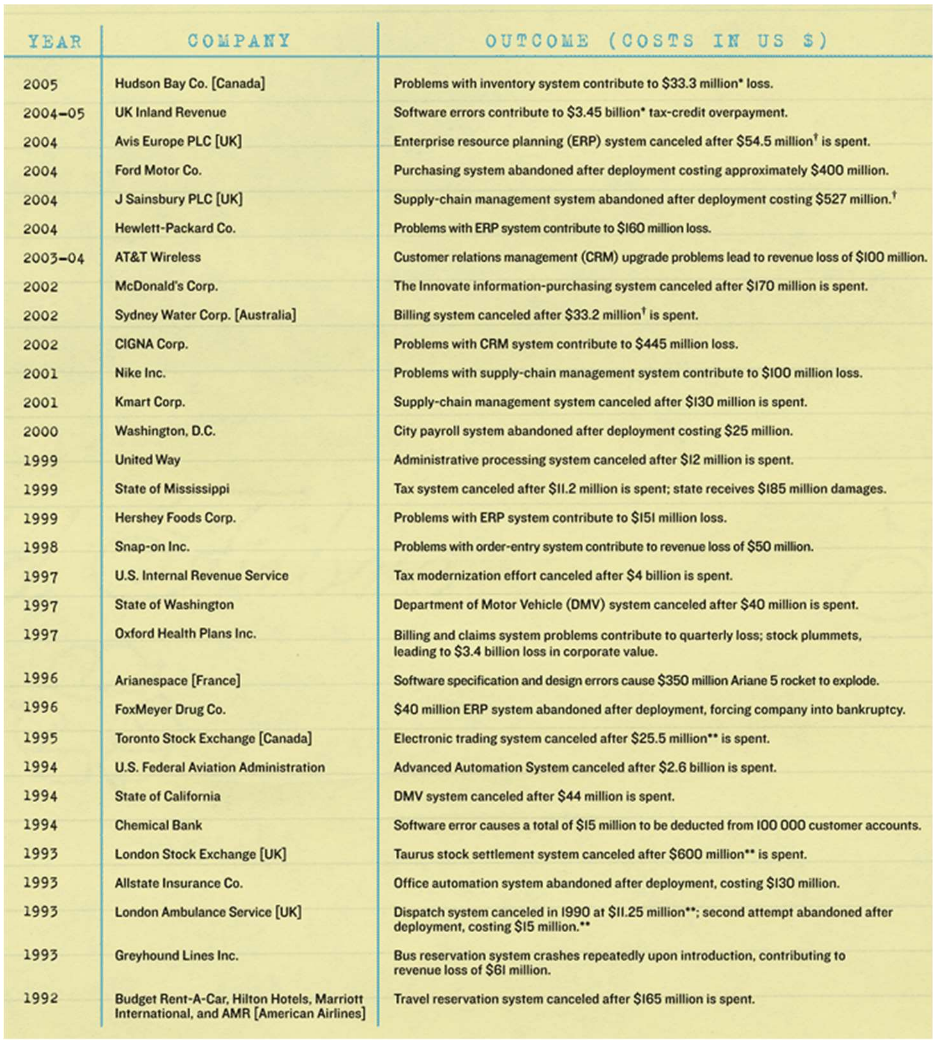
Photo: Odd Anersen/AFP/Getty Images

AIR JAM: The U.S. Federal Aviation Administration spent $2.6 billion trying to upgrade its air-traffic-control system, only to cancel the project in 1994. Gridlocked skies are still with us today.

Have you heard the one about the disappearing warehouse? One day, it vanished--not from physical view, but from the watchful eyes of a well-known retailer's automated distribution system. A software glitch had somehow erased the warehouse's existence, so that goods destined for the warehouse were rerouted elsewhere, while goods at the warehouse languished. Because the company was in financial trouble and had been shuttering other warehouses to save money, the employees at the "missing" warehouse kept quiet. For three years, nothing arrived or left. Employees were still getting their paychecks, however, because a different computer system handled the payroll. When the software glitch finally came to light, the merchandise in the warehouse was sold off, and upper management told employees to say nothing about the episode.

Photo: Graham Barclay/Bloomberg News/Landov
MARKET CRASH: After its new automated supply-chain management system failed last October, leaving merchandise stuck in company warehouses, British food retailer Sainsbury's had to hire 3000 additional clerks to stock its shelves.

This story has been floating around the information technology industry for 20-some years. It's probably apocryphal, but for those of us in the business, it's entirely plausible. Why? Because episodes like this happen all the time. Last October, for instance, the giant British food retailer J Sainsbury PLC had to write off its US $526 million investment in an automated supply-chain management system. It seems that merchandise was stuck in the company's depots and warehouses and was not getting through to many of its stores. Sainsbury was forced to hire about 3000 additional clerks to stock its shelves manually [see photo above, "Market Crash"].



YEAR	COMPANY	OUTCOME (COSTS IN US $)
2005	Hudson Bay Co. [Canada]	Problems with inventory system contribute to $33.3 million* loss.
2004–05	UK Inland Revenue	Software errors contribute to $3.45 billion* tax-credit overpayment.
2004	Avis Europe PLC [UK]	Enterprise resource planning (ERP) system canceled after $54.5 million† is spent.
2004	Ford Motor Co.	Purchasing system abandoned after deployment costing approximately $400 million.
2004	J Sainsbury PLC [UK]	Supply-chain management system abandoned after deployment costing $527 million.†
2004	Hewlett-Packard Co.	Problems with ERP system contribute to $160 million loss.
2003–04	AT&T Wireless	Customer relations management (CRM) upgrade problems lead to revenue loss of $100 million.
2002	McDonald's Corp.	The Innovate information-purchasing system canceled after $170 million is spent.
2002	Sydney Water Corp. [Australia]	Billing system canceled after $33.2 million† is spent.
2002	CIGNA Corp.	Problems with CRM system contribute to $445 million loss.
2001	Nike Inc.	Problems with supply-chain management system contribute to $100 million loss.
2001	Kmart Corp.	Supply-chain management system canceled after $130 million is spent.
2000	Washington, D.C.	City payroll system abandoned after deployment costing $25 million.
1999	United Way	Administrative processing system canceled after $12 million is spent.
1999	State of Mississippi	Tax system canceled after $11.2 million is spent; state receives $185 million damages.
1999	Hershey Foods Corp.	Problems with ERP system contribute to $151 million loss.
1998	Snap-on Inc.	Problems with order-entry system contribute to revenue loss of $50 million.
1997	U.S. Internal Revenue Service	Tax modernization effort canceled after $4 billion is spent.
1997	State of Washington	Department of Motor Vehicle (DMV) system canceled after $40 million is spent.
1997	Oxford Health Plans Inc.	Billing and claims system problems contribute to quarterly loss; stock plummets, leading to $3.4 billion loss in corporate value.
1996	Arianespace [France]	Software specification and design errors cause $350 million Ariane 5 rocket to explode.
1996	FoxMeyer Drug Co.	$40 million ERP system abandoned after deployment, forcing company into bankruptcy.
1995	Toronto Stock Exchange [Canada]	Electronic trading system canceled after $25.5 million** is spent.
1994	U.S. Federal Aviation Administration	Advanced Automation System canceled after $2.6 billion is spent.
1994	State of California	DMV system canceled after $44 million is spent.
1994	Chemical Bank	Software error causes a total of $15 million to be deducted from 100 000 customer accounts.
1993	London Stock Exchange [UK]	Taurus stock settlement system canceled after $600 million** is spent.
1993	Allstate Insurance Co.	Office automation system abandoned after deployment, costing $130 million.
1993	London Ambulance Service [UK]	Dispatch system canceled in 1990 at $11.25 million**; second attempt abandoned after deployment, costing $15 million.**
1993	Greyhound Lines Inc.	Bus reservation system crashes repeatedly upon introduction, contributing to revenue loss of $61 million.
1992	Budget Rent-A-Car, Hilton Hotels, Marriott International, and AMR [American Airlines]	Travel reservation system canceled after $165 million is spent.

This is only one of the latest in a long, dismal history of IT projects gone awry [see table above, "Software Hall of Shame" for other notable fiascoes]. Most IT experts agree that such failures occur far more often than they should. What's more, the failures are universally unprejudiced: they happen in every country; to large companies and small; in commercial, nonprofit, and governmental organizations; and without regard to status or reputation. The business and societal costs of these failures--in terms of wasted taxpayer and shareholder dollars as well as investments that can't be made--are now well into the billions of dollars a year.
The problem only gets worse as IT grows ubiquitous. This year, organizations and governments will spend an estimated $1 trillion on IT hardware, software, and services worldwide. Of the IT projects that are initiated, from 5 to 15 percent will be abandoned before or shortly after delivery as hopelessly inadequate. Many others will arrive late and over budget or require massive reworking. Few IT projects, in other words, truly succeed.
The biggest tragedy is that software failure is for the most part predictable and avoidable. Unfortunately, most organizations don't see preventing failure as an urgent matter, even though that view risks harming the organization and maybe even destroying it. Understanding why this attitude persists is not just an academic exercise; it has tremendous implications for business and society.

SOFTWARE IS EVERYWHERE. It's what lets us get cash from an ATM, make a phone call, and drive our cars. A typical cellphone now contains 2 million lines of software code; by 2010 it will likely have 10 times as many. General Motors Corp. estimates that by then its cars will each have 100 million lines of code.

The average company spends about 4 to 5 percent of revenue on information technology, with those that are highly IT dependent--such as financial and telecommunications companies--spending more than 10 percent on it. In other words, IT is now one of the largest corporate expenses outside employee costs. Much of that money goes into hardware and software upgrades, software license fees, and so forth, but a big chunk is for new software projects meant to create a better future for the organization and its customers.

Governments, too, are big consumers of software. In 2003, the United Kingdom had more than 100 major government IT projects under way that totaled $20.3 billion. In 2004, the U.S. government cataloged 1200 civilian IT projects costing more than $60 billion, plus another $16 billion for military software.

Any one of these projects can cost over $1 billion. To take two current examples, the computer modernization effort at the U.S. Department of Veterans Affairs is projected to run $3.5 billion, while automating the health records of the UK's National Health Service is likely to cost more than $14.3 billion for development and another $50.8 billion for deployment.

Such megasoftware projects, once rare, are now much more common, as smaller IT operations are joined into "systems of systems." Air traffic control is a prime example, because it relies on connections among dozens of networks that provide communications, weather, navigation, and other data. But the trick of integration has stymied many an IT developer, to the point where academic researchers increasingly believe that computer science itself may need to be rethought in light of these massively complex systems.

When a project fails , it jeopardizes an organization's prospects. If the failure is large enough, it can steal the company's entire future. In one stellar meltdown, a poorly implemented resource planning system led FoxMeyer Drug Co., a $5 billion wholesale drug distribution company in Carrollton, Texas, to plummet into bankruptcy in 1996.

IT failure in government can imperil national security, as the FBI's Virtual Case File debacle has shown. The $170 million VCF system, a searchable database intended to allow agents to "connect the dots" and follow up on disparate pieces of intelligence, instead ended five months ago without any system's being deployed [see "Who Killed the Virtual Case File?" in this issue].

IT failures can also stunt economic growth and quality of life. Back in 1981, the U.S. Federal Aviation Administration began looking into upgrading its antiquated air-traffic-control system, but the effort to build a replacement soon became riddled with problems [see photo, "Air Jam," at top of this article]. By 1994, when the agency finally gave up on the project, the predicted cost had tripled, more than $2.6 billion had been spent, and the expected delivery date had slipped by several years. Every airplane passenger who is delayed because of gridlocked skyways still feels this cancellation; the cumulative economic impact of all those delays on just the U.S. airlines (never mind the passengers) approaches $50 billion.

Worldwide, it's hard to say how many software projects fail or how much money is wasted as a result. If you define failure as the total abandonment of a project before or shortly after it is delivered, and if you accept a conservative failure rate of 5 percent, then billions of dollars are wasted each year on bad software.

For example, in 2004, the U.S. government spent $60 billion on software (not counting the embedded software in weapons systems); a 5 percent failure rate means $3 billion was probably wasted. However, after several decades as an IT consultant, I am convinced that the failure rate is 15 to 20 percent for projects that have budgets of $10 million or more. Looking at the total investment in new software projects--both government and corporate--over the last five years, I estimate that project failures have likely cost the U.S. economy at least $25 billion and maybe as much as $75 billion.

Of course, that $75 billion doesn't reflect projects that exceed their budgets--which most projects do. Nor does it reflect projects delivered late--which the majority are. It also fails to account for the opportunity costs of having to start over once a project is abandoned or the costs of bug-ridden systems that have to be repeatedly reworked.

Then, too, there's the cost of litigation from irate customers suing suppliers for poorly implemented systems. When you add up all these extra costs, the yearly tab for failed and troubled software conservatively runs somewhere from

$60 billion to $70 billion in the United States alone. For that money, you could launch the space shuttle 100 times, build and deploy the entire 24-satellite Global Positioning System, and develop the Boeing 777 from scratch--and still have a few billion left over.

Why do projects fail so often?

Among the most common factors:

- Unrealistic or unarticulated project goals
- Inaccurate estimates of needed resources
- Badly defined system requirements
- Poor reporting of the project's status
- Unmanaged risks
- Poor communication among customers, developers, and users
- Use of immature technology
- Inability to handle the project's complexity
- Sloppy development practices
- Poor project management
- Stakeholder politics
- Commercial pressures

Of course, IT projects rarely fail for just one or two reasons. The FBI's VCF project suffered from many of the problems listed above. Most failures, in fact, can be traced to a combination of technical, project management, and business decisions. Each dimension interacts with the others in complicated ways that exacerbate project risks and problems and increase the likelihood of failure.

Consider a simple software chore: a purchasing system that automates the ordering, billing, and shipping of parts, so that a salesperson can input a customer's order, have it automatically checked against pricing and contract requirements, and arrange to have the parts and invoice sent to the customer from the warehouse.

The requirements for the system specify four basic steps. First, there's the sales process, which creates a bill of sale. That bill is then sent through a legal process, which reviews the contractual terms and conditions of the potential sale and approves them. Third in line is the provision process, which sends out the parts contracted for, followed by the finance process, which sends out an invoice.

Let's say that as the first process, for sales, is being written, the programmers treat every order as if it were placed in the company's main location, even though the company has branches in several states and countries. That mistake, in turn, affects how tax is calculated, what kind of contract is issued, and so on.

The sooner the omission is detected and corrected, the better. It's kind of like knitting a sweater. If you spot a missed stitch right after you make it, you can simply unravel a bit of yarn and move on. But if you don't catch the mistake until the end, you may need to unravel the whole sweater just to redo that one stitch.

If the software coders don't catch their omission until final system testing--or worse, until after the system has been rolled out--the costs incurred to correct the error will likely be many times greater than if they'd caught the mistake while they were still working on the initial sales process.

And unlike a missed stitch in a sweater, this problem is much harder to pinpoint; the programmers will see only that errors are appearing, and these might have several causes. Even after the original error is corrected, they'll need to change other calculations and documentation and then retest every step.

In fact, studies have shown that software specialists spend about 40 to 50 percent of their time on avoidable rework rather than on what they call value-added work, which is basically work that's done right the first time. Once a piece of software makes it into the field, the cost of fixing an error can be 100 times as high as it would have been during the development stage.

If errors abound, then rework can start to swamp a project, like a dinghy in a storm. What's worse, attempts to fix an error often introduce new ones. It's like you're bailing out that dinghy, but you're also creating leaks. If too many errors are produced, the cost and time needed to complete the system become so great that going on doesn't make sense.

In the simplest terms, an IT project usually fails when the rework exceeds the value-added work that's been budgeted for. This is what happened to Sydney Water Corp., the largest water provider in Australia, when it attempted to introduce an automated customer information and billing system in 2002 [see box, "Case Study #2"]. According to an investigation by the Australian Auditor General, among the factors that doomed the project were inadequate planning and specifications, which in turn led to numerous change requests and significant added costs and delays. Sydney Water aborted the project midway, after spending AU $61 million (US $33.2 million).

All of which leads us to the obvious question: why do so many errors occur?

Software project failures have a lot in common with airplane crashes. Just as pilots never intend to crash, software developers don't aim to fail. When a commercial plane crashes, investigators look at many factors, such as the weather, maintenance records, the pilot's disposition and training, and cultural factors within the airline. Similarly, we need to look at the business environment, technical management, project management, and organizational culture to get to the roots of software failures.

Chief among the business factors are competition and the need to cut costs. Increasingly, senior managers expect IT departments to do more with less and do it faster than before; they view software projects not as investments but as pure costs that must be controlled.

Political exigencies can also wreak havoc on an IT project's schedule, cost, and quality. When Denver International Airport attempted to roll out its automated baggage-handling system, state and local political leaders held the project to one unrealistic schedule after another. The failure to deliver the system on time delayed the 1995 opening of the airport (then the largest in the United States), which compounded the financial impact manyfold.

Even after the system was completed, it never worked reliably: it chewed up baggage, and the carts used to shuttle luggage around frequently derailed. Eventually, United Airlines, the airport's main tenant, sued the system contractor, and the episode became a testament to the dangers of political expediency.

A lack of upper-management support can also damn an IT undertaking. This runs the gamut from failing to allocate enough money and manpower to not clearly establishing the IT project's relationship to the organization's business. In 2000, retailer Kmart Corp., in Troy, Mich., launched a $1.4 billion IT modernization effort aimed at linking its sales, marketing, supply, and logistics systems, to better compete with rival Wal-Mart Corp., in Bentonville, Ark. Wal-Mart proved too formidable, though, and 18 months later, cash-strapped Kmart cut back on modernization, writing off the $130 million it had already invested in IT. Four months later, it declared bankruptcy; the company continues to struggle today.

Frequently, IT project managers eager to get funded resort to a form of liar's poker, overpromising what their project will do, how much it will cost, and when it will be completed. Many, if not most, software projects start off with budgets that are too small. When that happens, the developers have to make up for the shortfall somehow, typically by trying to increase productivity, reducing the scope of the effort, or taking risky shortcuts in the review and testing phases. These all increase the likelihood of error and, ultimately, failure.

A state-of-the-art travel reservation system spearheaded by a consortium of Budget Rent-A-Car, Hilton Hotels, Marriott, and AMR, the parent of American Airlines, is a case in point. In 1992, three and a half years and $165 million into the project, the group abandoned it, citing two main reasons: an overly optimistic development schedule and an underestimation of the technical difficulties involved. This was the same group that had earlier built the hugely successful Sabre reservation system, proving that past performance is no guarantee of future results.

After crash investigators consider the weather as a factor in a plane crash, they look at the airplane itself. Was there something in the plane's design that caused the crash? Was it carrying too much weight?

In IT project failures, similar questions invariably come up regarding the project's technical components: the hardware and software used to develop the system and the development practices themselves. Organizations are often seduced by the siren song of the technological imperative--the uncontrollable urge to use the latest technology in hopes of gaining a competitive edge. With technology changing fast and promising fantastic new capabilities, it is easy to succumb. But using immature or untested technology is a sure route to failure.

In 1997, after spending $40 million, the state of Washington shut down an IT project that would have processed driver's licenses and vehicle registrations. Motor vehicle officials admitted that they got caught up in chasing

technology instead of concentrating on implementing a system that met their requirements. The IT debacle that brought down FoxMeyer Drug a year earlier also stemmed from adopting a state-of-the-art resource-planning system and then pushing it beyond what it could feasibly do.
A project's sheer size is a fountainhead of failure. Studies indicate that large-scale projects fail three to five times more often than small ones. The larger the project, the more complexity there is in both its static elements (the discrete pieces of software, hardware, and so on) and its dynamic elements (the couplings and interactions among hardware, software, and users; connections to other systems; and so on). Greater complexity increases the possibility of errors, because no one really understands all the interacting parts of the whole or has the ability to test them.
Sobering but true: it's impossible to thoroughly test an IT system of any real size. Roger S. Pressman pointed out in his book Software Engineering, one of the classic texts in the field, that "exhaustive testing presents certain logistical problems....Even a small 100-line program with some nested paths and a single loop executing less than twenty times may require 10 to the power of 14 possible paths to be executed." To test all of those 100 trillion paths, he noted, assuming each could be evaluated in a millisecond, would take 3170 years.
All IT systems are intrinsically fragile. In a large brick building, you'd have to remove hundreds of strategically placed bricks to make a wall collapse. But in a 100 000-line software program, it takes only one or two bad lines to produce major problems. In 1991, a portion of ATandamp;T's telephone network went out, leaving 12 million subscribers without service, all because of a single mistyped character in one line of code.
Sloppy development practices are a rich source of failure, and they can cause errors at any stage of an IT project. To help organizations assess their software-development practices, the U.S. Software Engineering Institute, in Pittsburgh, created the Capability Maturity Model, or CMM. It rates a company's practices against five levels of increasing maturity. Level 1 means the organization is using ad hoc and possibly chaotic development practices. Level 3 means the company has characterized its practices and now understands them. Level 5 means the organization quantitatively understands the variations in the processes and practices it applies.
As of January, nearly 2000 government and commercial organizations had voluntarily reported CMM levels. Over half acknowledged being at either level 1 or 2, 30 percent were at level 3, and only 17 percent had reached level 4 or 5. The percentages are even more dismal when you realize that this is a self-selected group; obviously, companies with the worst IT practices won't subject themselves to a CMM evaluation. (The CMM is being superseded by the CMM-Integration, which aims for a broader assessment of an organization's ability to create software-intensive systems.)
Immature IT practices doomed the U.S. Internal Revenue Service's $4 billion modernization effort in 1997, and they have continued to plague the IRS's current $8 billion modernization. It may just be intrinsically impossible to translate the tax code into software code--tax law is complex and based on often-vague legislation, and it changes all the time. From an IT developer's standpoint, it's a requirements nightmare. But the IRS hasn't been helped by open hostility between in-house and outside programmers, a laughable underestimation of the work involved, and many other bad practices.
THE PILOT'S ACTIONS JUST BEFORE a plane crashes are always of great interest to investigators. That's because the pilot is the ultimate decision-maker, responsible for the safe operation of the craft. Similarly, project managers play a crucial role in software projects and can be a major source of errors that lead to failure.

Back in 1986, the London Stock Exchange decided to automate its system for settling stock transactions. Seven years later, after spending $600 million, it scrapped the Taurus system's development, not only because the design was excessively complex and cumbersome but also because the management of the project was, to use the word of one of its own senior managers, "delusional." As investigations revealed, no one seemed to want to know the true status of the project, even as more and more problems appeared, deadlines were missed, and costs soared [see box, "Case Study #3"].
The most important function of the IT project manager is to allocate resources to various activities. Beyond that, the project manager is responsible for project planning and estimation, control, organization, contract management, quality management, risk management, communications, and human resource management.

Bad decisions by project managers are probably the single greatest cause of software failures today. Poor technical management, by contrast, can lead to technical errors, but those can generally be isolated and fixed. However, a bad project management decision--such as hiring too few programmers or picking the wrong type of contract--can wreak havoc. For example, the developers of the doomed travel reservation system claim that they were hobbled in part by the use of a fixed-price contract. Such a contract assumes that the work will be routine; the reservation system turned out to be anything but.

Project management decisions are often tricky precisely because they involve tradeoffs based on fuzzy or incomplete knowledge. Estimating how much an IT project will cost and how long it will take is as much art as science. The larger or more novel the project, the less accurate the estimates. It's a running joke in the industry that IT project estimates are at best within 25 percent of their true value 75 percent of the time.

There are other ways that poor project management can hasten a software project's demise. A study by the Project Management Institute, in Newton Square, Pa., showed that risk management is the least practiced of all project management disciplines across all industry sectors, and nowhere is it more infrequently applied than in the IT industry. Without effective risk management, software developers have little insight into what may go wrong, why it may go wrong, and what can be done to eliminate or mitigate the risks. Nor is there a way to determine what risks are acceptable, in turn making project decisions regarding tradeoffs almost impossible.

Poor project management takes many other forms, including bad communication, which creates an inhospitable atmosphere that increases turnover; not investing in staff training; and not reviewing the project's progress at regular intervals. Any of these can help derail a software project.

The last area that investigators look into after a plane crash is the organizational environment. Does the airline have a strong safety culture, or does it emphasize meeting the flight schedule above all? In IT projects, an organization that values openness, honesty, communication, and collaboration is more apt to find and resolve mistakes early enough that rework doesn't become overwhelming.

If there's a theme that runs through the tortured history of bad software, it's a failure to confront reality. On numerous occasions, the U.S. Department of Justice's inspector general, an outside panel of experts, and others told the head of the FBI that the VCF system was impossible as defined, and yet the project continued anyway. The same attitudes existed among those responsible for the travel reservation system, the London Stock Exchange's Taurus system, and the FAA's air-traffic-control project--all indicative of organizational cultures driven by fear and arrogance.

A recent report by the National Audit Office in the UK found numerous cases of government IT projects' being recommended not to go forward yet continuing anyway. The UK even has a government department charged with preventing IT failures, but as the report noted, more than half of the agencies the department oversees routinely ignore its advice. I call this type of behavior irrational project escalation--the inability to stop a project even after it's obvious that the likelihood of success is rapidly approaching zero. Sadly, such behavior is in no way unique.

In the final analysis , big software failures tend to resemble the worst conceivable airplane crash, where the pilot was inexperienced but exceedingly rash, flew into an ice storm in an untested aircraft, and worked for an airline that gave lip service to safety while cutting back on training and maintenance. If you read the investigator's report afterward, you'd be shaking your head and asking, "Wasn't such a crash inevitable?"

So, too, the reasons that software projects fail are well known and have been amply documented in countless articles, reports, and books [see sidebar, To Probe Further]. And yet, failures, near-failures, and plain old bad software continue to plague us, while practices known to avert mistakes are shunned. It would appear that getting quality software on time and within budget is not an urgent priority at most organizations.

It didn't seem to be at Oxford Health Plans Inc., in Trumbull, Conn., in 1997. The company's automated billing system was vital to its bottom line, and yet senior managers there were more interested in expanding Oxford's business than in ensuring that its billing system could meet its current needs [see box, "Case Study #1"]. Even as problems arose, such as invoices' being sent out months late, managers paid little attention. When the billing system effectively collapsed, the company lost tens of millions of dollars, and its stock dropped from $68 to $26 per share in one day, wiping out $3.4 billion in corporate value. Shareholders brought lawsuits, and several government agencies investigated the company, which was eventually fined $3 million for regulatory violations.

Even organizations that get burned by bad software experiences seem unable or unwilling to learn from their mistakes. In a 2000 report, the U.S. Defense Science Board, an advisory body to the Department of Defense, noted that various studies commissioned by the DOD had made 134 recommendations for improving its software development, but only 21 of those recommendations had been acted on. The other 113 were still valid, the board noted, but were being ignored, even as the DOD complained about the poor state of defense software development!

Some organizations do care about software quality, as the experience of the software development firm Praxis High Integrity Systems, in Bath, England, proves. Praxis demands that its customers be committed to the project, not only financially, but as active participants in the IT system's creation. The company also spends a tremendous amount of time understanding and defining the customer's requirements, and it challenges customers to explain what they want and why. Before a single line of code is written, both the customer and Praxis agree on what is desired, what is feasible, and what risks are involved, given the available resources.

After that, Praxis applies a rigorous development approach that limits the number of errors. One of the great advantages of this model is that it filters out the many would-be clients unwilling to accept the responsibility of articulating their IT requirements and spending the time and money to implement them properly. [See "The Exterminators," in this issue.]

Some level of software failure will always be with us. Indeed, we need true failures--as opposed to avoidable blunders--to keep making technical and economic progress. But too many of the failures that occur today are avoidable. And as our society comes to rely on IT systems that are ever larger, more integrated, and more expensive, the cost of failure may become disastrously high.

Even now, it's possible to take bets on where the next great software debacle will occur. One of my leading candidates is the IT systems that will result from the U.S. government's American Health Information Community, a public-private collaboration that seeks to define data standards for electronic medical records. The idea is that once standards are defined, IT systems will be built to let medical professionals across the country enter patient records digitally, giving doctors, hospitals, insurers, and other health-care specialists instant access to a patient's complete medical history. Health-care experts believe such a system of systems will improve patient care, cut costs by an estimated $78 billion per year, and reduce medical errors, saving tens of thousands of lives.

But this approach is a mere pipe dream if software practices and failure rates remain as they are today. Even by the most optimistic estimates, to create an electronic medical record system will require 10 years of effort, $320 billion in development costs, and $20 billion per year in operating expenses--assuming that there are no failures, overruns, schedule slips, security issues, or shoddy software. This is hardly a realistic scenario, especially because most IT experts consider the medical community to be the least computer-savvy of all professional enterprises.

Patients and taxpayers will ultimately pay the price for the development, or the failure, of boondoggles like this. Given today's IT practices, failure is a distinct possibility, and it would be a loss of unprecedented magnitude. But then, countries throughout the world are contemplating or already at work on many initiatives of similar size and impact--in aviation, national security, and the military, among other arenas.

Like electricity, water, transportation, and other critical parts of our infrastructure, IT is fast becoming intrinsic to our daily existence. In a few decades, a large-scale IT failure will become more than just an expensive inconvenience: it will put our way of life at risk. In the absence of the kind of industrywide changes that will mitigate software failures, how much of our future are we willing to gamble on these enormously costly and complex systems?

We already know how to do software well. It may finally be time to act on what we know.

About the Author

Robert N. Charette is president of ITABHI Corp., a risk-management consultancy in Spotsylvania, Va. An IEEE member, he is the author of several books on risk management and chair of the ISO/IEEE committee revising the 16085 standard on software and systems engineering risk management.

History's Worst Software Bugs

Simson Garfinkel 11.08.05

Last month automaker Toyota announced a recall of 160,000 of its Prius hybrid vehicles following reports of vehicle warning lights illuminating for no reason, and cars' gasoline engines stalling unexpectedly. But unlike the large-scale auto recalls of years past, the root of the Prius issue wasn't a hardware problem -- it was a programming error in the smart car's embedded code. The Prius had a software bug.

With that recall, the Prius joined the ranks of the buggy computer -- a club that began in 1945 when engineers found a moth in Panel F, Relay #70 of the Harvard Mark II system.The computer was running a test of its multiplier and adder when the engineers noticed something was wrong. The moth was trapped, removed and taped into the computer's logbook with the words: "first actual case of a bug being found."

Sixty years later, computer bugs are still with us, and show no sign of going extinct. As the line between software and hardware blurs, coding errors are increasingly playing tricks on our daily lives. Bugs don't just inhabit our operating systems and applications -- today they lurk within our cell phones and our pacemakers, our power plants and medical equipment. And now, in our cars.

But which are the worst?

It's all too easy to come up with a list of bugs that have wreaked havoc. It's harder to rate their severity. Which is worse -- a security vulnerability that's exploited by a computer worm to shut down the internet for a few days or a typo that triggers a day-long crash of the nation's phone system? The answer depends on whether you want to make a phone call or check your e-mail.

Many people believe the worst bugs are those that cause fatalities. To be sure, there haven't been many, but cases like the Therac-25 are widely seen as warnings against the widespread deployment of software in safety critical applications. Experts who study such systems, though, warn that even though the software might kill a few people, focusing on these fatalities risks inhibiting the migration of technology into areas where smarter processing is sorely needed. In the end, they say, the lack of software might kill more people than the inevitable bugs.

What seems certain is that bugs are here to stay. Here, in chronological order, is the *Wired News* list of the 10 worst software bugs of all time ... so far.

July 28, 1962 -- Mariner I space probe. A bug in the flight software for the Mariner 1 causes the rocket to divert from its intended path on launch. Mission control destroys the rocket over the Atlantic Ocean. The investigation into the accident discovers that a formula written on paper in pencil was improperly transcribed into computer code, causing the computer to miscalculate the rocket's trajectory.

1982 -- Soviet gas pipeline. Operatives working for the Central Intelligence Agency allegedly (.pdf) plant a bug in a Canadian computer system purchased to control the trans-Siberian gas pipeline. The Soviets had obtained the system as part of a wide-ranging effort to covertly purchase or steal sensitive U.S. technology. The CIA reportedly found out about the program and decided to make it backfire with equipment that would pass Soviet inspection and then fail once in operation. The resulting event is reportedly the largest non-nuclear explosion in the planet's history.

1985-1987 -- Therac-25 medical accelerator. A radiation therapy device malfunctions and delivers lethal radiation doses at several medical facilities. Based upon a previous design, the Therac-25 was an "improved" therapy system that could deliver two different kinds of radiation: either a low-power electron beam (beta particles) or X-rays. 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. A second "improvement" was the replacement of the older Therac-20's electromechanical safety interlocks with software control, a decision made because software was perceived to be more reliable.

What engineers didn't know was that both the 20 and the 25 were built upon an operating system that had been kludged together by a programmer with no formal training. Because of a subtle bug called a "race condition," a quick-fingered 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. At least five patients die; others are seriously injured.

1988 -- Buffer overflow in Berkeley Unix finger daemon. The first internet worm (the so-called Morris Worm) infects between 2,000 and 6,000 computers in less than a day by taking advantage of a buffer overflow. The specific code is a function in the standard input/output library routine called *gets()* designed to get a line of text over the network. Unfortunately, *gets()* has no provision to limit its input, and an overly large input allows the worm to take over any machine to which it can connect.

Programmers respond by attempting to stamp out the *gets()* function in working code, but they refuse to remove it from the C programming language's standard input/output library, where it remains to this day.

1988-1996 -- Kerberos Random Number Generator. The authors of the Kerberos security system neglect to properly "seed" the program's random number generator with a truly random seed. As a result, for eight years it is possible to trivially break into any computer that relies on Kerberos for authentication. It is unknown if this bug was ever actually exploited.

January 15, 1990 -- AT&T Network Outage. A bug in a new release of the software that controls AT&T's #4ESS long distance switches causes these mammoth computers to crash when they receive a specific message from one of their neighboring machines -- a message that the neighbors send out when they recover from a crash.

One day a switch in New York crashes and reboots, causing its neighboring switches to crash, then their neighbors' neighbors, and so on. Soon, 114 switches are crashing and rebooting every six seconds, leaving an estimated 60 thousand people without long distance service for nine hours. The fix: engineers load the previous software release.

1993 -- Intel Pentium floating point divide. A silicon error causes Intel's highly promoted Pentium chip to make mistakes when dividing floating-point numbers that occur within a specific range. For example, dividing 4195835.0/3145727.0 yields 1.33374 instead of 1.33382, an error of 0.006 percent. Although the bug affects few users, it becomes a public relations nightmare. With an estimated 3 million to 5 million defective chips in circulation, at first Intel only offers to replace Pentium chips for consumers who can prove that they need high accuracy; eventually the company relents and agrees to replace the chips for anyone who complains. The bug ultimately costs Intel $475 million.

1995/1996 -- The Ping of Death. A lack of sanity checks and error handling in the IP fragmentation reassembly code makes it possible to crash a wide variety of operating systems by sending a malformed "ping" packet from anywhere on the internet. Most obviously affected are computers running Windows, which lock up and display the so-called "blue screen of death" when they receive these packets. But the attack also affects many Macintosh and Unix systems as well.

June 4, 1996 -- Ariane 5 Flight 501. Working code for the Ariane 4 rocket is reused in the Ariane 5, but the Ariane 5's faster engines trigger a bug in an arithmetic routine inside the rocket's flight computer. The error is in the code that converts a 64-bit floating-point number to a 16-bit signed integer. The faster engines cause the 64-bit numbers to be larger in the Ariane 5 than in the Ariane 4, triggering an overflow condition that results in the flight computer crashing.

First Flight 501's backup computer crashes, followed 0.05 seconds later by a crash of the primary computer. As a result of these crashed computers, the rocket's primary processor overpowers the rocket's engines and causes the rocket to disintegrate 40 seconds after launch.

November 2000 -- National Cancer Institute, Panama City. In a series of accidents, therapy planning software created by Multidata Systems International, a U.S. firm, miscalculates the proper dosage of radiation for patients undergoing radiation therapy.

Multidata's software allows a radiation therapist to draw on a computer screen the placement of metal shields called "blocks" designed to protect healthy tissue from the radiation. But the software will only allow technicians to use four shielding blocks, and the Panamanian doctors wish to use five.

The doctors discover that they can trick the software by drawing all five blocks as a single large block with a hole in the middle. What the doctors don't realize is that the Multidata software gives different answers in this configuration depending on how the hole is drawn: draw it in one direction and the correct dose is calculated, draw in another direction and the software recommends twice the necessary exposure.

At least eight patients die, while another 20 receive overdoses likely to cause significant health problems. The physicians, who were legally required to double-check the computer's calculations by hand, are indicted for murder.

An Investigation of the Therac-25 Accidents

Nancy Leveson, University of Washington
Clark S. Turner, University of California, Irvine

Reprinted with permission, *IEEE Computer*, Vol. 26, No. 7, July 1993, pp. 18-41.

Computers are increasingly being introduced into safety-critical systems and, as a consequence, have been involved in accidents. Some of the most widely cited software-related accidents in safety-critical systems involved a computerized radiation therapy machine called the Therac-25. Between June 1985 and January 1987, six known accidents involved massive overdoses by the Therac-25 -- with resultant deaths and serious injuries. They have been described as the worst series of radiation accidents in the 35-year history of medical accelerators.[1]

With information for this article taken from publicly available documents, we present a detailed accident investigation of the factors involved in the overdoses and the attempts by the users, manufacturers, and the US and Canadian governments to deal with them. Our goal is to help others learn from this experience, not to criticize the equipment's manufacturer or anyone else. The mistakes that were made are not unique to this manufacturer but are, unfortunately, fairly common in other safety-critical systems. As Frank Houston of the US Food and Drug Administration (FDA) said, "A significant amount of software for life-critical systems comes from small firms, especially in the medical device industry; firms that fit the profile of those resistant to or uninformed of the principles of either system safety or software engineering."[2]

Furthermore, these problems are not limited to the medical industry. It is still a common belief that any good engineer can build software, regardless of whether he or she is trained in state-of-the-art software-engineering procedures. Many companies building safety-critical software are not using proper procedures from a software-engineering and safety-engineering perspective.

Most accidents are system accidents; that is, they stem from complex interactions between various components and activities. To attribute a single cause to an accident is usually a serious mistake. In this article, we hope to demonstrate the complex nature of accidents and the need to investigate all aspects of system development and operation to understand what has happened and to prevent future accidents.

Despite what can be learned from such investigations, fears of potential liability or loss of business make it difficult to find out the details behind serious engineering mistakes. When the equipment is regulated by government agencies, some information may be available. Occasionally, major accidents draw the attention of the US Congress or President and result in formal accident investigations (for instance, the Rogers commission investigation of the Challenger accident and the Kemeny commission investigation of the Three Mile Island incident).

The Therac-25 accidents are the most serious computer-related accidents to date (at least nonmilitary and admitted) and have even drawn the attention of the popular press. (Stories about the Therac-25 have appeared in trade journals, newspapers, People Magazine, and on television's 20/20 and McNeil/ Lehrer News Hour.) Unfortunately, the previous accounts of the Therac-25 problems have been oversimplified, with misleading omissions.

In an effort to remedy this, we have obtained information from a wide variety of sources, including lawsuits and the US and Canadian government agencies responsible for regulating such equipment. We have tried to be very careful to present only what we could document from original sources, but there is no guarantee that the documentation itself is correct. When possible, we looked for multiple confirming sources for the more important facts.

We have tried not to bias our description of the accidents, but it is difficult not to filter unintentionally what is described. Also, we were unable to investigate firsthand or get information about some aspects of the accidents that may be very relevant. For example, detailed information about the manufacturer's software development, management, and quality control was unavailable. We had to infer most information about these from statements in correspondence or other sources.

As a result, our analysis of the accidents may omit some factors. But the facts available support previous hypotheses about the proper development and use of software to control dangerous processes and suggest hypotheses that need further evaluation. Following our account of the accidents and the responses of the manufacturer, government agencies, and users, we present what we believe are the most compelling lessons to be learned in the context of software engineering, safety engineering, and government and user standards and oversight.

Genesis of the Therac-25

Medical linear accelerators (linacs) accelerate electrons to create high- energy beams that can destroy tumors with minimal impact on the surrounding healthy tissue. Relatively shallow tising healthy tissue. Relatively shallow tissue is treated with the accelerated electrons; to reach deeper tissue, the electron beam is converted into X-ray photons.

Atomic Energy Commission Limited (AECL) and a French company called CGR collaborated to build linear accelerators. (AECL is an arms-length entity, called a crown corporation, of the Canadian government. Since the time of the incidents related in this article, AECL Medical, a division of AECL, is in the process of being privatized and is now called Theratronics International Limited. Currently, AECL's primary business is the design and installation of nuclear reactors.) The products of AECL and CGR's cooperation were (1) the Therac-6, a 6 million electron volt (MeV) accelerator capable of producing X rays only and, later, (2) the Therac-20, a 20-MeV dual-mode (X rays or electrons) accelerator. Both were versions of older CGR machines, the Neptune and Sagittaire, respectively, which were augmented with computer control using a DEC PDP 11 minicomputer.

Software functionality was limited in both machines: The computer merely added convenience to the existing hardware, which was capable of standing alone. Industry-standard hardware safety features and interlocks in the underlying machines were retained. We know that some old Therac-6 software routines were used in the Therac-20 and that CGR developed the initial software.

The business relationship between AECL and CGR faltered after the Therac-20 effort. Citing competitive pressures, the two companies did not renew their cooperative agreement when scheduled in 1981. In the mid-1970s, AECL developed a radical new "double-pass" concept for electron acceleration. A double-pass accelerator needs much less space to develop comparable energy levels because it folds the long physical mechanism required to accelerate the electrons, and it is more economic to produce (since it uses a magnetron rather than a klystron as the energy source).

Using this double-pass concept, AECL designed the Therac-25, a dual-mode linear accelerator that can deliver either photons at 25 MeV or electrons at various energy levels (see Figure 1. Typical Therac-25 Facility.). Compared with the Therac-20, the Therac-25 is notably more compact, more versatile, and arguably easier to use. The higher energy takes advantage of the phenomenon of "depth dose": As the energy increases, the depth in the body at which maximum dose buildup occurs also increases, sparing the tissue above the target area. Economic advantages also come into play for the customer, since only one machine is required for both treatment modalities (electrons and photons).

Several features of the Therac-25 are important in understanding the accidents. First, like the Therac-6 and the Therac-20, the Therac-25 is controlled by a PDP 11. However, AECL designed the Therac-25 to take advantage of computer control from the outset; AECL did not build on a stand-alone machine. The Therac-6 and Therac-20 had been designed around machines that already had histories of clinical use without computer control.

In addition, the Therac-25 software has more responsibility for maintaining safety than the software in the previous machines. The Therac-20 has independent protective circuits for monitoring electron-beam scanning, plus mechanical interlocks for policing the machine and ensuring safe operation. The Therac-25 relies more on software for these functions. AECL took advantage of the computer's abilities to control and monitor the hardware and decided not to duplicate all the existing hardware safety mechanisms and interlocks. This approach is becoming more common as companies decide that hardware interlocks and backups are not worth the expense, or they put more faith (perhaps misplaced) on software than on hardware reliability.

Finally, some software for the machines was interrelated or reused. In a letter to a Therac-25 user, the AECL quality assurance manager said, "The same Therac-6 package was used by the AECL software people when they started the Therac-25 software. The Therac-20 and Therac-25 software programs were done independently, starting from a common base." Reuse of Therac-6 design features or modules may explain some of the problematic aspects of the Therac-25 software (see the sidebar "Therac-25 software development and design"). The quality assurance manager was apparently unaware that some Therac-20 routines were also used in the Therac-25; this was discovered after a bug related to one of the Therac-25 accidents was found in the Therac-20 software.

AECL produced the first hardwired prototype of the Therac-25 in 1976, and the completely computerized commercial version was available in late 1982. (The sidebars provide details about the machine's design and controlling software, important in understanding the accidents. Side-Bar: Therac-25 Software Development and Design)

In March 1983, AECL performed a safety analysis on the Therac-25. This analysis was in the form of a fault tree and apparently excluded the software. According to the final report, the analysis made several assumptions:

(1) Programming errors have been reduced by extensive testing on a hardware simulator and under field conditions on teletherapy units. Any residual software errors are not included in the analysis.

(2) Program software does not degrade due to wear, fatigue, or reproduction process.

(3) Computer execution errors are caused by faulty hardware components and by "soft" (random) errors induced by alpha particles and electromagnetic noise.

The fault tree resulting from this analysis does appear to include computer failure, although apparently, judging from these assumptions, it considers only hardware failures. For example, in one OR gate leading to the event of getting the wrong energy, a box contains "Computer selects wrong energy" and a probability of 10^11 is assigned to this event. For "Computer selects wrong mode," a probability of 4 x 10^9 is given. The report provides no justification of either number.

Side-Bar: Major Event Time Line

Accident history

Eleven Therac-25s were installed: five in the US and six in Canada. Six accidents involving massive overdoses to patients occurred between 1985 and 1987. The machine was recalled in 1987 for extensive design changes, including hardware safeguards against software errors.

Related problems were found in the Therac-20 software. These were not recognized until after the Therac-25 accidents because the Therac-20 included hardware safety interlocks and thus no injuries resulted.

In this section, we present a chro-nological account of the accidents and the responses from the manufacturer, government regulatory agencies, and users.users.

Kennestone Regional Oncology Center, 1985. Details of this accident in Marietta, Georgia, are sketchy since it was never carefully investigated. There was no admission that the injury was caused by the Therac-25 until long after the occurrence, despite claims by the patient that she had been injured during treatment, the obvious and severe radiation burns the patient suffered, and the suspicions of the radiation physicist involved.

After undergoing a lumpectomy to remove a malignant breast tumor, a 61-year-old woman was receiving follow-up radiation treatment to nearby lymph nodes on a Therac-25 at the Kennestone facility in Marietta. The Therac-25 had been operating at Kennestone for about six months; other Therac-25s had been operating, apparently without incident, since 1983.

On June 3, 1985, the patient was set up for a 10-MeV electron treatment to the clavicle area. When the machine turned on, she felt a "tremendous force of heat . . . this red-hot sensation." When the technician came in, the patient said, "You burned me." The technician replied that that was not possible. Although there were no marks on the patient at the time, the treatment area felt "warm to the touch."

It is unclear exactly when AECL learned about this incident. Tim Still, the Kennestone physicist, said that he contacted AECL to ask if the Therac-25 could operate in electron mode without scanning to spread the beam. Three days later, the engineers at AECL called the physicist back to explain that improper scanning was not possible.

In an August 19, 1986, letter from AECL to the FDA, the AECL quality assurance manager said, "In March of 1986, AECL received a lawsuit from the patient involved. . . This incident was never reported to AECL prior to this date, although some rather odd questions had been posed by Tim Still, the hospital physicist." The physicist at a hospital in Tyler, Texas, where a later accident occurred, reported, "According to Tim Still, the patient filed suit in October 1985 listing the hospital, manufacturer, and service organization responsible for the machine. AECL was notified informally about the suit by the hospital, and AECL received official notification of a lawsuit in November 1985."

Because of the lawsuit (filed on November 13, 1985), some AECL administrators must have known about the Marietta accident -- although no investigation occurred at this time. Further comments by FDA investigators point to the lack of a mechanism in AECL to follow up reports of suspected accidents. The lack of follow-up in this case appears to be evidence of such a problem in the organization.

The patient went home, but shortly afterward she developed a reddening and swelling in the center of the treatment area. Her pain had increased to the point that her shoulder "froze" and she experienced spasms. She was admitted to West Paces Ferry Hospital in Atlanta, but her oncologists continued to send her to Kennestone for Therac-25 treatments. Clinical explanation was sought for the reddening of the skin, which at first her oncologist attributed to her disease or to normal treatment reaction.

About two weeks later, the physicist at Kennestone noticed that the patient had a matching reddening on her back as though a burn had gone through her body, and the swollen area had begun to slough off layers of skin. Her shoulder was immobile, and she was apparently in great pain. It was obvious that she had a radiation burn, but the hospital and her doctors could provide no satisfactory explanation. Shortly afterward, she initiated a lawsuit against the hospital and AECL regarding her injury.

The Kennestone physicist later estimated that she received one or two doses of radiation in the 15,000- to 20,000-rad (radiation absorbed dose) range. He does not believe her injury could have been caused by less than 8,000 rads. Typical single therapeutic doses are in the 200-rad range. Doses of 1,000 rads can be fatal if delivered to the whole body; in fact, the accepted figure for whole-body radiation that will cause death in 50 percent of the cases is 500 rads. The consequences of an overdose to a smaller part of the body depend on the tissue's radiosensitivity. The director of radiation oncology at the Kennestone facility explained their confusion about the accident as due to the fact that they had never seen an overtreatment of that magnitude before.

Eventually, the patient's breast had to be removed because of the radiation burns. She completely lost the use of her shoulder and her arm, and was in constant pain. She had suffered a serious radiation burn, but the manufacturer and operators of the machine refused to believe that it could have been caused by the Therac-25. The treatment prescription printout feature was disabled at the time of the accident, so there was no hard copy of the treatment data. The lawsuit was eventually settled out of court.

From what we can determine, the accident was not reported to the FDA until after the later Tyler accidents in 1986 (described in later sections). The reporting regulations for medical device incidents at that time applied only to equipment manufacturers and importers, not users. The regulations required that manufacturers and importers report deaths, serious injuries, or malfunctions that could result in those consequences. Health-care professionals and institutions were not required to report incidents to manufacturers. (The law was amended in 1990 to require health-care facilities to report incidents to the manufacturer and the FDA.) The comptroller general of the US Government Accounting Office, in testimony before Congress on November 6, 1989, expressed great concern about the viability of the incident-reporting regulations in preventing or spotting medical-device problems. According to a GAO study, the FDA knows of less than 1 percent of deaths, serious injuries, or equipment malfunctions that occur in hospitals.[3]

At this point, the other Therac-25 users were unaware that anything untoward had occurred and did not learn about any problems with the machine until after subsequent accidents. Even then, most of their information came through personal communication among themselves.

Ontario Cancer Foundation, 1985. The second in this series of accidents occurred at this Hamilton, Ontario, Canada, clinic about seven weeks after the Kennestone patient was overdosed. At that time, the Therac-25 at the Hamilton clinic had been in use for more than six months. On July 26, 1985, a 40-year-old patient came to the clinic for her 24th Therac-25 treatment for carcinoma of the cervix. The operator activated the machine, but the Therac shut down after five seconds with an "H-tilt" error message. The Therac's dosimetry system display read "no dose" and indicated a "treatment pause."

Since the machine did not suspend and the control display indicated no dose was delivered to the patient, the operator went ahead with a second attempt at treatment by pressing the "P" key (the proceed command), expecting the machine to deliver the proper dose this time. This was standard operating procedure and, as described in the sidebar The operator interface, Therac-25 operators had become accustomed to frequent malfunctions that had no untoward consequences for the patient. Again, the machine shut down in the same manner. The operator repeated this process four times after the original attempt -- the display showing "no dose" delivered to the patient each time. After the fifth pause, the machine went into treatment suspend, and a hospital service technician was called. The technician found nothing wrong with the machine. This also was not an unusual scenario, according to a Therac-25 operator.

After the treatment, the patient complained of a burning sensation, described as an "electric tingling shock" to the treatment area in her hip. Six other patients were treated later that day without incident. The patient came back for further treatment on July 29 and complained of burning, hip pain, and excessive swelling in the region of treatment. The machine was taken out of service, as radiation overexposure was suspected. The patient was hospitalized for the condition on July 30. AECL was informed of the apparent radiation injury and sent a service engineer to investigate. The FDA, the then-Canadian Radiation Protection Bureau (CRPB), and the users were informed that there was a problem, although the users claim that they were never informed that a patient injury had occurred. (On April 1, 1986, the CRPB and the Bureau of Medical Devices were merged to form the Bureau of Radiation and Medical Devices or BRMD.)

Users were told that they should visually confirm the turntable alignment until further notice (which occurred three months later).

The patient died on November 3, 1985, of an extremely virulent cancer. An autopsy revealed the cause of death as the cancer, but it was noted that had she not died, a total hip replacement would have been necessary as a result of the radiation overexposure. An AECL technician later estimated the patient had received between 13,000 and 17,000 rads.

Manufacturer response. AECL could not reproduce the malfunction that had occurred, but suspected a transient failure in the microswitch used to determine turntable position. During the investigation of the accident, AECL hardwired the error conditions they assumed were necessary for the malfunction and, as a result, found some design weaknesses and potential mechanical problems involving the turntable positioning.

The computer senses and controls turntable position by reading a 3-bit signal about the status of three microswitches in the turntable switch assembly (see the sidebar Turntable positioning). Essentially, AECL determined that a 1-bit error in the microswitch codes (which could be caused by a single open-circuit fault on the switch lines) could produce an ambiguous position message for the computer. The problem was exacerbated by the design of the mechanism that extends a plunger to lock the turntable when it is in one of the three cardinal positions: The plunger could be extended when the turntable was way out of position, thus giving a second false position indication. AECL devised a method to indicate turntable position that tolerated a 1-bit error: The code would still unambiguously reveal correct position with any one microswitch failure.

In addition, AECL altered the software so that the computer checked for "in transit" status of the switches to keep further track of the switch operation and the turntable position, and to give additional assurance that the switches were working and the turntable was moving.

As a result of these improvements, AECL claimed in its report and correspondence with hospitals that "analysis of the hazard rate of the new solution indicates an improvement over the old system by at least five orders of magnitude." A claim that safety had been improved by five orders of magnitude seems exaggerated, especially given that in its final incident report to the FDA, AECL concluded that it "cannot be firm on the exact cause of the accident but can only suspect. . ." This underscores the company's inability to determine the cause of the accident with any certainty. The AECL quality assurance manager testified that AECL could not reproduce the switch malfunction and that testing of the microswitch was "inconclusive." The similarity of the errant behavior and the injuries to patients in this accident and a later one in Yakima, Washington, (attributed to software error) provide good reason to believe that the Hamilton overdose was probably related to software error rather than to a microswitch failure.

Government and user response. The Hamilton accident resulted in a voluntary recall by AECL, and the FDA termed it a Class II recall. Class II means "a situation in which the use of, or exposure to, a violative product may cause temporary or medically reversible adverse health consequences or where the probability of serious adverse health consequences is remote." Four users in the US were advised by a letter from AECL on August 1, 1985, to visually check the ionization chamber to make sure it was in its correct position in the collimator opening before any treatment and to discontinue treatment if they got an H-tilt message with an incorrect dose indicated. The letter did not mention that a patient injury was involved. The FDA audited AECL's subsequent modifications. After the modifications, the users were told that they could return to normal operating procedures.

As a result of the Hamilton accident, the head of advanced X-ray systems in the CRPB, Gordon Symonds, wrote a report that analyzed the design and performance characteristics of the Therac-25 with respect to radiation safety. Besides citing the flawed microswitch, the report faulted both hardware and software components of the Therac's design. It concluded with a list of four modifications to the Therac-25 necessary for minimum compliance with Canada's Radiation Emitting Devices (RED) Act. The RED law, enacted in 1971, gives government officials power to ensure the safety of radiation-emitting devices.

The modifications recommended in the Symonds report included redesigning the microswitch and changing the way the computer handled malfunction conditions. In particular, treatment was to be terminated in the event of a dose-rate malfunction, giving a treatment "suspend." This would have removed the option to proceed simply by pressing the "P" key. The report also made recommendations regarding collimator test procedures and message and command formats. A November 8, 1985 letter signed by Ernest Létourneau, M.D., director of the CRPB, asked that AECL make changes to the Therac-25 based on the Symonds report "to be in compliance with the RED Act."

Although, as noted above, AECL did make the microswitch changes, it did not comply with the directive to change the malfunction pause behavior into treatment suspends, instead reducing the maximum number of retries from five to three. According to Symonds, the deficiencies outlined in the CRPB letter of November 8 were still pending when subsequent accidents five months later changed the priorities. If these later accidents had not occurred, AECL would have been compelled to comply with the requirements in the letter.

Immediately after the Hamilton accident, the Ontario Cancer Foundation hired an independent consultant to investigate. He concluded in a September 1985 report that an independent system (beside the computer) was needed to verify turntable position and suggested the use of a potentiometer. The CRPB wrote a letter to AECL in November 1985 requesting that AECL install such an independent upper collimator positioning interlock on the Therac-25. Also, in January 1986, AECL received a letter from the attorney representing the Hamilton clinic. The letter said there had been continuing problems with the turntable, including four incidents at Hamilton, and requested the installation of an independent system (potentiometer) to verify turntable position. AECL did not comply: No independent interlock was installed on the Therac-25s at this time.

Yakima Valley Memorial Hospital, 1985. As with the Kennestone overdose, machine malfunction in this accident in Yakima, Washington, was not acknowledged until after later accidents were understood.

The Therac-25 at Yakima had been modified in September 1985 in response to the overdose at Hamilton. During December 1985, a woman came in for treatment with the Therac-25. She developed erythema (excessive reddening of the skin) in a parallel striped pattern at one port site (her right hip) after one of the treatments. Despite this, she continued to be treated by the Therac-25 because the cause of her reaction was not determined to be abnormal until January or February of 1986. On January 6, 1986, her treatments were completed.

The staff monitored the skin reaction closely and attempted to find possible causes. The open slots in the blocking trays in the Therac-25 could have produced such a striped pattern, but by the time the skin reaction had been determined to be abnormal, the blocking trays had been discarded. The blocking arrangement and tray striping orientation could not be reproduced. A reaction to chemotherapy was ruled out because that should have produced reactions at the other ports and would not have produced stripes. When it was discovered that the woman slept with a heating pad, a possible explanation was offered on the basis of the parallel wires that deliver the heat in such pads. The staff x-rayed the heating pad and discovered that the wire pattern did not correspond to the erythema pattern on the patient's hip.

The hospital staff sent a letter to AECL on January 31, and they also spoke on the phone with the AECL technical support supervisor. On February 24, 1986, the AECL technical support supervisor sent a written response to the director of radiation therapy at Yakima saying, "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." The letter goes on to support this opinion by listing two pages of technical reasons why an overdose by the Therac-25 was impossible, along with the additional argument that there have "apparently been no other instances of similar damage to this or other patients." The letter ends, "In closing, I wish to advise that this matter has been brought to the attention of our Hazards Committee, as is normal practice."

The hospital staff eventually ascribed the skin/tissue problem to "cause unknown." In a report written on this first Yakima incident after another Yakima overdose a year later (described in a later section), the medical physicist involved wrote

At that time, we did not believe that [the patient] was overdosed because the manufacturer had installed additional hardware and software safety devices to the accelerator.

In a letter from the manufacturer dated 16-Sep-85, it is stated that "Analysis of the hazard rate resulting from these modifications indicates an improvement of at least five orders of magnitude"! With such an improvement in safety (10,000,000 percent) we did not believe that there could have been any accelerator malfunction. These modifications to the accelerator were completed on 5,6-Sep-85.

Even with fairly sophisticated physics support, the hospital staff, as users, did not have the ability to investigate the possibility of machine malfunction further. They were not aware of any other incidents, and, in fact, were told that there had been none, so there was no reason for them to pursue the matter. However, it seems that the fact that three similar incidents had occurred with this equipment should have triggered some suspicion and investigation by the manufacturer and the appropriate government agencies. This assumes, of course, that these incidents were all reported and known by AECL and by the government regulators. If they were not, then it is appropriate to ask why they were not and how this could be remedied in the future.

About a year later (in February 1987), after the second Yakima overdose led the hospital staff to suspect that the first injury had been due to a Therac-25 fault, the staff investigated and found that this patient had a chronic skin ulcer, tissue necrosis (death) under the skin, and was in constant pain. This was surgically repaired, skin grafts were made, and the symptoms relieved. The patient is alive today, with minor disability and some scarring related to the overdose. The hospital staff concluded that the dose accidentally delivered to this patient must have been much lower than in the second accident, as the reaction was significantly less intense and necrosis did not develop until six to eight months after exposure. Some other factors related to the place on the body where the overdose occurred also kept her from having more significant problems as a result of the exposure.

East Texas Cancer Center, March 1986. More is known about the Tyler, Texas, accidents than the others because of the diligence of the Tyler hospital physicist, Fritz Hager, without whose efforts the understanding of the software problems might have been delayed even further.

The Therac-25 was at the East Texas Cancer Center (ETCC) for two years before the first serious accident occurred; during that time, more than 500 patients had been treated. On March 21, 1986, a male patient came into ETCC for his ninth treatment on the Therac-25, one of a series prescribed as follow-up to the removal of a tumor from his back.

The patient's treatment was to be a 22-MeV electron-beam treatment of 180 rads over a 10 x 17-cm field on the upper back and a little to the left of his spine, or a total of 6,000 rads over a period of 6 1/2 weeks. He was taken into the treatment room and placed face down on the treatment table. The operator then left the treatment room, closed the door, and sat at the control terminal.

The operator had held this job for some time, and her typing efficiency had increased with experience. She could quickly enter prescription data and change it conveniently with the Therac's editing features. She entered the patient's prescription data quickly, then noticed that for mode she had typed "x" (for X ray) when she had intended "e" (for electron). This was a common mistake since most treatments involved X rays, and she had become accustomed to typing this. The mistake was easy to fix; she merely used the cursor up key to edit the mode entry.

Since the other parameters she had entered were correct, she hit the return key several times and left their values unchanged. She reached the bottom of the screen where a message indicated that the parameters had been "verified" and the terminal displayed "beam ready," as expected. She hit the one-key command "B" (for "beam on") to begin the treatment. After a moment, the machine shut down and the console displayed the message "Malfunction 54." The machine also displayed a "treatment pause," indicating a problem of low priority (see the operator interface sidebar). The sheet on the side of the machine explained that this malfunction was a "dose input 2" error. The ETCC did not have any other information available in its instruction manual or other Therac-25 documentation to explain the meaning of Malfunction 54. An AECL technician later testified that "dose input 2" meant that a dose had been delivered that was either too high or too low.

The machine showed a substantial underdose on its dose monitor display: 6 monitor units delivered, whereas the operator had requested 202 monitor units. The operator was accustomed to the quirks of the machine, which would frequently stop or delay treatment. In the past, the only consequences had been inconvenience. She immediately took the normal action when the machine merely paused, which was to hit the "P" key to proceed with the treatment. The machine promptly shut down with the same "Malfunction 54" error and the same underdose shown by the display terminal.

The operator was isolated from the patient, since the machine apparatus was inside a shielded room of its own. The only way the operator could be alerted to patient difficulty was through audio and video monitors. On this day, the video display was unplugged and the audio monitor was broken.

After the first attempt to treat him, the patient said that he felt like he had received an electric shock or that someone had poured hot coffee on his back: He felt a thump and heat and heard a buzzing sound from the equipment. Since this was his ninth treatment, he knew that this was not normal. He began to get up from the treatment table to go for help. It was at this moment that the operator hit the "P" key to proceed with the treatment. The patient said that he felt like his arm was being shocked by electricity and that his hand was leaving his body. He went to the treatment room door and pounded on it. The operator was shocked and immediately opened the door for him. He appeared shaken and upset.

The patient was immediately examined by a physician, who observed intense erythema over the treatment area, but suspected nothing more serious than electric shock. The patient was discharged with instructions to return if he suffered any further reactions. The hospital physicist was called in, and he found the machine calibration within specifications. The meaning of the malfunction message was not understood. The machine was then used to treat patients for the rest of the day.

In actuality, but unknown to anyone at that time, the patient had received a massive overdose, concentrated in the center of the treatment area. After-the-fact simulations of the accident revealed possible doses of 16,500 to 25,000 rads in less than 1 second over an area of about 1 cm.

During the weeks following the accident, the patient continued to have pain in his neck and shoulder. He lost the function of his left arm and had periodic bouts of nausea and vomiting. He was eventually hospitalized for radiation-induced myelitis of the cervical cord causing paralysis of his left arm and both legs, left vocal cord paralysis (which left him unable to speak), neurogenic bowel and bladder, and paralysis of the left diaphragm. He also had a lesion on his left lung and recurrent herpes simplex skin infections. He died from complications of the overdose five months after the accident.

User and manufacturer response. The Therac-25 was shut down for testing the day after this accident. One local AECL engineer and one from the home office in Canada came to ETCC to investigate. They spent a day running the machine through tests but could not reproduce a Malfunction 54. The AECL home office engineer reportedly explained that it was not possible for the Therac-25 to overdose a patient. The ETCC physicist claims that he asked AECL at this time if there were any other reports of radiation overexposure and that the AECL personnel (including the quality assurance manager) told him that AECL knew of no accidents involving radiation overexposure by the Therac-25. This seems odd since AECL was surely at least aware of the Hamilton accident that had occurred seven months before and the Yakima accident, and, even by its own account, AECL learned of the Georgia lawsuit about this time (the suit had been filed four months earlier). The AECL engineers then suggested that an electrical problem might have caused this accident.

The electric shock theory was checked out thoroughly by an independent engineering firm. The final report indicated that there was no electrical grounding problem in the machine, and it did not appear capable of giving a patient an electrical shock. The ETCC physicist checked the calibration of the Therac-25 and found it to be satisfactory. The center put the machine back into service on April 7, 1986, convinced that it was performing properly.

East Texas Cancer Center, April 1986. Three weeks after the first ETCC accident, on Friday, April 11, 1986, another male patient was scheduled to receive an electron treatment at ETCC for a skin cancer on the side of his face. The prescription was for 10 MeV to an area of approximately 7 x 10 cm. The same technician who had treated the first Tyler accident victim prepared this patient for treatment. Much of what follows is from the deposition of the Tyler Therac-25 operator.

As with her former patient, she entered the prescription data and then noticed an error in the mode. Again she used the cursor up key to change the mode from X ray to electron. After she finished editing, she pressed the return key several times to place the cursor on the bottom of the screen. She saw the "beam ready" message displayed and turned the beam on.

Within a few seconds the machine shut down, making a loud noise audible via the (now working) intercom. The display showed Malfunction 54 again. The operator rushed into the treatment room, hearing her patient moaning for help. The patient began to remove the tape that had held his head in position and said something was wrong. She asked him what he felt, and he replied "fire" on the side of his face. She immediately went to the hospital physicist and told him that another patient appeared to have been burned. Asked by the physicist to describe what he had experienced, the patient explained that something had hit him on the side of the face, he saw a flash of light, and he heard a sizzling sound reminiscent of frying eggs. He was very agitated and asked, "What happened to me, what happened to me?"

This patient died from the overdose on May 1, 1986, three weeks after the accident. He had disorientation that progressed to coma, fever to 104 degrees Fahrenheit, and neurological damage. Autopsy showed an acute high-dose radiation injury to the right temporal lobe of the brain and the brain stem.

User and manufacturer response. After this second Tyler accident, the ETCC physicist immediately took the machine out of service and called AECL to alert the company to this second apparent overexposure. The Tyler physicist then began his own careful investigation. He worked with the operator, who remembered exactly what she had done on this occasion. After a great deal of effort, they were eventually able to elicit the Malfunction 54 message. They determined that data-entry speed during editing was the key factor in producing the error condition: If the prescription data was edited at a fast pace (as is natural for someone who has repeated the procedure a large number of times), the overdose occurred.

It took some practice before the physicist could repeat the procedure rapidly enough to elicit the Malfunction 54 message at will. Once he could do this, he set about measuring the actual dose delivered under the error condition. He took a measurement of about 804 rads but realized that the ion chamber had become saturated. After making adjustments to extend his measurement ability, he determined that the dose was somewhere over 4,000 rads.

The next day, an engineer from AECL called and said that he could not reproduce the error. After the ETCC physicist explained that the procedure had to be performed quite rapidly, AECL could finally produce a similar malfunction on its own machine. AECL then set up its own set of measurements to test the dosage delivered. Two days after the accident, AECL said they had measured the dosage (at the center of the field) to be 25,000 rads. An AECL engineer explained that the frying sound heard by the patient was the ion chambers being saturated.

In fact, it is not possible to determine the exact dose each of the accident victims received; the total dose delivered during the malfunction conditions was found to vary enormously when different clinics simulated the faults. The number of pulses delivered in the 0.3 second that elapsed before interlock shutoff varied because the software adjusted the start-up pulse-repetition frequency to very different values on different machines. Therefore, there is still some uncertainty as to the doses actually received in the accidents.[1]

In one lawsuit that resulted from the Tyler accidents, the AECL quality control manager testified that a "cursor up" problem had been found in the service mode at the Kennestone clinic and one other clinic in February or March 1985 and also in the summer of 1985. Both times, AECL thought that the software problems had been fixed. There is no way to determine whether there is any relationship between these problems and the Tyler accidents.

Related Therac-20 problems. After the Tyler accidents, Therac-20 users (who had heard informally about the Tyler accidents from Therac-25 users) conducted informal investigations to determine whether the same problem could occur with their machines. As noted earlier, the software for the Therac-25 and Therac-20 both "evolved" from the Therac-6 software. Additional functions had to be added because the Therac-20 (and Therac-25) operates in both X-ray and electron mode, while the Therac-6 has only X-ray mode. The CGR employees modified the software for the Therac-20 to handle the dual modes.

When the Therac-25 development began, AECL engineers adapted the software from the Therac-6, but they also borrowed software routines from the Therac-20 to handle electron mode.

The agreements between AECL and CGR gave both companies the right to tap technology used in joint products for their other products.

After the second Tyler accident, a physicist at the University of Chicago Joint Center for Radiation Therapy heard about the Therac-25 software problem and decided to find out whether the same thing could happen with the Therac-20. At first, the physicist was unable to reproduce the error on his machine, but two months later he found the link.

The Therac-20 at the University of Chicago is used to teach students in a radiation therapy school conducted by the center. The center's physicist, Frank Borger, noticed that whenever a new class of students started using the Therac-20, fuses and breakers on the machine tripped, shutting down the unit. These failures, which had been occurring ever since the center had acquired the machine, might appear three times a week while new students operated the machine and then disappear for months. Borger determined that new students make lots of different types of mistakes and use "creative methods of editing" parameters on the console. Through experimentation, he found that certain editing sequences correlated with blown fuses and determined that the same computer bug (as in the Therac-25 software) was responsible. The physicist notified the FDA, which notified Therac-20 users.[4]

The software error is just a nuisance on the Therac-20 because this machine has independent hardware protective circuits for monitoring the electron-beam scanning. The protective circuits do not allow the beam to turn on, so there is no danger of radiation exposure to a patient. While the Therac-20 relies on mechanical interlocks for monitoring the machine, the Therac-25 relies largely on software.

The software problem. A lesson to be learned from the Therac-25 story is that focusing on particular software bugs is not the way to make a safe system. Virtually all complex software can be made to behave in an unexpected fashion under certain conditions. The basic mistakes here involved poor software-engineering practices and building a machine that relies on the software for safe operation. Furthermore, the particular coding error is not as important as the general unsafe design of the software overall. Examining the part of the code blamed for the Tyler accidents is instructive, however, in showing the overall software design flaws. The following explanation of the problem is from the description AECL provided for the FDA, although we have tried to clarify it somewhat. The description leaves some unanswered questions, but it is the best we can do with the information we have.

As described in the sidebar on Therac-25 software development and design, the treatment monitor task (Treat) controls the various phases of treatment by executing its eight subroutines (see Figure 2). The treatment phase indicator variable (Tphase) is used to determine which subroutine should be executed. Following the execution of a particular subroutine, Treat reschedules itself.

One of Treat's subroutines, called Datent (data entry), communicates with the keyboard handler task (a task that runs concurrently with Treat) via a shared variable (Data-entry completion flag) to determine whether the prescription data has been entered. The keyboard handler recognizes the completion of data entry and changes the Data-entry completion variable to denote this. Once the Data-entry completion variable is set, the Datent subroutine detects the variable's change in status and changes the value of Tphase from 1 (Data Entry) to 3 (Set-Up Test). In this case, the Datent subroutine exits back to the Treat subroutine, which will reschedule itself and begin execution of the Set-Up Test subroutine. If the Data-entry completion variable has not been set, Datent leaves the value of Tphase unchanged and exits back to Treat's main line. Treat will then reschedule itself, essentially rescheduling the Datent subroutine.

The command line at the lower right corner of the screen is the cursor's normal position when the operator has completed all necessary changes to the prescription. Prescription editing is signified by cursor movement off the command line. As the program was originally designed, the Data-entry completion variable by itself is not sufficient since it does not ensure that the cursor is located on the command line. Under the right circumstances, the data-entry phase can be exited before all edit changes are made on the screen.

The keyboard handler parses the mode and energy level specified by the operator and places an encoded result in another shared variable, the 2-byte mode/energy offset (MEOS) variable. The low-order byte of this variable is used by another task (Hand) to set the collimator/turntable to the proper position for the selected mode/energy. The high-order byte of the MEOS variable is used by Datent to set several operating parameters.

Initially, the data-entry process forces the operator to enter the mode and energy, except when the operator selects the photon mode, in which case the energy defaults to 25 MeV. The operator can later edit the mode and energy separately. If the keyboard handler sets the data-entry completion variable before the operator changes the data in MEOS, Datent will not detect the changes in MEOS since it has already exited and will not be reentered again. The upper collimator, on the other hand, is set to the position dictated by the low-order byte of MEOS by another concurrently running task (Hand) and can therefore be inconsistent with the parameters set in accordance with the information in the high-order byte of MEOS. The software appears to include no checks to detect such an incompatibility.

The first thing that Datent does when it is entered is to check whether the mode/energy has been set in MEOS. If so, it uses the high-order byte to index into a table of preset operating parameters and places them in the digital-to-analog output table. The contents of this output table are transferred to the digital-analog converter during the next clock cycle. Once the parameters are all set, Datent calls the subroutine Magnet, which sets the bending magnets. Figure 3 is a simplified pseudocode description of relevant parts of the software.

Setting the bending magnets takes about 8 seconds. Magnet calls a subroutine called Ptime to introduce a time delay. Since several magnets need to be set, Ptime is entered and exited several times. A flag to indicate that bending magnets are being set is initialized upon entry to the Magnet subroutine and cleared at the end of Ptime. Furthermore, Ptime checks a shared variable, set by the keyboard handler, that indicates the presence of any editing requests. If there are edits, then Ptime clears the bending magnet variable and exits to Magnet, which then exits to Datent. But the edit change variable is checked by Ptime only if the bending magnet flag is set. Since Ptime clears it during its first execution, any edits performed during each succeeding pass through Ptime will not be recognized. Thus, an edit change of the mode or energy, although reflected on the operator's screen and the mode/energy offset variable, will not be sensed by Datent so it can index the appropriate calibration tables for the machine parameters.

Recall that the Tyler error occurred when the operator made an entry indicating the mode/energy, went to the command line, then moved the cursor up to change the mode/energy, and returned to the command line all within 8 seconds. Since the magnet setting takes about 8 seconds and Magnet does not recognize edits after the first execution of Ptime, the editing had been completed by the return to Datent, which never detected that it had occurred. Part of the problem was fixed after the accident by clearing the bending-magnet variable at the end of Magnet (after all the magnets have been set) instead of at the end of Ptime.

But this was not the only problem. Upon exit from the Magnet subroutine, the data-entry subroutine (Datent) checks the data-entry completion variable. If it indicates that data entry is complete, Datent sets Tphase to 3 and Datent is not entered again. If it is not set, Datent leaves Tphase unchanged, which means it will eventually be rescheduled. But the data-entry completion variable only indicates that the cursor has been down to the command line, not that it is still there. A potential race condition is set up. To fix this, AECL introduced another shared variable controlled by the keyboard handler task that indicates the cursor is not positioned on the command line. If this variable is set, then prescription entry is still in progress and the value of Tphase is left unchanged.

Government and user response. The FDA does not approve each new medical device on the market: All medical devices go through a classification process that determines the level of FDA approval necessary. Medical accelerators follow a procedure called pre-market notification before commercial distribution. In this process, the firm must establish that the product is substantially equivalent in safety and effectiveness to a product already on the market. If that cannot be done to the FDA's satisfaction, a pre-market approval is required. For the Therac-25, the FDA required only a pre-market notification.

The agency is basically reactive to problems and requires manufacturers to report serious ones. Once a problem is identified in a radiation-emitting product, the FDA must approve the manufacturer's corrective action plan (CAP).

The first reports of the Tyler accidents came to the FDA from the state of Texas health department, and this triggered FDA action. The FDA investigation was well under way when AECL produced a medical device report to discuss the details of the radiation overexposures at Tyler. The FDA declared the Therac-25 defective under the Radiation Control for Health and Safety Act and ordered the firm to notify all purchasers, investigate the problem, determine a solution, and submit a corrective action plan for FDA approval.

The final CAP consisted of more than 20 changes to the system hardware and software, plus modifications to the system documentation and manuals. Some of these changes were unrelated to the specific accidents, but were improvements to the general machine safety. The full implementation of the CAP, including an extensive safety analysis, was not complete until more than two years after the Tyler accidents.

AECL made its accident report to the FDA on April 15, 1986. On that same date, AECL sent a letter to each Therac user recommending a temporary "fix" to the machine that would allow continued clinical use. The letter (shown in its complete form) read as follows:

SUBJECT: CHANGE IN OPERATING PROCEDURES FOR THE THERAC 25 LINEAR ACCELERATOR

Effective immediately, and until further notice, the key used for moving the cursor back through the prescription sequence (i.e., cursor "UP" inscribed with an upward pointing arrow) must not be used for editing or any other purpose.

To avoid accidental use of this key, the key cap must be removed and the switch contacts fixed in the open position with electrical tape or other insulating material. For assistance with the latter you should contact your local AECL service representative.

Disabling this key means that if any prescription data entered is incorrect then [an] "R" reset command must be used and the whole prescription reentered.

For those users of the Multiport option, it also means that editing of dose rate, dose, and time will not be possible between ports.

On May 2, 1986, the FDA declared the Therac defective, demanded a CAP, and required renotification of all the Therac customers. In the letter from the FDA to AECL, the director of compliance, Center for Devices and Radiological Health, wrote

We have reviewed Mr. Downs' April 15 letter to purchasers and have concluded that it does not satisfy the requirements for notification to purchasers of a defect in an electronic product. Specifically, it does not describe the defect nor the hazards associated with it. The letter does not provide any reason for disabling the cursor key and the tone is not commensurate with the urgency for doing so. In fact, the letter implies the inconvenience to operators outweighs the need to disable the key. We request that you immediately renotify purchasers.

AECL promptly made a new notice to users and also requested an extension to produce a CAP. The FDA granted this request.

About this time, the Therac-25 users created a user group and held their first meeting at the annual conference of the American Association of Physicists in Medicine. At the meeting, users discussed the Tyler accident and heard an AECL representative present the company's plans for responding to it. AECL promised to send a letter to all users detailing the CAP.

Several users described additional hardware safety features that they had added to their own machines to provide additional protection. An interlock (that checked gun current values), which the Vancouver clinic had previously added to its Therac-25, was labeled as redundant by AECL. The users disagreed. There were further discussions of poor design and other problems that caused 10- to 30-percent underdosing in both modes.

The meeting notes said

. . . there was a general complaint by all users present about the lack of information propagation. The users were not happy about receiving incomplete information. The AECL representative countered by stating that AECL does not wish to spread rumors and that AECL has no policy to "keep things quiet." The consensus among the users was that an improvement was necessary.

After the first user group meeting, there were two user group newsletters. The first, dated fall 1986, contained letters from Still, the Kennestone physicist, who complained about what he considered to be eight major problems he had experienced with the Therac-25. These problems included poor screen-refresh subroutines that left trash and erroneous information on the operator console, and some tape-loading problems upon start-up, which he discovered involved the use of "phantom tables" to trigger the interlock system in the event of a load failure instead of using a check sum. He asked the question, "Is programming safety relying too much on the software interlock routines?" The second user group newsletter, in December 1986, further discussed the implications of the "phantom table" parameterization.

AECL produced the first CAP on June 13, 1986. It contained six items:

(1) Fix the software to eliminate the specific behavior leading to the Tyler problem.
(2) Modify the software sample-and-hold circuits to detect one pulse above a nonadjustable threshold. The software sample-and-hold circuit monitors the magnitude of each pulse from the ion chambers in the beam. Previously, three consecutive high readings were required to shut off the high-voltage circuits, which resulted in a shutdown time of 300 ms. The software modification results in a reading after each pulse, and a shutdown after a single high reading.
(3) Make Malfunctions 1 through 64 result in treatment *suspend* rather than *pause.*
(4) Add a new circuit, which only administrative staff can reset, to shut down the modulator if the sample-and-hold circuits detect a high pulse. This is functionally equivalent to the circuit described in item 2. However, a new circuit board is added that monitors the five sample-and-hold circuits. The new circuit detects ion-chamber signals above a fixed threshold and inhibits the trigger to the modulator after detecting a high pulse. This shuts down the beam independently of the software.
(5) Modify the software to limit editing keys to cursor up, backspace, and return.
(6) Modify the manuals to reflect the changes.

FDA internal memos describe their immediate concerns regarding the CAP. One memo suggests adding an independent circuit that "detects and shuts down the system when inappropriate outputs are detected," warnings about when ion chambers are saturated, and under-standable system error messages. Another memo questions "whether all possible hardware options have been investigated by the manufacturer to prevent any future inadvertent high exposure."

On July 23 the FDA officially responded to AECL's CAP submission. They conceptually agreed with the plan's direction but complained about the lack of specific information necessary to evaluate the plan, especially with regard to the software. The FDA requested a detailed description of the software- development procedures and documentation, along with a revised CAP to include revised the software setup table, and the software interlock interactions. The

FDA also made a very detailed request for a documented test plan, and detailed descriptions of the revised edit modes, the changes made to the software setup table, and the software interlock interactions. The FDA also made a very detailed request for a documented test plan.

AECL responded on September 26 with several documents describing the software and its modifications but no test plan. They explained how the Therac-25 software evolved from the Therac-6 software and stated that "no single test plan and report exists for the software since both hardware and software were tested and exercised separately and together over many years." AECL concluded that the current CAP improved "machine safety by many orders of magnitude and virtually eliminates the possibility of lethal doses as delivered in the Tyler incident."

An FDA internal memo dated October 20 commented on these AECL submissions, raising several concerns:

Unfortunately, the AECL response also seems to point out an apparent lack of documentation on software specifications and a software test plan.

. . . concerns include the question of previous knowledge of problems by AECL, the apparent paucity of software QA [quality assurance] at the manufacturing facility, and possible warnings and information dissemination to others of the generic type problems.

. . . As mentioned in my first review, there is some confusion on whether the manufacturer should have been aware of the software problems prior to the [accidental radiation overdoses] in Texas. AECL had received official notification of a lawsuit in November 1985 from a patient claiming accidental over-exposure from a Therac-25 in Marietta, Georgia. . . If knowledge of these software deficiencies were known beforehand, what would be the FDA's posture in this case?

. . . The materials submitted by the manufacturer have not been in sufficient detail and clarity to ensure an adequate software QA program currently exists. For example, a response has not been provided with respect to the software part of the CAP to the CDRH [FDA Center for Devices and Radiological Health] request for documentation on the revised requirements and specifications for the new software. In addition, an analysis has not been provided, as requested, on the interaction with other portions of the software to demonstrate the corrected software does not adversely affect other software functions.

The July 23 letter from the CDRH requested a documented test plan includ-ing several specific pieces of information identified in the letter. This request has been ignored up to this point by the manufacturer. Considering the ramifi-cations of the current software problem, changes in software QA attitudes are needed at AECL.

On October 30, the FDA responded to AECL's additional submissions, complaining about the lack of a detailed description of the accident and of sufficient detail in flow diagrams. Many specific questions addressed the vagueness of the AECL response and made it clear that additional CAP work must precede approval.

AECL, in response, created CAP Revision 1 on November 12. This CAP contained 12 new items under "software modifications," all (except for one cosmetic change) designed to eliminate potentially unsafe behavior. The submission also contained other relevant documents including a test plan.

The FDA responded to CAP Revision 1 on December 11. The FDA explained that the software modifications appeared to correct the specific deficiencies discovered as a result of the Tyler accidents. They agreed that the major items listed in CAP Revision 1 would improve the Therac's operation. However, the FDA required AECL to attend to several further system problems before CAP approval. AECL had proposed to retain treatment pause for some dose-rate and beam-tilt malfunctions. Since these are dosimetry system problems, the FDA considered them safety interlocks and believed treatment must be suspended for these malfunctions.

AECL also planned to retain the malfunction codes, but the FDA required better warnings for the operators. Furthermore, AECL had not planned on any quality assurance testing to ensure exact copying of software, but the FDA insisted on it. The FDA further requested assurances that rigorous testing would become a standard part of AECL's software-modification procedures:

We also expressed our concern that you did not intend to perform the protocol to future modifications to software. We believe that the rigorous testing must be performed each time a modification is made in order to ensure the modification does not adversely affect the safety of the system.

AECL was also asked to draw up an installation test plan to ensure both hardware and software changes perform as designed when installed.

AECL submitted CAP Revision 2 and supporting documentation on December 22, 1986. They changed the CAP to have dose malfunctions suspend treatment and included a plan for meaningful error messages and highlighted dose error messages. They also expanded diagrams of software modifications and expanded the test plan to cover hardware and software.

On January 26, 1987, AECL sent the FDA their "Component and Instal-lation Test Plan" and explained that their delays were due to the investigation of a new accident on January 17 at Yakima.

Yakima Valley Memorial Hospital, 1987. On Saturday, January 17, 1987, the second patient of the day was to be treated at the Yakima Valley Memorial Hospital for a carcinoma. This patient was to receive two film-verification exposures of 4 and 3 rads, plus a 79-rad photon treatment (for a total exposure of 86 rads).

Film was placed under the patient and 4 rads was administered with the collimator jaws opened to 22 x 18 cm. After the machine paused, the collimator jaws opened to 35 x 35 cm automatically, and the second exposure of 3 rads was administered. The machine paused again.

The operator entered the treatment room to remove the film and verify the patient's precise position. He used the hand control in the treatment room to rotate the turntable to the field-light position, a feature that let him check the machine's alignment with respect to the patient's body to verify proper beam position. The operator then either pressed the set button on the hand control or left the room and typed a set command at the console to return the turntable to the proper position for treatment; there is some confusion as to exactly what transpired. When he left the room, he forgot to remove the film from underneath the patient. The console displayed "beam ready," and the operator hit the "B" key to turn the beam on.

The beam came on but the console displayed no dose or dose rate. After 5 or 6 seconds, the unit shut down with a pause and displayed a message. The message "may have disappeared quickly"; the operator was unclear on this point. However, since the machine merely paused, he was able to push the "P" key to proceed with treatment.

The machine paused again, this time displaying "flatness" on the reason line. The operator heard the patient say something over the intercom, but couldn't understand him. He went into the room to speak with the patient, who reported "feeling a burning sensation" in the chest. The console displayed only the total dose of the two film exposures (7 rads) and nothing more.

Later in the day, the patient developed a skin burn over the entire treatment area. Four days later, the redness took on the striped pattern matching the slots in the blocking tray. The striped pattern was similar to the burn a year earlier at this hospital that had been attributed to "cause unknown."

AECL began an investigation, and users were told to confirm the turntable position visually before turning on the beam. All tests run by the AECL engineers indicated that the machine was working perfectly. From the information gathered to that point, it was suspected that the electron beam had come on when the turntable was in the field-light position. But the investigators could not reproduce the fault condition that produced the overdose.

On the following Thursday, AECL sent an engineer from Ottawa to investigate. The hospital physicist had, in the meantime, run some tests with film. He placed a film in the Therac's beam and ran two exposures of X-ray parameters with the turntable in field-light position. The film appeared to match the film that was left (by mistake) under the patient during the accident.

After a week of checking the hardware, AECL determined that the "incorrect machine operation was probably not caused by hardware alone." After checking the software, AECL discovered a flaw (described in the next section) that could explain the erroneous behavior. The coding problems explaining this accident differ from those associated with the Tyler accidents.

AECL's preliminary dose measurements indicated that the dose delivered under these conditions - that is, when the turntable was in the field-light position - was on the order of 4,000 to 5,000 rads. After two attempts, the patient could have received 8,000 to 10,000 instead of the 86 rads prescribed. AECL again called users on January 26 (nine days after the accident) and gave them detailed instructions on how to avoid this problem. In an FDA internal report on the accident, an AECL quality assurance manager investigating the problem is quoted as saying that the software and hardware changes to be retrofitted following the Tyler accident nine months earlier (but which had not yet been installed) would have prevented the Yakima accident.

The patient died in April from complications related to the overdose. He had been suffering from a terminal form of cancer prior to the radiation overdose, but survivors initiated lawsuits alleging that he died sooner than he would have and endured unnecessary pain and suffering due to the overdose. The suit was settled out of court.

The Yakima software problem. The software problem for the second Yakima accident is fairly well established and different from that implicated in the Tyler accidents. There is no way to determine what particular software design errors were related to the Kennestone, Hamilton, and first Yakima accidents. Given the unsafe programming practices exhibited in the code, it is possible that unknown race conditions or errors could have been responsible. There is speculation, however, that the Hamilton accident was the same as this second Yakima overdose. In a report of a conference call on January 26, 1987, between the AECL quality assurance manager and Ed Miller of the FDA discussing the Yakima accident, Miller notes

This situation probably occurred in the Hamilton, Ontario, accident a couple of years ago. It was not discovered at that time and the cause was attributed to intermittent interlock failure. The subsequent recall of the multiple microswitch logic network did not really solve the problem.

The second Yakima accident was again attributed to a type of race condition in the software - this one allowed the device to be activated in an error setting (a "failure" of a software interlock). The Tyler accidents were related to problems in the data-entry routines that allowed the code to proceed to Set-Up Test before the full prescription had been entered and acted upon. The Yakima accident involves problems encountered later in the logic after the treatment monitor Treat reaches Set-Up Test.

The Therac-25's field-light feature permits very precise positioning of the patient for treatment. The operator can control the Therac-25 right at the treatment site using a small hand control offering certain limited functions for patient setup, including setting gantry, collimator, and table motions.

Normally, the operator enters all the prescription data at the console (outside the treatment room) before the final setup of all machine parameters is completed in the treatment room. This gives rise to an "unverified" condition at the console. The operator then completes the patient setup in the treatment room, and all relevant parameters now "verify." The console displays the message "Press set button" while the turntable is in the field-light position. The operator now presses the set button on the hand control or types "set" at the console. That should set the collimator to the proper position for treatment.

In the software, after the prescription is entered and verified by the Datent routine, the control variable Tphase is changed so that the Set-Up Test routine is entered (see Figure 4 Yakima software flaw). Every pass through the Set-Up Test routine increments the upper collimator position check, a shared variable called Class3. If Class3 is nonzero, there is an inconsistency and treatment should not proceed. A zero value for Class3 indicates that the relevant parameters are consistent with treatment, and the beam is not inhibited.

After setting the Class3 variable, Set-Up Test next checks for any malfunctions in the system by checking another shared variable (set by a routine that actually handles the interlock checking) called F$mal to see if it has a nonzero value. A nonzero value in F$mal indicates that the machine is not ready for treatment, and the Set-Up Test subroutine is rescheduled. When F$mal is zero (indicating that everything is ready for treatment), the Set-Up Test subroutine sets the Tphase variable equal to 2, which results in next scheduling the Set-Up Done subroutine, and the treatment is allowed to continue.

The actual interlock checking is performed by a concurrent Housekeeper task (Hkeper). The upper collimator position check is performed by a subroutine of Hkeper called Lmtchk (analog/digital limit checking). Lmtchk first checks the Class3 variable. If Class3 contains a nonzero value, Lmtchk calls the Check Collimator (Chkcol) subroutine. If Class3 contains zero, Chkcol is bypassed and the upper collimator position check is not performed. The Chkcol subroutine sets or resets bit 9 of the F$mal shared variable, depending on the position of the upper collimator (which in turn is checked by the Set-Up Test subroutine of Datent so it can decide whether to reschedule itself or proceed to Set-Up Done).

During machine setup, Set-Up Test will be executed several hundred times since it reschedules itself waiting for other events to occur. In the code, the Class3 variable is incremented by one in each pass through Set-Up Test. Since the Class3 variable is 1 byte, it can only contain a maximum value of 255 decimal. Thus, on every 256th pass through the Set-Up Test code, the variable overflows and has a zero value. That means that on every 256th pass through Set-Up Test, the upper collimator will not be checked and an upper collimator fault will not be detected.

The overexposure occurred when the operator hit the "set" button at the precise moment that Class3 rolled over to zero. Thus Chkcol was not executed, and F$mal was not set to indicate the upper collimator was still in field-light position. The software turned on the full 25 MeV without the target in place and without scanning. A highly concentrated electron beam resulted, which was scattered and deflected by the stainless steel mirror that was in the path.

AECL described the technical "fix" implemented for this software flaw as simple: The program is changed so that the Class3 variable is set to some fixed nonzero value each time through Set-Up Test instead of being incremented.

Manufacturer, government, and user response. On February 3, 1987, after interaction with the FDA and others, including the user group, AECL announced to its customers

- a new software release to correct both the Tyler and Yakima software problems,
- a hardware single-pulse shutdown circuit,
- a turntable potentiometer to independently monitor turntable position, and
- a hardware turntable interlock circuit.

The second item, a hardware single-pulse shutdown circuit, essentially acts as a hardware interlock to prevent overdosing by detecting an unsafe level of radiation and halting beam output after one pulse of high energy and current. This provides an independent safety mechanism to protect against a wide range of potential hardware failures and software errors. The turntable potentiometer was the safety device recommended by several groups, including the CRPB, after the Hamilton accident.

After the second Yakima accident, the FDA became concerned that the use of the Therac-25 during the CAP process, even with AECL's interim operating instructions, involved too much risk to patients. The FDA concluded that the accidents had demonstrated that the software alone cannot be replied upon to assure safe operation of the machine. In a February 18, 1987 internal FDA memorandum, the director of the Division of Radiological Products wrote the following:

It is impossible for CDRH to find all potential failure modes and conditions of the software. AECL has indicated the "simple software fix" will correct the turntable position problem displayed at Yakima. We have not yet had the opportunity to evaluate that modification. Even if it does, based upon past history, I am not convinced that there are not other software glitches that could result in serious injury.

For example, we are aware that AECL issued a user's bulletin January 21 reminding users of the proper procedure to follow if editing of prescription parameter is desired after entering the "B" (beam on) code but before the CR [carriage return] is pressed. It seems that the normal edit keys (down arrow, right arrow, or line feed) will be interpreted as a CR and initiate exposure. One must use either the backspace or left arrow key to edit.

We are also aware that if the dose entered into the prescription tables is below some preset value, the system will default to a phantom table value unbeknownst to the operator. This problem is supposedly being addressed in proposed interim revision 7A, although we are unaware of the details.

We are in the position of saying that the proposed CAP can reasonably be expected to correct the deficiencies for which they were developed (Tyler). We cannot say that we are [reasonably] confident about the safety of the entire system to prevent or minimize exposure from other fault conditions.

On February 6, 1987, Miller of the FDA called Pavel Dvorak of Canada's Health and Welfare to advise him that the FDA would recommend all Therac-25s be shut down until permanent modifications could be made. According to Miller's notes on the phone call, Dvorak agreed and indicated that they would coordinate their actions with the FDA.

On February 10, 1987, the FDA gave a Notice of Adverse Findings to AECL declaring the Therac-25 to be defective under US law. In part, the letter to AECL reads:

In January 1987, CDRH was advised of another accidental radiation occurrence in Yakima, which was attributed to a second software defect related to the "Set" command. In addition, the CDRH has become aware of at least two other software features that provide potential for unnecessary or inadvertent patient exposure. One of these is related to the method of editing the prescription after the "B" command is entered and the other is the calling of phantom tables when low doses are prescribed.

Further review of the circumstances surrounding the accidental radiation occurrences and the potential for other such incidents has led us to conclude that in addition to the items in your proposed corrective action plan, hardware interlocking of the turntable to insure its proper position prior to beam activation appears to be necessary to enhance system safety and to correct the Therac-25 defect. Therefore, the corrective action plan as currently proposed is insufficient and must be amended to include turntable interlocking and corrections for the three software problems mentioned above.

Without these corrections, CDRH has concluded that the consequences of the defects represents a significant potential risk of serious injury even if the Therac-25 is operated in accordance with your interim operating instructions. CDRH, therefore, requests that AECL immediately notify all purchasers and recommend that use of the device on patients for routine therapy be discontinued until such time that an amended corrective action plan approved by CDRH is fully completed. You may also advise purchasers that if the need for an individual patient treatment outweighs the potential risk, then extreme caution and strict adherence to operating safety procedures must be exercised.

At the same time, the Health Protection Branch of the Canadian government instructed AECL to recommend to all users in Canada that they discontinue the operation of the Therac-25 until "the company can complete an exhaustive analysis of the design and operation of the safety systems employed for patient and operator protection." AECL was told that the letter to the users should include information on how the users can operate the equipment safely in the event that they must continue with patient treatment. If AECL could not provide information that would guarantee safe operation of the equipment, AECL was requested to inform the users that they cannot operate the equipment safely. AECL complied by letters dated February 20, 1987, to Therac-25 purchasers. This recommendation to discontinue use of the Therac-25 was to last until August 1987.

On March 5, 1987, AECL issued CAP Revision 3, which was a CAP for both the Tyler and Yakima accidents. It contained a few additions to the Revision 2 modifications, notably

- changes to the software to eliminate the behavior leading to the latest Yakima accident,
- four additional software functional modifications to improve safety, and
- a turntable position interlock in the software.

In their response on April 9, the FDA noted that in the appendix under "turntable position interlock circuit" the descriptions were wrong. AECL had indicated "high" signals where "low" signals were called for and vice versa. The FDA also questioned the reliability of the turntable potentiometer design and asked whether the backspace key could still act as a carriage return in the edit mode. They requested a detailed description of the software portion of the single-pulse shutdown and a block diagram to demonstrate the PRF (pulse repetition frequency) generator, modulator, and associated interlocks.

AECL responded on April 13 with an update on the Therac CAP status and a schedule of the nine action items pressed by the users at a user group meeting in March. This unique and highly productive meeting provided an unusual opportunity to involve the users in the CAP evaluation process. It brought together all concerned parties in one place so that they could decide on and approve a course of action as quickly as possible. The attendees included representatives from the manufacturer (AECL); all users, including their technical and legal staffs; the US FDA; the Canadian BRMD; the Canadian Atomic Energy Control Board; the Province of Ontario; and the Radiation Regulations Committee of the Canadian Association of Physicists.

According to Symonds of the BRMD, this meeting was very important to the resolution of the problems since the regulators, users, and the manufacturer arrived at a consensus in one day.

At this second users meeting, the participants carefully reviewed all the six known major Therac-25 accidents and discussed the elements of the CAP along with possible additional modifications. They came up with a prioritized list of modifications that they wanted included in the CAP and expressed concerns about the lack of independent software evaluation and the lack of a hard-copy audit trail to assist in diagnosing faults.

The AECL representative, who was the quality assurance manager, responded that tests had been done on the CAP changes, but that the tests were not documented, and independent evaluation of the software "might not be possible." He claimed that two outside experts had reviewed the software, but he could not provide their names. In response to user requests for a hard-copy audit trail and access to source code, he explained that memory limitations would not permit including an audit option, and source code would not be made available to users.

On May 1, AECL issued CAP Revision 4 as a result of the FDA comments and users meeting input. The FDA response on May 26 approved the CAP subject to submission of the final test plan results and an independent safety analysis, distribution of the draft revised manual to customers, and completion of the CAP by June 30, 1987. The FDA concluded by rating this a Class I recall: a recall in which there is a reasonable probability that the use of or exposure to a violative product will cause serious adverse health consequences or death.[5]

AECL sent more supporting documentation to the FDA on June 5, 1987, including the CAP test plan, a draft operator's manual, and the draft of the new safety analysis (described in the sidebar Safety analysis of the Therac-25). The safety analysis revealed four potentially hazardous subsystems that were not covered by CAP Revision 4:

(1) electron-beam scanning,
(2) electron-energy selection,
(3) beam shutoff, and
(4) calibration and/or steering.

AECL planned a fifth revision of the CAP to include the testing and safety analysis results.

Referring to the test plan at this, the final stage of the CAP process, an FDA reviewer said

Amazingly, the test data presented to show that the software changes to handle the edit problems in the Therac-25 are appropriate prove the exact opposite result. A review of the data table in the test results indicates that the final beam type and energy (edit change) [have] no effect on the initial beam type and energy. I can only assume that either the fix is not right or the data was entered incorrectly. The manufacturer should be admonished for this error. Where is the QC [quality control] review for the test program? AECL must: (1) clarify this situation, (2) change the test protocol to prevent this type of error from occurring, and (3) set up appropriate QC control on data review.

A further FDA memo said the AECL quality assurance manager

. . . could not give an explanation and will check into the circumstances. He subsequently called back and verified that the technician completed the form incorrectly. Correct operation was witnessed by himself and others. They will repeat and send us the correct data sheet.

At the American Association of Physicists in Medicine meeting in July 1987, a third user group meeting was held. The AECL representative gave the status of CAP Revision 5. He explained that the FDA had given verbal approval and he expected full implementation by the end of August 1987. He reviewed and commented on the prioritized concerns of the last meeting. AECL had included in the CAP three of the user-requested hardware changes. Changes to tape-load error messages and check sums on the load data would wait until after the CAP was done.

Two user-requested hardware modifications had not been included in the CAP. One of these, a push-button energy and selection mode switch, AECL would work on after completing the CAP, the quality assurance manager said. The other, a fixed ion chamber with dose/pulse monitoring, was being installed at Yakima, had already been installed by Halifax on their own, and would be an option for other clinics. Software documentation was described as a lower priority task that needed definition and would not be available to the FDA in any form for more than a year.

On July 6, 1987, AECL sent a letter to all users to inform them of the FDA's verbal approval of the CAP and delineated how AECL would proceed. On July 21, 1987, AECL issued the fifth and final CAP revision. The major features of the final CAP are as follows:

- All interruptions related to the dosimetry system will go to a treatment suspend, not a treatment pause. Operators will not be allowed to restart the machine without reentering all parameters.
- A software single-pulse shutdown will be added.
- An independent hardware single-pulse shutdown will be added.
- Monitoring logic for turntable position will be improved to ensure that the turntable is in one of the three legal positions.
- A potentiometer will be added to the turntable. It will provide a visible signal of position that operators will use to monitor exact turntable location.
- Interlocking with the 270-degree bending magnet will be added to ensure that the target and beam flattener are in position if the X-ray mode is selected.
- Beam on will be prevented if the turntable is in the field-light or an intermediate position.
- Cryptic malfunction messages will be replaced with meaningful messages and highlighted dose-rate messages.
- Editing keys will be limited to cursor up, backspace, and return. All other keys will be inoperative.

- A motion-enable foot switch will be added, which the operator must hold closed during movement of certain parts of the machine to prevent unwanted motions when the operator is not in control (a type of "dead man's switch").
- Twenty-three other changes to the software to improve its operation and reliability, including disabling of unused keys, changing the operation of the set and reset commands, preventing copying of the control program on site, changing the way various detected hardware faults are handled, eliminating errors in the software that were detected during the review process, adding several additional software interlocks, disallowing changing to the service mode while a treatment is in progress, and adding meaningful error messages.
- The known software problems associated with the Tyler and Yakima accidents will be fixed.
- The manuals will be fixed to reflect the changes.

In a 1987 paper, Miller, director of the Division of Standards Enforcement, CDRH, wrote about the lessons learned from the Therac-25 experiences.[6] The first was the importance of safe versus "user-friendly" operator interfaces - in other words, making the machine as easy as possible to use may conflict with safety goals. The second is the importance of providing fail-safe designs:

The second lesson is that for complex interrupt-driven software, timing is of critical importance. In both of these situations, operator action within very narrow time-frame windows was necessary for the accidents to occur. It is unlikely that software testing will discover all possible errors that involve operator intervention at precise time frames during software operation. These machines, for example, have been exercised for thousands of hours in the factory and in the hospitals without accident. Therefore, one must provide for prevention of catastrophic results of failures when they do occur.

I, for one, will not be surprised if other software errors appear with this or other equipment in the future.

Miller concluded the paper with

FDA has performed extensive review of the Therac-25 software and hardware safety systems. We cannot say with absolute certainty that all software problems that might result in improper dose have been found and eliminated. However, we are confident that the hardware and software safety features recently added will prevent future catastrophic consequences of failure.

Lessons learned

Often, it takes an accident to alert people to the dangers involved in technology. A medical physicist wrote about the Therac-25 accidents:

In the past decade or two, the medical accelerator "industry" has become perhaps a little complacent about safety. We have assumed that the manufacturers have all kinds of safety design experience since they've been in the business a long time. We know that there are many safety codes, guides, and regulations to guide them and we have been reassured by the hitherto excellent record of these machines. Except for a few incidents in the 1960s (e.g., at Hammersmith, Hamburg) the use of medical accelerators has been remarkably free of serious radiation accidents until now. Perhaps, though, we have been spoiled by this success.[1]

Accidents are seldom simple - they usually involve a complex web of interacting events with multiple contributing technical, human, and organizational factors. One of the serious mistakes that led to the multiple Therac-25 accidents was the tendency to believe that the cause of an accident had been determined (for example, a microswitch failure in the Hamilton accident) without adequate evidence to come to this conclusion and without looking at all possible contributing factors. Another mistake was the assumption that fixing a particular error (eliminating the current software bug) would prevent future accidents. There is always another software bug.

Accidents are often blamed on a single cause like human error. But virtually all factors involved in accidents can be labeled human error, except perhaps for hardware wear-out failures. Even such hardware failures could be attributed to human error (for example, the designer's failure to provide adequate redundancy or the failure of operational personnel to properly maintain or replace parts): Concluding that an accident was the result of human error is not very helpful or meaningful.

It is nearly as useless to ascribe the cause of an accident to a computer error or a software error. Certainly software was involved in the Therac-25 accidents, but it was only one contributing factor. If we assign software error as the cause of the Therac-25 accidents, we are forced to conclude that the only way to prevent such accidents in the future is to build perfect software that will never behave in an unexpected or undesired way under any circumstances (which is clearly impossible) or not to use software at all in these types of systems. Both conclusions are overly pessimistic.

We must approach the problem of accidents in complex systems from a system-engineering point of view and consider all possible contributing factors. For the Therac-25 accidents, contributing factors included

- management inadequacies and lack of procedures for following through on all reported incidents,
- overconfidence in the software and removal of hardware interlocks (making the software into a single point of failure that could lead to an accident),
- presumably less-than-acceptable software-engineering practices, and
- unrealistic risk assessments along with overconfidence in the results of these assessments.

The exact same accident may not happen a second time, but if we examine and try to ameliorate the contributing factors to the accidents we have had, we may be able to prevent different accidents in the future. In the following sections, we present what we feel are important lessons learned from the Therac-25. You may draw different or additional conclusions.

System engineering. A common mistake in engineering, in this case and many others, is to put too much confidence in software. Nonsoftware professionals seem to feel that software will not or cannot fail; this attitude leads to complacency and overreliance on computerized functions.

Although software is not subject to random wear-out failures like hardware, software design errors are much harder to find and eliminate. Furthermore, hardware failure modes are generally much more limited, so building protection against them is usually easier. A lesson to be learned from the Therac-25 accidents is not to remove standard hardware interlocks when adding computer control.

Hardware backups, interlocks, and other safety devices are currently being replaced by software in many different types of systems, including commercial aircraft, nuclear power plants, and weapon systems. Where the hardware interlocks are still used, they are often controlled by software. Designing any dangerous system in such a way that one failure can lead to an accident violates basic system-engineering principles. In this respect, software needs to be treated as a single component. Software should not be assigned sole responsibility for safety, and systems should not be designed such that a single software error or software-engineering error can be catastrophic.

A related tendency among engineers is to ignore software. The first safety analysis on the Therac-25 did not include software (although nearly full responsibility for safety rested on the software). When problems started occurring, investigators assumed that hardware was the cause and focused only on the hardware. Investigation of software's possible contribution to an accident should not be the last avenue explored after all other possible explanations are eliminated.

In fact, a software error can always be attributed to a transient hardware failure, since software (in these types of process-control systems) reads and issues commands to actuators. Without a thorough investigation (and without on-line monitoring or audit trails that save internal state information), it is not possible to determine whether the sensor provided the wrong information, the software provided an incorrect command, or the actuator had a transient failure and did the wrong thing on its own. In the Hamilton accident, a transient microswitch failure was assumed to be the cause, even though the engineers were unable to reproduce the failure or find anything wrong with the microswitch.

Patient reactions were the only real indications of the seriousness of the problems with the Therac-25. There were no independent checks that the software was operating correctly (including software checks). Such verification cannot be assigned to operators without providing them with some means of detecting errors. The Therac-25 software "lied" to the operators, and the machine itself could not detect that a massive overdose had occurred. The Therac-25 ion chambers could not handle the high density of ionization from the unscanned electron beam at high-beam current; they thus became saturated and gave an indication of a low dosage. Engineers need to design for the worst case.

Every company building safety-critical systems should have audit trails and incident-analysis procedures that they apply whenever they find any hint of a problem that might lead to an accident. The first phone call by Still should have led to an extensive investigation of the events at Kennestone. Certainly, learning about the first lawsuit should have triggered an immediate response. Although hazard logging and tracking is required in the standards for safety-critical military projects, it is less common in nonmilitary projects. Every company building hazardous equipment should have hazard logging and tracking as well as incident reporting and analysis as parts of its quality control procedures. Such follow-up and tracking will not only help prevent accidents, but will easily pay for themselves in reduced insurance rates and reasonable settlement of lawsuits when they do occur.

Finally, overreliance on the numerical output of safety analyses is unwise. The arguments over whether very low probabilities are meaningful with respect to safety are too extensive to summarize here. But, at the least, a healthy skepticism is in order. The claim that safety had been increased five orders of magnitude as a result of the microswitch fix after the Hamilton accident seems hard to justify. Perhaps it was based on the probability of failure of the microswitch (typically 10^5) ANDed with the other interlocks. The problem with all such analyses is that they exclude aspects of the problem (in this case, software) that are difficult to quantify but which may have a larger impact on safety than the quantifiable factors that are included.

Although management and regulatory agencies often press engineers to obtain such numbers, engineers should insist that any risk assessment numbers used are in fact meaningful and that statistics of this sort are treated with caution. In our enthusiasm to provide measurements, we should not attempt to measure the unmeasurable. William Ruckelshaus, two-time head of the US Environmental Protection Agency, cautioned that "risk assessment data can be like the captured spy; if you torture it long enough, it will tell you anything you want to know."[7] E.A. Ryder of the British Health and Safety Executive has written that the numbers game in risk assessment "should only be played in private between consenting adults, as it is too easy to be misinterpreted."[8]

Software engineering. The Therac-25 accidents were fairly unique in having software coding errors involved -- most computer-related accidents have not involved coding errors but rather errors in the software requirements such as omissions and mishandled environmental conditions and system states. Although using good basic software-engineering practices will not prevent all software errors, it is certainly required as a minimum. Some companies introducing software into their systems for the first time do not take software engineering as seriously as they should. Basic software-engineering principles that apparently were violated with the Therac-25 include:

- Documentation should not be an afterthought.
- Software quality assurance practices and standards should be established.
- Designs should be kept simple.
- Ways to get information about errors -- for example, software audit trails -- should be designed into the software from the beginning.
- The software should be subjected to extensive testing and formal analysis at the module and software level; system testing alone is not adequate.

In addition, special safety-analysis and design procedures must be incorporated into safety-critical software projects. Safety must be built into software, and, in addition, safety must be assured at the system level despite software errors.[9,10] The Therac-20 contained the same software error implicated in the Tyler deaths, but the machine included hardware interlocks that mitigated its consequences. Protection against software errors can also be built into the software itself.

Furthermore, important lessons about software reuse can be found here. A naive assumption is often made that reusing software or using commercial off-the-shelf software increases safety because the software has been exercised extensively. Reusing software modules does not guarantee safety in the new system to which they are transferred and sometimes leads to awkward and dangerous designs. Safety is a quality of the system in which the software is used; it is not a quality of the software itself. Rewriting the entire software to get a clean and simple design may be safer in many cases.

Taking a couple of programming courses or programming a home computer does not qualify anyone to produce safety-critical software. Although certification of software engineers is not yet required, more events like those associated with the Therac-25 will make such certification inevitable. There is activity in Britain to specify required courses for those working on critical software. Any engineer is not automatically qualified to be a software engineer -- an extensive program of study and experience is required. Safety-critical software engineering requires training and experience in addition to that required for noncritical software.

Although the user interface of the Therac-25 has attracted a lot of attention, it was really a side issue in the accidents. Certainly, it could have been improved, like many other aspects of this software. Either software engineers need better training in interface design, or more input is needed from human factors engineers. There also needs to be greater recognition of potential conflicts between user-friendly interfaces and safety. One goal of interface design is to make the interface as easy as possible for the operator to use. But in the Therac-25, some design features (for example, not requiring the operator to reenter patient prescriptions after mistakes) and later changes (allowing a carriage return to indicate that information has been entered correctly) enhanced usability at the expense of safety.

Finally, not only must safety be considered in the initial design of the software and it operator interface, but the reasons for design decisions should be recorded so that decisions are not inadvertently undone in future modifications.

User and government oversight and standards. Once the FDA got involved in the Therac-25, their response was impressive, especially considering how little experience they had with similar problems in computerized medical devices. Since the Therac-25 events, the FDA has moved to improve the reporting system and to augment their procedures and guidelines to include software. The problem of deciding when to forbid the use of medical devices that are also saving lives has no simple answer and involves ethical and political issues that cannot be answered by science or engineering alone. However, at the least, better procedures are certainly required for reporting problems to the FDA and to users.

The issues involved in regulation of risky technology are complex. Overly strict standards can inhibit progress, require techniques behind the state of the art, and transfer responsibility from the manufacturer to the government. The fixing of responsibility requires a delicate balance. Someone must represent the public's needs, which may be subsumed by a company's desire for profits. On the other hand, standards can have the undesirable effect of limiting the safety efforts and investment of companies that feel their legal and moral responsibilities are fulfilled if they follow the standards.

Some of the most effective standards and efforts for safety come from users. Manufacturers have more incentive to satisfy customers than to satisfy government agencies. The American Association of Physicists in Medicine established a task group to work on problems associated with computers in radiation therapy in 1979, long before the Therac-25 problems began. The accidents intensified these efforts, and the association is developing user-written standards. A report by J.A. Rawlinson of the Ontario Cancer Institute attempted to define the physicist's role in assuring adequate safety in medical accelerators:

> We could continue our traditional role, which has been to provide input to the manufacturer on safety issues but to leave the major safety design decisions to the manufacturer. We can provide this input through a number of mechanisms. . . These include participation in standards organizations such as the IEC [Interna-tional Electrotechnical Commission], in professional association groups . . . and in accelerator user groups such as the Therac-25 user group. It includes also making use of the Problem Reporting Program for Radiation Therapy Devices . . . and it includes consultation in the drafting of the government safety regulations. Each of these if pursued vigorously will go a long way to improving safety. It is debatable however whether these actions would be sufficient to prevent a future series of accidents.

Perhaps what is needed in addition is a mechanism by which the safety of any new model of accelerator is assessed independently of the manufacturer. This task could be done by the individual physicist at the time of acceptance of a new machine. Indeed many users already test at least the operation of safety interlocks during commissioning. Few however have the time or resources to conduct a comprehensive assessment of safety design.

A more effective approach might be to require that prior to the use of a new type of accelerator in a particular jurisdiction, an independent safety analysis is made by a panel (including but not limited to medical physicists). Such a panel could be established within or without a regulatory framework.[1]

It is clear that users need to be involved. It was users who found the problems with the Therac-25 and forced AECL to respond. The process of fixing the Therac-25 was user driven -- the manufacturer was slow to respond. The Therac-25 user group meetings were, according to participants, important to the resolution of the problems. But if users are to be involved, then they must be provided with information and the ability to perform this function. Manufacturers need to understand that the adversarial approach and the attempt to keep government agencies and users in the dark about problems will not be to their benefit in the long run.

The US Air Force has one of the most extensive programs to inform users. Contractors who build space systems for the Air Force must provide an Accident Risk Assessment Report (AFAR) to system users and operators that describes the hazardous subsystems and operations associated with that system and its interfaces. The AFAR also comprehensively identifies and evaluates the system's accident risks; provides a means of substantiating compliance with safety requirements; summarizes all system-safety analyses and testing performed on each system and subsystem; and identifies design and operating limits to be imposed on system components to preclude or minimize accidents that could cause injury or damage.

An interesting requirement in the Air Force AFAR is a record of all safety-related failures or accidents associated with system acceptance, test, and checkout, along with an assessment of the impact on flight and ground safety and action taken to prevent recurrence. The AFAR also must address failures, accidents, or incidents from previous missions of this system or other systems using similar hardware. All corrective action taken to prevent recurrence must be documented. The accident and correction history must be updated throughout the life of the system. If any design or operating parameters change after government approval, the AFAR must be updated to include all changes affecting safety.

Unfortunately, the Air Force program is not practical for commercial systems. However, government agencies might require manufacturers to provide similar information to users. If required for everyone, competitive pressures to withhold information might be lessened. Manufacturers might find that providing such information actually increases customer loyalty and confidence. An emphasis on safety can be turned into a competitive advantage.

Most previous accounts of the Therac-25 accidents blamed them on a software error and stopped there. This is not very useful and, in fact, can be misleading and dangerous: If we are to prevent such accidents in the future, we must dig deeper. 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.

Accidents are unlikely to occur in exactly the same way again. If we patch only the symptoms and ignore the deeper underlying causes or we fix only the specific cause of one accident, we are unlikely to prevent or mitigate future accidents. The series of accidents involving the Therac-25 is a good example of exactly this problem: 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.

Although these accidents occurred in software controlling medical devices, the lessons apply to all types of systems where computers control dangerous devices. In our experience, the same types of mistakes are being made in nonmedical systems. We must learn from our mistakes so we do not repeat them.

Acknowledgments

Ed Miller of the FDA was especially helpful, both in providing information to be included in this article and in reviewing and commenting on the final version. Gordon Symonds of the Canadian Government Health Protection Branch also reviewed and commented on a draft of the article. Finally, the referees, several of whom were apparently intimately involved in some of the accidents, were also very helpful in providing additional information about the accidents.

References

The information in this article was gathered from official FDA documents and internal memos, lawsuit depositions, letters, and various other sources that are not publicly available. Computer does not provide references to documents that are unavailable to the public.

1. J.A. Rawlinson, "Report on the Therac-25," OCTRF/OCI Physicists Meeting, Kingston, Ont., Canada, May 7, 1987.

2. F. Houston, "What Do the Simple Folk Do?: Software Safety in the Cottage Industry," IEEE Computers in Medicine Conf., 1985.

3. C.A. Bowsher, "Medical Devices: The Public Health at Risk," US Gov't Accounting Office Report GAO/T-PEMD-90-2, 046987/139922, 1990.

4. M. Kivel, ed., Radiological Health Bulletin, Vol. XX, No. 8, US Federal Food and Drug Administration, Dec. 1986.

5. Medical Device Recalls, Examination of Selected Cases, GAO/PEMD-90-6, 1989.

6. E. Miller, "The Therac-25 Experience," Proc. Conf. State Radiation Control Program Directors, 1987.

7. W.D. Ruckelshaus, "Risk in a Free Society," Risk Analysis, Vol. 4, No. 3, 1984, pp. 157-162.

8. E.A. Ryder, "The Control of Major Hazards: The Advisory Committee's Third and Final Report," Transcript of Conf. European Major Hazards, Oyez Scientific and Technical Services and Authors, London, 1984.

9. N.G. Leveson, "Software Safety: Why, What, and How," ACM Computing Surveys, Vol. 18, No. 2, June 1986, pp. 25-69.

10. N.G. Leveson, "Software Safety in Embedded Computer Systems," Comm. ACM, Feb. 1991, pp. 34-46.

Nancy G. Leveson is Boeing professor of Computer Science and Engineering at the University of Washington. Previously, she was a professor in the Information and Computer Science Department at the University of California, Irvine. Her research interests are software safety and reliability, including software hazard analysis, requirements specification and analysis, design for safety, and verification of safety. She consults worldwide for industry and government on safety-critical systems.

Leveson received a BA in mathematics, an MS in operations research, and a PhD in computer science, all from the University of California at Los Angeles. She is the editor in chief of *IEEE Transactions on Software Engineering* and a member of the board of directors of the Computing Research Association.

Clark S. Turner is seeking his PhD in the Information and Computer Science Department at the University of California, Irvine, studying under Nancy Leveson. He is also an attorney admitted to practice in California, New York, and Massachusetts. His interests include risk analysis of safety-critical software systems and legal liability issues involving unsafe software systems.

Turner received a BS in mathematics from King's College in Pennsylvania, an MA in mathematics from Pennsylvania State University, a JD from the University of Maine, and an MS in computer science from the University of California, Irvine.

Readers can contact Leveson at the Department of Computer Science and Engineering, FR-35, University of Washington, Seattle, WA 98195, e-mail leveson@cs.washington.edu; or Turner at the Information and Computer Science Department, University of California, Irvine, Irvine, CA 92717, e-mail turner@ics.uci.edu.

Who Let The Worms Out?

A rogue program infected 10 percent of the Internet in 1988. Could such an outbreak happen again?

Its one of the biggest Internet disasters of all time, yet many of todays technology consultants dont remember the online carnage.

Long before virus outbreaks like NakedWife, Kournikova, Melissa and ILOVEYOU, there was the infamous Morris Worm.

Flash back to Nov. 2, 1988. The Los Angeles Dodgers had just won the World Series, Ronald Reagan was about to exit the White House, and a shy programmer named Robert T. Morris was set to unleash a digital plague that infected 10 percent of the Net.

Those closest to the case say Morris story should be required reading for aspiring security consultants, e-business partners and systems integrators alike.

> The Morris case involved a 99-line program written to infiltrate Digital VAX and Sun 3 systems. The so-called worm didnt contain any malicious code. Instead, Morris simply wanted to prove that he could use programs like sendmail to propagate a worm across the Internet.

Bad Code But when Morris released the program on the Internet, a design flaw caused the worm to reproduce faster than a jackrabbit. It quickly penetrated 10 percent of the Internet and bogged down thousands of systems. Dozens of major colleges, government facilities and research centers fell victim to Morris rogue code. The casualties included Lawrence Livermore Labs, UC Berkeley, UC San Diego, Stanford University and dozens of other sites.

"Back then, there was no Web, and the Internet was largely academically driven," says Keith Bostic, who fought the worm at UC Berkeley. "The universities ran the big sites, and those were the sites that the worm hit hardest."

Adds Peter Yee, another UC Berkeley veteran: "I was at school that night, and we noticed the computers were all getting slower and slower. The worm crawled into a machine and then tried to get into other machines. It kept on re-infecting machines that were already infected."

In the days before Internet commerce and global e-mail, the Morris Worm cleanup effort cost anywhere between $200 to $53,000 per site, according to court documents. In todays world of interconnected sites, the clean-up costs for a similar outbreak could be astronomical.

Repeat Offender Could a plague like the Morris Worm infect 10 percent—or more—of todays Internet? It depends upon whom you ask. Some security experts say todays Internet is too heterogeneous for a single worm to infiltrate so many different platforms. But Global Integrity cyber law expert Mark Rasch—the attorney who prosecuted the Morris case—says the Net is just as vulnerable today as it was in 1988.

Morris, now working at MITs Lab for Computer Sciences, declined comment for this article. But interviews with programmers who fought the worm, as well as court documents and Internet archives, paint a vivid picture of the disaster and the man behind it all.

Good Kid, Bad Move Morris didnt set out to become a cyberpunk. And its certainly unfair to lump Morris in with former dark-side hackers like Justin Tanner Petersen or media hounds like Kim Schmitz.

Morris defenders say the worm incident was merely a complicated software experiment gone bad. "Rob was a curious guy who accidentally opened a Pandoras box," says a friend of Morris, who requested anonymity.

At the time of the worm incident, Morris was a first-year graduate student in Cornell Universitys computer science Ph.D. program. Morris wrote the worm in October 1988 and released it onto the Internet on Nov. 2 of that year. The worm infiltrated systems through holes in sendmail and finger daemon, among other things. Its first target was a VAX server at MITs Artificial Intelligence Lab. Morris selected MITs systems to disguise the fact that the worm came from Cornell, according to court documents.

Morris designed the worm to ask Sun-3 and VAX systems whether they already had a local copy of the worm. The worm would skip systems that replied "yes." In theory, this would prevent the worm from copying itself endlessly and bogging down the Internet.

However, Morris was concerned that systems administrators would block the worm by programming their computers to falsely respond "yes." To beat that potential defensive measure, Morris programmed the worm to duplicate itself every seventh time it received a "yes" response, according to court documents.

Big Mistake Morris seven-to-one ratio turned out to be a fatal design flaw. The ratio wasnt high enough to slow the programs reproduction. The worm quickly spread from systems on the East Coast to the West Coast, and the Internets first disaster was under way.

When Morris realized the worm was reproducing faster than he had expected, he contacted a friend at Harvard, Andy Sudduth. The two allegedly discussed fixes for the worm, according to court documents. Sudduth quickly posted an anonymous message on the Internet, warning users about a rapidly reproducing worm and instructing readers how to defeat it.

But Sudduths message got blocked by a downed Internet gateway. In a cruel ironic twist, an administrator had shut down the gateway in an attempt to limit the worms progress.

Sudduths warning message didnt get through the gateway for about two days, but dozens of administrators around the world began to notice problems within hours of the worms release.

Yee, a UC Berkeley student and a contract worker for NASA at the time, was among the first people to spot the problem. "I was up all night working through the Morris worm," says Yee, who now works for Spyrus, a security vendor in San Jose, Calif. "I dont think I got home until 7 a.m. the next day."

Yee posted a message about the problems to a TCP-IP mailing list within hours of the worms release. With Sudduths message still blocked, Yees electronic dispatch was one of the first known communications about the worm. The message suggested turning off several services that the worm apparently used, including telnet, ftp, finger, rsh and SMTP.

"Turning off those services was the short-term fix," says Yee. "We left those services off while the research group worked to decompile it." Decompiling the worm was a critical step. This procedure unlocked the worms source code, allowing researchers to identify security holes that Morris program was exploiting. "Once you figure out how the program works, you can figure out which [security] holes to patch," says Yee.

Systems administrators at UC Berkeley, MIT and other schools worked around the clock for nearly two days to analyze the worm. By noon on Nov. 4, MIT and Berkeley had completely disassembled the worm. Most of the infected systems were back online within days of the incident.

Hit and Run Researchers say the worm had an "attack and defense" design. First, the worm would locate Internet hosts and user accounts to penetrate, then it would exploit security holes on remote systems to pass across a copy of the worm. The worm also used three defense tactics: It changed its name to minimize intrusion detection; it moved into memory and deleted its own file-system data to cover its tracks; and it used a short burst of random numbers to test a connection before moving onto a system.

Fortunately, the worm had no malicious code. Unlike some recent viruses, the Morris worm didnt erase or corrupt any of the hosts data, and it didnt attempt to steal any information.

"The [Morris] worm took systems down from load," says Eugene Spafford, a professor of computer sciences at Purdue University and a widely regarded security expert. "It didnt really damage systems."

"The Morris worm could have been a lot worse," adds Bostic, who now works for Sleepycat Software. "It just tied up the CPU. Imagine if the worm had been written to delete all of the hosts data instead? Fortunately, most worm authors dont have malicious intent. Its mostly kids having fun and showing off. But every once in a while you get an _ _ _hole in the mix."

Such was the case last week, when NakedWife became the latest virus to spread across the Internet via Microsofts Outlook program.

While the Morris worm moved from system to system without any user interaction, a virus like NakedWife (a.k.a. JibJab) needs unsuspecting users to propagate itself. NakedWife arrives as an e-mail attachment. When users activate the attachment, the virus wipes out vital Windows files and uses Outlook to e-mail itself to more unsuspecting users.

As we went to press, NakedWife had infected nearly 70 organizations. Virtually every major media outlet covered the story, yet NakedWife was a relatively minor disaster compared with the Morris Worm, which infected 10 percent of the Internet during its brief outbreak.

Famous Last Words E-commerce proponents downplay the risk of another Morris-type outbreak. They point out that todays Net is built on a long list of heterogenous operating systems—including Unix, Linux, Windows NT, Windows 2000, MacOS and so on.

In theory, the odds are relatively low that a single silver bullet could kill such a diverse system.

Yet those who fought the Morris worm believe history could repeat itself. "Something like that could certainly happen again," says Bostic. "As more and more Windows machines get connected to the Net, it could create a more homogenous system with lots and lots of vulnerabilities."

That was the case with most recent Internet-related viruses, which used Outlook—Microsofts nearly ubiquitous e-mail client—to propagate .

Experts say even the 13-year-old Morris Worm could take down some of todays Internet sites. Explains Purdues Spafford: "The old worm would need to be updated to use current library calls appropriately, but the basic technology would still allow it to propagate a little—many sites still havent fixed the remote login problem. If the Worm were updated to probe for buffer overflows in other programs than the finger daemon, then that would work, too. We still have companies releasing software with that form of bug in place."

So, does anyone actually still have the worm? Reveals Spafford: "I deleted that information years ago, although I may have it on tape somewhere."

Maybe theres a sequel in the making. Just dont offer the lead role to Robert T. Morris. Hes not much for the limelight.

MARS CLIMATE ORBITER FAILURE BOARD RELEASES REPORT, NUMEROUS NASA ACTIONS UNDERWAY IN RESPONSE

Wide-ranging managerial and technical actions are underway at NASA's Jet Propulsion Laboratory, Pasadena, CA, in response to the loss of the Mars Climate Orbiter and the initial findings of the mission failure investigation board, whose first report was released today.

Focused on the upcoming landing of NASA's Mars Polar Lander, these actions include: a newly assigned senior management leader, freshly reviewed and augmented work plans, detailed fault tree analyses for pending mission events, daily telecons to evaluate technical progress and plan work yet to be done, increased availability of the Deep Space Network for communications with the spacecraft, and independent peer review of all operational and contingency procedures.

The board recognizes that mistakes occur on spacecraft projects, the report said. However, sufficient processes are usually in place on projects to catch these mistakes before they become critical to mission success. Unfortunately for MCO, the root cause was not caught by the processes in place in the MCO project.

"We have mobilized the very best talent at the Jet Propulsion Laboratory (JPL) to respond thoroughly to the specific recommendations in the board's report and the other areas of concern highlighted by the board," said Dr. Edward Stone, director of JPL. "Special attention is being directed at navigation and propulsion issues, and a fully independent 'red team' will review and approve the closure of all subsequent actions. We are committed to doing whatever it takes to maximize the prospects for a successful landing on Mars on Dec. 3."

The failure board's first report identifies eight contributing factors that led directly or indirectly to the loss of the spacecraft. These contributing causes include inadequate consideration of the entire mission and its post-launch operation as a total system, inconsistent communications and training within the project, and lack of complete end-to-end verification of navigation software and related computer models.

"The 'root cause' of the loss of the spacecraft was the failed translation of English units into metric units in a segment of ground-based, navigation-related mission software, as NASA has previously announced," said Arthur Stephenson, chairman of the Mars Climate Orbiter Mission Failure Investigation Board. "The failure review board has identified other significant factors that allowed this error to be born, and then let it linger and propagate to the point where it resulted in a major error in our understanding of the spacecraft's path as it approached Mars.

"Based on these findings, we have communicated a range of recommendations and associated observations to the team planning the landing of the Polar Lander, and the team has given these recommendations some serious attention," said Stephenson, director of NASA's Marshall Space Flight Center, Huntsville, AL.

The board's report cites the following contributing factors:

- errors went undetected within ground-based computer models of how small thruster firings on the spacecraft were predicted and then carried out on the spacecraft during its interplanetary trip to Mars
- the operational navigation team was not fully informed on the details of the way that Mars Climate Orbiter was pointed in space, as compared to the earlier Mars Global Surveyor mission
- a final, optional engine firing to raise the spacecraftÕs path relative to Mars before its arrival was considered but not performed for several interdependent reasons
- the systems engineering function within the project that is supposed to track and double-check all interconnected aspects of the mission was not robust enough, exacerbated by the first-time handover of a Mars-bound spacecraft from a group that constructed it and launched it to a new, multi-mission operations team
- some communications channels among project engineering groups were too informal
- the small mission navigation team was oversubscribed and its work did not receive peer review by independent experts
- personnel were not trained sufficiently in areas such as the relationship between the operation of the mission and its detailed navigational characteristics, or the process of filing formal anomaly reports
- the process to verify and validate certain engineering requirements and technical interfaces between some project groups, and between the project and its prime mission contractor, was inadequate

The failure board will now proceed with its work on a second report due by Feb. 1, 2000, which will address broader lessons learned and recommendations to improve NASA processes to reduce the probability of similar incidents in the future.

Mars Climate Orbiter and its sister mission, the Mars Polar Lander, are part of a series of missions in a long-term program of Mars exploration managed by the Jet Propulsion Laboratory for NASA's Office of Space Science, Washington, DC. JPL's industrial partner is Lockheed Martin Astronautics, Denver, CO. JPL is a division of the California Institute of Technology, Pasadena, CA.