SYSTEM UNDERSTANDING

System bugs are unexpected events during operation of the system that cost the customer or user in terms of time, money, or human life. Bugs can be very expensive:

- In 2000, software sales reached $180 billion, yet bugs cost between $20 billion and $60 billion to fix according to a study by the National Institute of Standards.
- On February 1, 2003, the Space Shuttle Columbia exploded on descent killing seven astronauts due to a piece of foam that came loose during its ascent.
- On August 14, 2003, electrical faults in Ohio caused a large-scale electrical blackout affecting 50 million residents across several states (costs to US economy estimated at $4-10 billion).
- On August 18, 2003, the Sobig-F computer virus flooded email servers worldwide with 200 million emails (estimates of productivity loss exceed $1 billion).
- On June 4, 1996, an Airane 5 rocket crashed shortly after lift-off due to buggy software that was not needed during launch (loss of $500 million).
- On February 25, 1991, a Patriot missile defense failed to intercept a Scud missile due to a software bug in the weapons control computer, subsequently killing 28 American soldiers.

Fixing and isolating root cause on known bugs once they are discovered consumes time and resources after bugs appear late in the development cycle or once the product has been shipped. The conventional wisdom is that taking simple steps to avoid future bugs is both costlier for product development teams and in conflict with their time-to-market or schedule requirements. When the cost to the customer, to do root cause analysis, and to fix these bugs are all accounted for, then avoiding bugs early on becomes the fastest way to go to market, reduce development costs, and ensure customer satisfaction all at once. Being resigned to bug-fixing as a routine can act as disincentive for development teams to identify and take advantage of opportunities that reduce or avoid bugs earlier in the development cycle which is easier than fixing these bugs later.

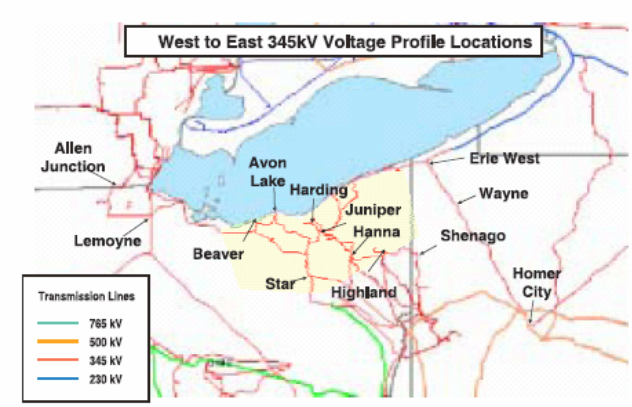
Examining the root cause analysis of the most expensive system bugs with documented case studies (Blackout of 2003, Osprey helicopter, Space Shuttle Columbia, Chernobyl, AT&T Switching Collapse, Bhopal) shows what can happen when system understanding is lost and that there are common factors present in each case that contributed to the bug—all related to lack of system understanding. The maxim “If it ain’t broke, don’t fix it”[1] needs to be replaced with “it better not break, but if it’s going to break then it’s much less expensive when both you and your customer understand when it will break.” Bugs that appear unexpectedly in the customer environment can be cost-effectively avoided if the system remains simple and understandable throughout its development, by investigating and eliminating technical ambiguity, and by making technical reviews both frequent and engaging. Products that perform to their expectations result in greater customer satisfaction, reputation in the marketplace, and repeat customer wins.

[1] See the debate surrounding fixes to foam shedding on the Space Shuttle in “For NASA, Misjudgments Led to Latest Shuttle Woes,” front page, New York Times, Sunday July 31, 2005.

© 2004 Devabhaktuni Srikrishna
http://www.devabhaktuni.us
email: srikrishna <at> devabhaktuni.us

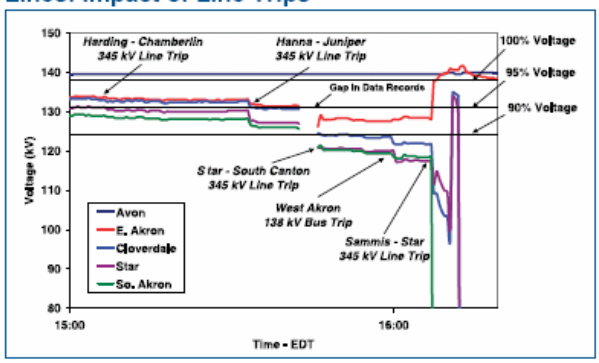
[3] [p. 34, U.S.-Canada Power System Outage Task Force]

© 2004 Devabhaktuni Srikrishna
http://www.devabhaktuni.us
email: srikrishna <at> devabhaktuni.us



3

Figure 5.14. Voltages on FirstEnergy's 138-kV Lines: Impact of Line Trips



4

On August 14, 2003, vast portions of the Midwest/Northeast United States and Ontario, Canada lost power due to a cascading electrical blackout that was triggered by a tree contacting a high-tension power line. With what initially began in Ohio, the cascade spread at an increasing rate in just over an hour across the Eastern Interconnection of the North American Electrical Grid to ultimately cover a region affecting 50 million residents. Known as the August 14th Blackout in the United States and Canada, it resulted in an estimated $4-10 billion in costs to the United States economy. For the month of August, Canada's GDP was down by 0.7%[5].

Several aspects of the construction and operation of the electrical grid resulted in the blackout which was triggered by events of that day as analyzed in the government-funded study [U.S.-Canada Power System Outage Task Force]. Nearly half of the causes (47%) were labeled by the study as "Inadequate System Understanding[6]."

The North American Electrical Grid has evolved into a vast network of transmission lines that deliver power to customers (i.e. load) all across the United States and Canada from power generation facilities located anywhere on the grid[7]. Like the Internet, it has no central point of control. Power utilities and producers under loose coordination of regional and national reliability councils own and operate the transmission and generation assets. This establishes a market for electricity that offers annual savings of an estimated $13 billion to US customers[8]. But unlike the Internet where data is actually stored and forwarded at intermediate routers, large scale storage of power has not been economical. A fundamental principle of the grid's operation is that all power generated is instantaneously consumed by the attached load[9].

---

[4] [p. 68, U.S.-Canada Power System Outage Task Force]
[5] [p. 1, U.S.-Canada Power System Outage Task Force]
[6] [p. 19, U.S.-Canada Power System Outage Task Force]
[7] Strictly speaking, the North American Electrical Grid is actually composed of three separate interconnections which are for the most part electrically isolated from each other, one for the Western US & Canada, one for the Eastern US & Canada, and one for Texas, see [p. 6, U.S.-Canada Power System Outage Task Force].
[8] [p. 32, U.S.-Canada Power System Outage Task Force]
[9] [p. 6, U.S.-Canada Power System Outage Task Force]

© 2004 Devabhaktuni Srikrishna

email: srikrishna <at> devabhaktuni.us

Any transmission or generation facility can fail randomly or unexpectedly. The grid is operated such that it remains tolerant of failure of the single most critical facility at all times (called the N-1 criterion)[10]. Whenever a failure occurs that puts the grid in violation of the N-1 criterion, operators of utilities and generation facilities are required to manage their portion of the grid to return it to the N-1 criterion within 30 minutes or less. Operators expected to do this by exploring worst case contingencies in light of the new conditions and performing load shedding. Here, load shedding refers to reducing load by turning off selected customers. Management of the grid necessitates planning and management on different timescales ranging from years to days ahead, as well as real-time monitoring to sense and react to unexpected contingencies that may affect earlier plans. This fail-over process is not automatic like in the Internet where routing technology automatically recalculates new paths for data to flow when other fail. Neither was there a single entity responsible for enforcing reliability of the vast interconnected electrical grid across a large number of operators. Reliability has been loosely coordinated by pan-utility organizations (reliability councils), which were in turn supported and funded by the utilities themselves. These councils, tasked with ensuring reliability across the grid, have no direct control over the utilities. The councils can only provide guidance to utilities about maintaining reliability based on projections of load and generation.

The blackout was triggered over the course of just over an hour by a sequence of losses of transmission lines and generation facilities supplying the Cleveland-Akron area of Northern Ohio. Neighboring regions, suddenly having to bear massive amounts of load originating in Northern Ohio, experienced an unsustainable draw of power. This caused the surrounding transmission lines and generation facilities to short, just like a short circuit induces a fuse to trip. To prevent such a cascading collapse in case of local failures, utility operators are required to invoke load shedding procedures to turn off selected customers and maintain stability of the grid. If loss in transmission and generation capability was followed by commensurate load shedding (1500 MW[11]) in the Cleveland-Akron area, the grid would have remained stable and a broader collapse averted. According to the Blackout Report, another one-third (35%) of the causes were labeled as “Inadequate Situational Awareness.” So why did the situation remain unrecognized for so long until it was too late, ultimately precluding the required action from being taken? This is primarily due to three reasons.

The origin of the blackout was traced to practices and events of the utility controlling the Cleveland-Akron area, First Energy. During the summer afternoon of August 14 when load was high but within expected ranges, First Energy’s portion of the grid was being operated close to its limits on (real and reactive) power/load, with little margin available in case of contingencies. First Energy’s operators did not have training in simulations of emergency or crisis situations, nor did First Energy’s reliability council (MISO) did have authority to enforce safety directives.[12] On August 14, several transmission lines were sagging and shorting due to contact with trees whose growth was unchecked, allowed by poor management of tree growth by First Energy.

These transmission line failures (induced by tree-growth) would normally have been detected by sensors, triggered alarms, and would have reported back to First Energy’s

[10] [p. 9, U.S.-Canada Power System Outage Task Force]
[11] [p. 45 & 70, U.S.-Canada Power System Outage Task Force]
[12] [p. 67, U.S.-Canada Power System Outage Task Force]

© 2004 Devabhaktuni Srikrishna
http://www.devabhaktuni.us
email: srikrishna <at> devabhaktuni.us

operators giving them the opportunity to take corrective action. But due to a bug in First Energy's alarm monitoring software unknown to them at the time, these events were not being reported to the operators, which permitted these conditions to recur and worsen. While operators of neighboring utilities were calling First Energy to report failures of the same transmission lines, First Energy operators ignored those warning signs and instead deferred to the results of their software without questioning what they saw.

Finally, MISO had in place its own software applications to model the electrical state of the grid (state estimator) and analyze it for contingencies (real-time contingency analysis). Earlier in the day, a transmission line in a neighboring region tripped outside the First Energy control area. The status of this tripped line outside of MISO footprint was not monitored by MISO in real-time, even though its status was essential to correctly estimating the current state of the electrical grid This inaccurate input fed to MISO's tools led to inaccurate predictions, so the output of the tool was useless in detecting the impending blackout. Should MISO have correctly estimated the state of the grid, the deteriorating conditions in First Energy's area might have been recognized in time for First Energy to take corrective action and prevent the blackout.

.

Several questions emerge about the electrical grid and its operation:

1. Why are only N-1 failures, and not compound (multiple) failures considered when assessing the stability of the grid?
2. Given how tightly interconnected the electrical grid is, why is there no system in place to automatically isolate local failures through load shedding and/or disconnection, to prevent them from spreading more broadly?
3. The blackout investigation employed extensive modeling and simulation to learn the causes of the blackout. What elements of that could have been done in real-time to aid operators by anticipating the conditions approaching the cascade in order to prevent it?
4. Why is the grid not monitored as a whole, and that too without built-in redundancy, in order to prevent failures of software as had occurred at First Energy's alarm monitoring and MISO's state estimator?
5. Why is responsibility for the reliability of the grid diffused across multiple organizations?

What Are System Bugs?

Introduction

System bugs are unexpected events during operation of the system that cost the customer or user in terms of time, money, or human life. If root cause analysis of bugs uncover technical failures which appear simple in hindsight, then why do these root causes exist in the first place? Bugs are created during product development when the product development organization does not understand their product's behavior and how it will interact with its operational environment. Bugs are not primarily a technical issue, but they are rooted in the beliefs and fault lines in high-tech organizations.

It is rare for people to question the belief that bugs are unavoidable, and that fixing them in advance involves extra effort which necessarily impacts schedule. The root cause of bugs is that ambiguity in the technical understanding leads to poor decision making,

© 2004 Devabhaktuni Srikrishna

email: srikrishna <at> devabhaktuni.us

creating these bugs that will then need to be fixed later at much greater cost. Team members also become reluctant to investigate and understand technical ambiguity when their colleagues and managers from whom they need buy-in do not yet appreciate or recognize the need for such investigation. In the absence of consensus that there is technical ambiguity, this tension creates prisoners' dilemma-like disincentives for team members to pursue fuller understanding of system bugs until after they damage is done, when the need for understanding and investigation becomes blatantly obvious.

Product development team members who are convinced that bugs are unavoidable, become accepting of lengthy bug-fix crunch in the latter stages of their projects. They become accustomed to the release of buggy products to customers. Using bug-fixing as modus-operandi creates makes it harder to see opportunities to reduce and avoid bugs earlier in the development cycle when it is much cheaper to do so. The longer a bug goes undetected as the system is being constructed, the more it becomes veiled by further development, the more likely it is to cascade into other bugs, and the more costly and time-consuming it ultimately becomes to identify and fix before damage is done.

Why do people believe bugs are so hard to anticipate or avoid? The best insurance against introduction of bugs is catching them early in technical reviews with team members. In practice, reviewers tend to be perceived as more intelligent by observers when they are negative than when they are positive, so reviews are often times avoided since they can turn out to be unproductive and painful for their participants. .Past success of a system is also incorrectly used as a reason to expect success in future situations leading to a false sense of confidence. Only bugs or failures can prove the existence of system behavior, but working systems alone can't offer proof that they will work the next time around.

Six Factors That Lead To Bugs

Six factors can typically be identified as having contributed directly to the introduction of bugs after root cause analysis has been done. Since these factors are widespread in engineering environments, it is not surprising why bugs are also common. Identifying one or more of these factors in an engineering or operational organization serves as a telltale sign that bugs lurk within the system

1. Lack of system understanding

Regardless of the size of the system, product development teams and system operators often cannot fully appreciate the relationships between components of a system.

2. Conflict of interest where quality is overridden

The system design, its operation, and/or the organizations involved are often subject to priorities which conflict with quality (or safety). No one is then left directly responsible for quality or even the assessment of quality. On the surface, these conflicts might seem inevitable due to economic and organizational constraints. But after a bug shows up in a real-world environment, the economic consequences of the bug become clear, and overriding quality rarely serves anyone's long-term interest.

3. Ignoring a known reliability issue

While a system failure proves the existence of bugs, the absence of failure does not offer assurance that the system will remain stable. System stability merely says that the

© 2004 Devabhaktuni Srikrishna
http://www.devabhaktuni.us
email: srikrishna <at> devabhaktuni.us

system has some chance of succeeding in the future, but not necessarily so. Several catastrophic system failures have been traced directly to unwarranted extrapolation of past success to the future. Known reliability issues within the system are often given less importance as a record of success is built up, until the day the system fails.

4. Hasty decision making

People with valuable technical understanding often offer warnings about a bug or impending failure situation. If these are not conclusive or provable at the time, they are easily ignored by others including managers making a decision regarding the bug.

5. Inability to sense danger

The severity of a system failure can be magnified when the system operator is offered insufficient opportunity to take actions to mitigate the system failure, unless exceptional cases are dealt with explicitly in the system design. When major effort often goes into engineering for the common case, little thought can end up being given to listing and detecting exceptional cases that lead to unexpected system behavior.

6. Intolerance of system design to component failure

Unexpected behavior or failures of component within a system are to be expected, but they don't necessarily have to lead to system failure.

Case Studies

We discuss several well-known cases of system failures and accidents which have led to costly failures and in some cases, fatalities, in customer environments. These were chosen since we have the benefit of publicly accessible accident investigation and failure analysis to understand what happened.

For each case we briefly review the economic impact, describe the bug, and then question its causes. Almost every one of six factors identified in the prior section can be identified among the root causes of the cases included in this book for which dedicated investigations took place (see table below).

© 2004 Devabhaktuni Srikrishna
email: srikrishna <at> devabhaktuni.us

Table 1 Six Factors That Lead to System Bugs

	Conflict of Interest: [14] **Decisions about bugs and quality made by individuals & organizations with competing goals and interests—no one is primarily responsible for system reliability and operational behavior.**	**Lack of System Understanding:** **Lack of technical system understanding** 1. **Not tested,** 2. **Not reviewed,** 3. **Not implemented as designed or intended** 4. **Requirements not understood** 5. **Unknown-unknowns**	**Ignoring a Known Reliability Issue:** **Known poor reliability of system or component (versus Unknown issue or unknown-unknown)**	**Hasty Decision-making:** [15] **Managers & workers did Not heed warning signs (denial) and took no action**	**Cannot Sense Danger:** **System's inability to anticipate impending failure**	**Intolerance of System Design to Component Failure:** **Design does not permit system recovery or graceful failure**
Blackout in the US & Canada	Utility-funded reliability councils	Running the Northern Ohio area grid near capacity and violation of NERC rules	Transmission lines shorted by tree growth	Operators failed to recognize transmission line problems after hearing about them from neighboring utilities	Alarm software application became inoperable (i.e. hung)	Trouble areas in the grid were neither isolated, nor could the rest of the grid be isolated into self-sustaining islands through load shedding or line disconnection
Space Shuttle Columbia	Safety decisions impacted by managerial goals To maintain launch schedule	1. Failure to classify the foam strikes as in-flight anomaly 2. Failure to understand fragility of thermal protection system.	Repeated instances of foam strikes on prior shuttle missions	Shuttle managers ignored warnings from reports and engineers	Could not See The Wing or Sense The Loss of Tiles	Thermal protection system could not withstand a failure of one or more fragile tiles
Bhopal	Plant Responsible for Safety & Not Reviewed By Community	Poor recognition of the risks posed by shutting off safety mechanisms.	Leaky valves, poor safety practices	Ignoring readings on gauges on day of accident due to their unreliability	Sensors, Logs, and Gauges Not Present or Functional	Large quantities stored instead of using smaller, protected containers
Chernobyl	Dual purpose: Weapons production and Power Generation	lack of understanding by experimenting engineers of 1. Reactor dynamics (negative coefficient) 2. Damage to reactor control		Engineer went ahead with experiment despite concerns of other colleagues in the plant	No built-in warnings or checks before irreversible changes to reactor	No containment shell
Osprey Helicopter	System Testing & Qualification Performed Exclusively by Manufacturer	Marines did not inspect or test software functioning which is critical To Osprey's operation, nor did manufacturer do a thorough job testing exceptional cases	Chafing of hydraulic lines	The known chafing of hydraulic lines did not arouse suspicion to look in inaccessible parts of the line	No warning of software bug leaving no time for pilot to react	Malfunction In Software Threw Helicopter Into Uncontrollable State

[14] See [p. 69-71, 74-75, Leveson] on conflicting goals and their consequence for safety engineering.
[15] See [p. 64-68, Leveson] for how ignoring warning signs is a precursor of accidents.

© 2004 Devabhaktuni Srikrishna

email: srikrishna <at> devabhaktuni.us

Osprey V-22 Crash

[16]

On December 11, 2000, four US Marines died conducting a training mission in North Carolina when the pilot lost control of their Osprey V-22 tilt-rotor aircraft and it crashed.[17] An accident investigation later determined that a combination of mechanical and software failures within the V-22 caused the crash.

The V-22 is a hybrid between a helicopter and a propeller airplane. Two propeller engines located in nacelles (a streamlined enclosure for each engine on the aircraft) on the tip of each wing can rotate their orientation from vertical to horizontal and vice-versa. At low-speeds the propellers are aimed vertically. As the aircraft speeds up the propellers change their orientation to horizontal allowing it to move even faster.

While in flight, a titanium hydraulic line ruptured. This line supplying actuators controlling the pitch of the blades on the left propeller rotor was under pressure. The rupture was induced by chafing from a wire bundle that rubbed against the line during flight. Although the rupture resulted in an uneven distribution of hydraulic pressure between left and right side actuators, the design of the aircraft was such that a single rupture would not have caused the accident.and would have kept the plane in the pilot's control. Another failure was required.

Sensing the loss of hydraulic pressure, the control software of the aircraft flashed a warning to the pilot instructing him to reset the computer. However, upon reset by the pilot per instruction, a software bug caused the rotors to respond unexpectedly without warning to the pilot. The reset should have done nothing, but instead it induced significant changes to both the pitch and thrust of the propeller rotors, while both sides

[16] http://www.fas.org/man/dod-101/sys/ac/v22-report.pdf
[17] See [Berndt].

© 2004 Devabhaktuni Srikrishna
email: srikrishna <at> devabhaktuni.us

were already operating with uneven hydraulic pressure. This unfortunate coincidence of system faults led to loss of control and the eventual crash of the aircraft.

Following the crash, US Marine accident investigators honed in on the need for more thorough software testing of the flight control software. A "Blue Ribbon Panel"[18] studying the V-22 program later discovered that the testing of the flight control software was conducted by the V-22's designer/manufacturer and it did not include the case of hydraulic failure. Based on inspections by the military, chafing of titanium hydraulic lines supplying the nacelles of the V-22 had been reported as early as June, 1999. However, they also found that the specific part of the line that experienced chafing in this accident was not amenable to inspection due to limited access granted by inspection panels on the nacelle.

These findings raise questions regarding the design, testing, and operational inspection of the V-22:

1. Was the complexity of the novel aircraft design, reflected in both the software and hardware design, enough to warrant increased attention to system validation or test harnesses than had been given to previous designs?
2. Why was the software not tested and reviewed for exceptional cases such as the hydraulic failure?
3. How were inspectors of the hydraulic lines satisfied if they could not access certain parts of the lines to check for chafing?

The Creation of System Bugs

The economics of bugs in the product development cycle

Bugs become costlier to fix the further development gets

The cost of finding root cause, fixing, and experiencing a bug rises rapidly as engineers build successive developments into a system.[19] Bugs may be introduced at any point during the product development process, and bugs left unfixed cascade into more bugs during development. One way to picture the economics of this activity is that the cost of making a change to fix or mitigate a bug increases at each successive stage. This is illustrated in Figure 1.

[18] The Blue Ribbon Panel was investigating the V-22 program on behalf of the US Secretary of Defense, see [p. 22-26, Dailey].

[19] These stages are widely understood to be requirements definition, design, prototyping, implementation, unit test, system test, beta, customer deployment, long-term customer usage. Each stage may be repeated or removed as deemed appropriate by a particular development team

© 2004 Devabhaktuni Srikrishna

email: srikrishna <at> devabhaktuni.us

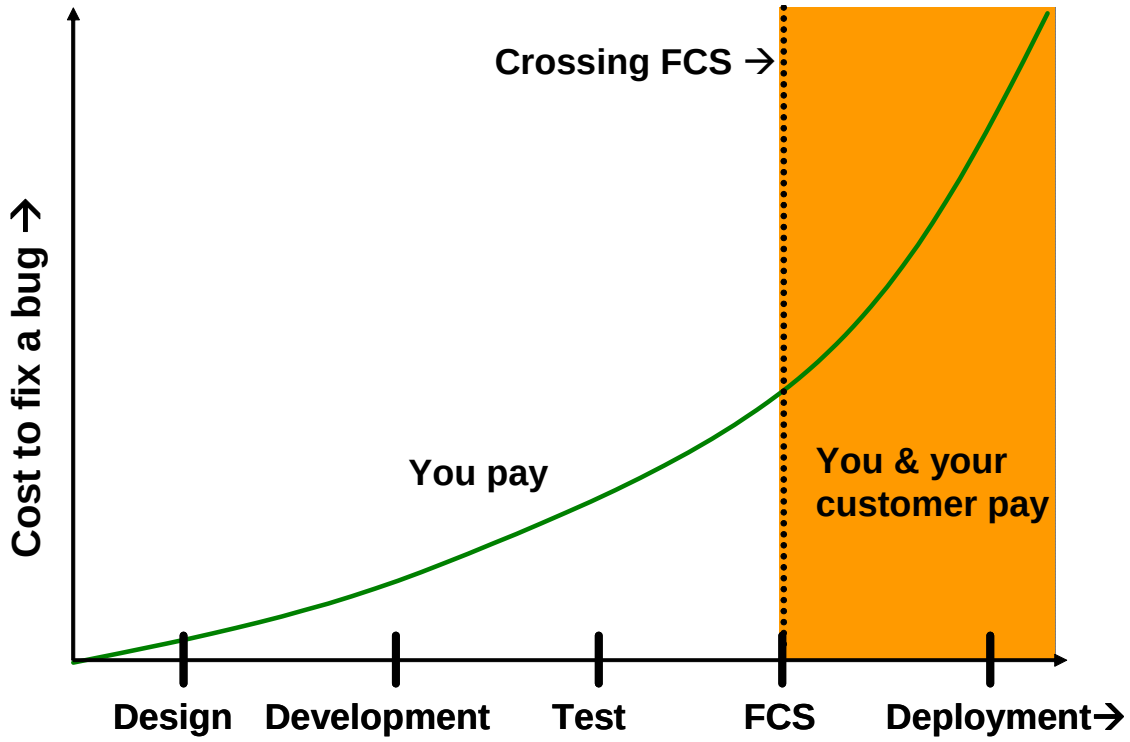


Figure 1 Crossing FCS (First Customer Shipment)

This picture of increasing cost is also supported by the representative model given by the National Institute of Standards for the rise in costs to fix bugs in software (repair defects), [p. 5-4, Tassey] as shown below. During the requirements and design phases, the product is fluid and it is easy to make changes. Once code has been written, it takes more effort to change or fix the code. For instance, when different code modules are integrated together, dependencies are created and multiple team members get involved, all of which introduces further cost to fix a bug. The beginning of system testing and then involvement of customers (who are by definition external to the organization) creates the greatest cost to any system change or correction.

Table 5-1. Relative Cost to Repair Defects When Found at Different Stages of Software Development (Example Only)
X is a normalized unit of cost and can be expressed terms of person-hours, dollars, etc.

Requirements Gathering and Analysis/ Architectural Design	Coding/Unit Test	Integration and Component/RAISE System Test	Early Customer Feedback/Beta Test Programs	Post-product Release
1X	5X	10X	15X	30X

Source: National Institute of Standards

Bugs may not show up for months or even years after FCS (first customer ship). At that point, the number of customers using the product will have risen, compounding the cost of a bug. Furthermore, each customer's cost of bearing the cost of the bug rises dramatically with time elapsed due to their increasing reliance on the product. If enough

© 2004 Devabhaktuni Srikrishna
email: srikrishna <at> devabhaktuni.us

time passes, organizational memory of the product development team is may even be lost due to turnover of staff and details forgotten, making it that much harder to fix the bug. The rising cost structure post-FCS is routinely confirmed by numerous case studies of engineering projects. One example is the Space Shuttle Onboard Software project which found that it cost seven times more to fix a bug after delivery than during system test [p. 106, Paulk].

Once bugs are introduced, development organizations are forced to rely on observations after the system is built. This occurs when failures or upset expectations serve to focus their attention on acquiring understanding of system behavior, sometimes at great length during failure analyses and accident investigations. It is more costly to rely on failures in the customer environment as the primary means of to recovering lost system understanding since even greater understanding can be gained while the system is being put together when it is much less expensive to avert bugs.

When a development team begins to respond to bug-reports that pour in from the field,,customer bugs can become more important and urgent compared to follow-on releases and future product development. This reactive mode can kick-start a vicious cycle in which attention paid to fixing bugs takes away time from understanding new functionality for follow-on releases, and features are rolled in without adequate design or testing. If understanding doesn't exist in the first place and is either not transferred or relearnt to as the product is further developed, the problem of follow-on releases is exacerbated. Some releases can end up being simply bug-fix releases with no new functionality added.

Bug-fixing is easy to justify because the effort involved in fixing known bugs is concrete, but the effort to prevent them is not known with as much certainty. Therefore bug-fixing efforts are likely to have more concrete justification on a case-by-case basis than bug-prevention. A straightforward interpretation of the rising cost to fix bugs says that it is most economical to begin learning system behavior as early as possible to prevent or fix bugs, well before you are forced to learn it once the customer uses the product.[21] Some system development requires repeated failures, in order to learn from and ultimately achieve success. There are some aspects of system behavior for which it may be only possible to learn empirically or experimentally after making progress on the implementation or even after the system is built and in the hands of the customer. That is not an excuse to waste early opportunities to learn about the system. Understanding the system is the foundation for its design, implementation, operation, and ultimate success, and doing so before the product is deployed or expected to work in a customer environment is the key. Some learning must take place in customer environments, and even that is best done selectively under a designated "alpha" or "beta" period whereby the customer's expectations are set appropriately.

Summary

1. The cost of identifying and fixing a bug rises with each successive stage of development.

[21] This is, or course, in contrast with product feature requirements for which it can sometimes be better to learn them after the customer can offer feedback based on using the product.

© 2004 Devabhaktuni Srikrishna
email: srikrishna <at> devabhaktuni.us

2. Once the customer becomes involved after FCS (first-customer-ship), the cost of identifying and fixing a bug rises dramatically.
3. Product development teams who view bug fixing as more concrete than bug-avoidance tend to be averse to avoiding bugs.
4. To avoid bugs, it pays to create and maintain a solid understanding of system behavior throughout the development cycle.

Technical Ambiguity

In some cases, it's very clear whether a bug is present within the product, and in such cases objective decisions based on customer impact can be made regarding what should be done—investigate it further, fix it, or just leave it alone. Such definite knowledge of the product behavior, when maintained throughout the product development cycle and well after FCS, permits rational decision making and risk mitigation.

Technical ambiguity creeps into the product due to the complexity introduced during the development process. One position is that technical ambiguity will always exist to some extent and all engineering decisions are imprecise to some degree since knowledge is imperfect and testing is never complete or foolproof [p. 202-203, Vaughan]. However, the degree of technical ambiguity remains largely under the control of the engineering organization. Provided that the will, incentives, talent, and resources exist, technical ambiguity can be avoided or at least minimized.

As understanding of system behavior is lost during the product development cycle, things don't appear cut and dry and this loss is difficult to reverse:

1. Often times it's not clear whether or not there is a bug lurking in the product. Someone (engineer or manager) may have reason to suspect the bug exists based on seasoned judgment or intuition, but this cannot be readily and clearly explained to others.
2. In some cases it's hard to even assess the consequences of the bug's presence without knowing whether or how often it is likely to show up.
3. This places managers and engineers in a tough situation as to whether they are justified in diverting valuable time and resources to investigate the bug, throwing a wrench into existing schedules and resource allocations. A positive test for the bug becomes the only definite outcome of testing, and continued negative tests for the bug say comparatively little.

Decisions made by engineers and managers can be only as good as the data available to them. Therefore, technical ambiguity is the source of many problems, not to mention headaches and discontent. Ambiguity invites the unpleasant situation of "your word versus mine." Without technical certainty, organizations can lumber along their path and find themselves in a precarious state ending up in disaster, as had been the case with both Space Shuttles, Challenger and Columbia,

> [p200, Columbia Accident Investigation Board] In neither case [Challenger or Columbia] did they have good data as a basis for decision-making. Because both problems had been previously normalized, resources sufficient for testing or hardware were not dedicated. The Space Shuttle Program had not produced good data on the correlation between cold temperature and O-ring [Challenger]

© 2004 Devabhaktuni Srikrishna

email: srikrishna <at> devabhaktuni.us

> resilience or good data on the potential effect of bipod ramp foam debris hits [Columbia]….Cultural beliefs about the low risk O-rings and foam debris posed, backed by years of Flight Readiness Review decisions and successful missions, provided a frame of reference against which the engineering analyses were judged. When confronted with the engineering risk assessments [warnings of danger prior to launch], top Shuttle Program managers held to the previous Flight Readiness Review assessments [i.e. to launch].

Therefore, with both Challenger and Columbia, managers' risk assessment differed from that of engineers due to technical ambiguity. Since the correct answer was not definitive, the final decision had a risk of being wrong, and in this particular case it turned out to be disastrous. Both managers' and engineers' differing goals were not wrong in and of themselves. If the technical ambiguity could have been resolved based on irrefutable fact, they would most likely have agreed on the same (right) course of action. Ambiguity places people in difficult situations where they are forced to make tough decisions.

As a general rule, since managers are accountable for delivering on schedule, they are prone to being more sensitive to schedules and less sensitive to technical ambiguity. Engineers who work close to technology tend to be more sensitive to product quality (and bugs) and will fight to eliminate technical ambiguity if given the opportunity and encouragement, but they are correspondingly prone to being less sensitive to schedule slips or variability in schedule. There are clearly going to be exceptions to this rule and the roles of some engineers and managers may well be reversed in some instances. But there are always two sides that create a schedule-quality tension, and we will use this assignment of different goals (schedule, quality) to uncover and explain their combined group behavior that can lead to undesirable outcomes.

Both the Space Shuttle Challenger mission and Columbia missions (STS-107) are examples of technical ambiguity forcing managers in the position of making tough decisions that ultimately cost the lives of their astronaut crew. Managers, Ham (Columbia) and Mulloy (Challenger), sought to justify their own position to launch the Space Shuttle based on prior history of successful missions,

> [p. 200, Columbia Accident Investigation Board] In the *Challenger* teleconference, where engineers were recommending that NASA delay the launch, the Marshall Solid Rocket Booster Project manager, Lawrence Mulloy, repeatedly challenged the contractor's risk assessment and restated Thiokol's engineering rationale for previous flights. STS-107 Mission Management Team Chair Linda Ham made many statements in meetings reiterating her understanding that foam was a maintenance problem and a turnaround issue, not a safety-of-flight issue. The effects of working as a manager in a culture with a cost/efficiency/safety conflict showed in managerial responses. In both cases, managers' techniques focused on the information that tended to support the expected or desired result at that time. In both cases, believing the safety of the mission was not at risk, managers drew conclusions that minimized the risk of delay.

The CAIB report illustrates how organizational culture, in the presence of technical ambiguity, is susceptible financial and time pressure. This outcome is not unique to NASA, but happens in both startups and large development organizations. The report points out that, in this case the safety organization, and more generally the technical organization, needs to have the ability to draw conclusions and judgment with certainty that are firmly grounded in fact. Achieving technical certainty allows the organization's

© 2004 Devabhaktuni Srikrishna

email: srikrishna <at> devabhaktuni.us

decisions to remain defensible in light of artificial pressures such as schedule and budget and allows these decisions to be neither be too conservative nor too lax, but simply accurate.

Summary

1. Technical ambiguity creeps into systems due to accumulation of complexity during product development.
2. Ambiguous system behavior leads to poor decision making during product development, as well as the classic schedule-quality tension between team members, managers, and engineers.

© 2004 Devabhaktuni Srikrishna

email: srikrishna <at> devabhaktuni.us

The Space Shuttle Columbia

22

By 2003, NASA had successfully completed well over 100 successful Space Shuttle missions in two decades with the sole exception of the Space Shuttle Challenger tragedy in 1984. On February 1, 2003, the Space Shuttle Columbia was destroyed during its descent into the Earth's atmosphere as part of mission STS-107, killing seven astronauts and putting the entire Space Shuttle Program in question.

The causes of the accident were explained in the Columbia Accident Investigation Board Report.[23] The report shows how a piece of foam insulation for the Shuttle's large central solid rocket booster broke loose due to the high-velocity of airflow during ascent. Although damage due to foam strikes was suspected while Columbia was in-orbit, the full extent of the damage remained unknown to pilots' and mission control's due to their inability to visually inspect the wing. On several earlier Space Shuttle flights, foam debris had struck the wing creating holes of various sizes in the thermal protection system. With the Columbia (STS-107), the hole was so large that it exposed the aluminum wing to hot gasses created during the high-velocity descent into Earth's atmosphere. The

[22] CAIB
[23] The 248-page report is a remarkably accessible account that leaves few stones unturned.

© 2004 Devabhaktuni Srikrishna

email: srikrishna <at> devabhaktuni.us

result was that the Columbia burnt up scattering its remnants across the south and southwestern United States.

In addition to identifying the technical cause, the CAIB report exposes the thoughts and actions of engineers and managers before and during the mission, and set in the broader political climate of the space program. The threat of foam debris was apparent from several Space Shuttle flights before, yet NASA managers did not appreciate how severe the consequences could have been and refused to classify these events as in-flight anomalies.

The board found that concerns about maintaining flights schedules on the part of NASA managers made them repeatedly decide against delaying future launches which would have been required to understand the root cause of the problem. Furthermore, engineers who harbored concerns either remained silent or did not get the time and resources needed from managers to investigate the warning signs and fix the problem, which would have ultimately saved lives and better served the goals of the Shuttle program, rather than bringing it to a halt. The board traces these unfortunate decisions to top-down pressure from NASA's senior leadership, the "better, faster, cheaper" mentality, and ultimately to shifting priorities set by Congress over multiple decades.

Questions about the technology and decision making in the Space Shuttle Program:

1. Was the fragile design involving foam necessary, or were there more robust alternatives that could have been employed instead?
2. Why was the root cause of repeated known foam strikes not resolved before the flight?
3. Why was there no diagnostic information available in-flight that could have warned of the breach in the thermal protection system?
4. Who was responsible for safety and quality at NASA?

How Does Team Behavior Contribute?

Sensitivity to perceptions can discourage understanding

What prevents engineers and managers from proactively trying to understand their product and seek out bugs even though they have every incentive to increase product quality? One major cause is uncontrolled interactions in the group that work to discourage engineers or managers from seeking deeper understanding of their product—the path of least organizational resistance. This happens when people pay closer attention to perceptions by their team members (peers, managers) and external influencers (customers, board members, salespeople) than to observing, experimenting with, and understanding system behavior.

It is rare that people don't worry about what others think, or that they are on the same page such that there is never a conflicting opinion. When team members care about their evaluation by others whose thoughts they have little direct control over, they tend to choose the outcome that is least risky from the standpoint of how they are perceived, rather than chose the lowest risk to the system for the following reasons:

1. In groups, people want to be perceived well, because being viewed in a negative light (i.e. taking blame or appearing less intelligent) can be damaging to their

© 2004 Devabhaktuni Srikrishna

email: srikrishna <at> devabhaktuni.us

career, sometimes irreparably. Therefore, any risk of being perceived negatively by others can quickly sway their own judgments or decisions to minimize this risk—often to the loss of the project, as we shall see.

2. By their nature, system bugs often fall into the class of high-severity, low-probability events for which the human bias is (generally) to dismiss as not worth investing resources in.[24]
3. While the results of what other people think are felt immediately, a failure that results from a system risk is hard to trace back to any one individual, engineer or manager, even when an extensive failure or accident analysis is undertaken.

Independent of what may be the right outcome from the perspective of system risk, people in organizations have an incentive to deal with risks later if dealing with them earlier would pose a risk to their perception and evaluation by others. This becomes clear when we consider the situation where a manager and engineer are trying to decide whether to invest their time and resources in gaining understanding or in fixing a bug, and each must justify their decisions to the other in order to do so. Such a situation sounds reasonable, and one would presume that it should result in the outcome that maximizes their shared interest in making sure the system works. However, this is not so when we consider that the manager and engineer don't have control over each other's subjective evaluations.

On one hand, managers are typically under pressure to meet schedules and deliver on a project, yet must also appear reasonable to the engineers who work for them. On the other hand, engineers are accountable to their managers and their career advancement relies on managers' evaluations. Each must choose one of two possible courses of action independent of the other's choice, i.e. to invest or not to invest. Once the choice is made, they run the risk that the other can harbor a negative evaluation of other. So there is built-in pressure to make decisions that will reduce or eliminate that risk of negative evaluation, rather than the shared risk of not understanding the system. The outcome is often a sub-optimal equilibrium, where the decision is not to seek the necessary understanding regardless of whether it would be prudent to do so from the standpoint of success of the system.[25]

If the manager wants to understand the system, the manager faces the risk is that the engineer will question why the manager is delaying the work to go down a (potentially) unfruitful path. Likewise, when the engineer wants to understand the system further, it can be deemed as unnecessary waste of time by the manager. Hence each can end up choosing against the course of furthering understanding since that turns out to be the safest choice for each with respect to protecting their perceptions of the other. The following tables explain each of the four possibilities.

[24] See [p. 60, Levenson]

[25] This dynamic this creates situations best represented by Nash equilibriums and the so-called "Prisoner's Dilemma."

© 2004 Devabhaktuni Srikrishna
http://www.devabhaktuni.us
email: srikrishna <at> devabhaktuni.us

Table 2 Whether to take shortcut in light of system risk

		Manager's choices	
		seeks understanding (Risks negative perception)	**insists on shortcut (No risk to perception)**
Engineer's choices	**seeks understanding (Risks negative perception)**	Lowers risk of system failure → ↓	Engineer can be accused of not delivering, not working hard enough, or wasting time ↓
	Engineer goes for shortcut (No risk to perception)	Manager can be blamed for delaying the schedule since engineer does not see the need →	Eliminates risk of negative perceptions, but increases risk of system failure i.e. deal with it later

A similar dynamic is in effect when the question is whether to try to address a problem ahead of its appearance in the customer environment (i.e. "don't fix it if it ain't broken").

Table 3 Whether to head off a potential problem or to wait

		Manager's choices	
		asks to solve problem ahead of its appearance (Risks negative perception)	**doesn't bother (No risk to perception)**
Engineer's choices	**solves problem ahead of its appearance (Risks negative perception)**	Lowers risk of system failure → ↓	Engineer can be accused of "fixing it when it ain't broken" ↓
	doesn't bother worrying (No risk to perception)	Manager appears to be making unreasonable demands. →	Eliminates risk of negative perceptions, but increases risk of system failure. i.e. deal with it later

© 2004 Devabhaktuni Srikrishna
email: srikrishna <at> devabhaktuni.us

Warnings or weak signals about a system pose a problem to decision makers since they can't be sure about the outcome of their decision. The problem is worsened when this leads to dismissal of the warning and rather than a decision to understand the issue more completely. An example of this can be seen in the Columbia Accident Investigation Board Report. They argue that so-called "weak signals" or warnings, which identify possible, but technically ambiguous risks, were not acted on by STS-107 mission management.

> [p.203, Columbia Accident Investigation Board]. NASA's challenge is to design systems that maximize the clarity of signals, amplify weak signals so they can be tracked, and account for missing signals. For both accidents there were moments when management definitions of risk might have been reversed were it not for the many missing signals – an absence of trend analysis, imagery data not obtained, concerns not voiced, information overlooked or dropped from briefings… It is obvious but worth acknowledging that people who are marginal and powerless in organizations may have useful information or opinions that they don't express. Even when these people are encouraged to speak, they find it intimidating to contradict a leader's strategy or a group consensus. Extra effort must be made to contribute all relevant information to discussions of risk. These strategies are important for all safety aspects, but especially necessary for ill-structured [*technically ambiguous*] problems like O-rings and foam debris.

In this case, the board argues that engineers' decision to voice concern was influenced by their managers' opinions, and therefore the board suggests that amplifying all perspectives is necessary to address the problem of safety. However, this is not a systematic solution since if all signals were amplified when most don't offer a definite conclusion, then they all remain as noise and there is nothing you can really do to make a better decision. Developing an unambiguous understanding is really the only way to achieve solid, actionable, information required to make sound engineering decisions.

Weak signals put managers in a tough position when they have to make decisions based on less than solid information—they must distinguish between an overly conservative prediction of failure and one that is more likely to affect their current decision. The only method at the disposal of managers is to push engineers for further "proof." As an example of this, the skepticism expressed by Challenger mission manager (Mulloy) towards engineering analysis of impending failure borders on threatening,

> [p200, Columbia Accident Investigation Board] At one point, Marshall's Mulloy, believing in the previous Flight Readiness Review assessments, *unconvinced* by the engineering analysis, and concerned about the schedule implications of the 53-degree temperature limit on launch the engineers proposed, said, "My God, Thiokol [an engineering contractor], when do you want me to launch, next April?"

The demand for proof or solid evidence is one way that managers push engineers to get increased certainty. Therefore it is not entirely misplaced—they are merely asking to eliminate technical ambiguity. However, the problem arises when the push for "proof" results in a decision not to gain further understanding, and the weak signals that remain weak, or even worse, become silenced into no signal at all.

© 2004 Devabhaktuni Srikrishna
email: srikrishna <at> devabhaktuni.us

In the case of Challenger, Mulloy maintained that delaying the launch to further understand the issue was inconceivable. Engineers could not to push back and stop the launch either, for they could not be sure of the outcome. Therefore, the net effect of even the slightest technical ambiguity is that it can create a lack of certainty and self-doubt which forces engineers into submission. As a result, they lack the confidence to pursue their thoughts about real risks which would merit further investigation.

> [p 200, Columbia Accident Investigation Board] Worried engineers in 1986 [Challenger] and again in 2003 [Columbia] found it impossible to reverse the Flight Readiness Review risk assessments that foam [Columbia] and O-rings [Challenger] did not pose safety-of-flight concerns. These engineers could not prove that foam strikes and cold temperatures were unsafe, even though the previous analyses that declared them safe had been incomplete and were based on insufficient data and testing.

The mistake that Columbia mission managers made is that the lack of "proof" was used to justify the final decision to launch without completely understanding the issue. The dynamics at work in this case are shown below.

Table 4 Whether to pursue a weak signal

		Manager's choices	
		encourages use of resources to investigate	**insists on solid proof before taking action**
Engineer's choices	**raises concern based on incomplete data**	Lowers risk of system failure, but unstable due to risk of negative perceptions → ↓	Engineer can be accused of not providing solid information ↓
	remains silent and does not investigate	Manager looks like he is wasting resources since nothing turns up most of the time. →	Eliminates risk of negative perceptions, but increases risk of system failure.

Managers and engineers who are under severe pressure to deliver may attempt to justify their decisions by referring to past successes or prior experience, without actually basing their assessments on true understanding of their system or solid evidence based on sound principles. While it is true that an instance of failure in the past or an unexpected event says something about the system with 100% certainty, past success can only provide, at best, some indication of the likelihood of future success. Unless there are enough samples to statistically quantify the probability of success based on well understood underlying probabilistic phenomenon, success cannot be directly translated into statements about expected system behavior. Such was the case with the justification given for the ill-fated Challenger launch in 1984,

© 2004 Devabhaktuni Srikrishna
email: srikrishna <at> devabhaktuni.us

> [p. 130, Columbia Accident Investigation Board] The acceptance of events that are not supposed to happen has been described by sociologist Diane Vaughan as the "normalization of deviance." …Dr. Richard Feynman, a member of the Presidential Commission on the Space Shuttle Challenger Accident, discusses this phenomena in the context of the *Challenger* accident. The parallels are striking: "The phenomenon of accepting … flight seals that had shown erosion and blow-by [cause of Challenger accident] in previous flights is very clear. The Challenger flight is an excellent example. There are several references to flights that had gone before. The acceptance and success of these flights is taken as evidence of safety. But erosions and blow-by are not what the design expected. They are warnings that something is wrong …"

The seemingly obvious yet typical mistake that managers and engineers can make is to infer based on a sample size that is too small or not representative of the full spectrum of possibilities in the customer environment, as was the case with the Shuttle Columbia launch decision,

> [p. 125, Columbia Accident Investigation Board] What drove managers to reject the recommendation that the foam loss be deemed an In-Flight Anomaly [which would have halted the launch? Why did they take the unprecedented step of scheduling not one but eventually two missions to fly before the External Tank - project was to report back on foam losses? It seems that Shuttle managers had become conditioned over time to not regard foam loss or debris as a safety-of-flight concern [due to successful past missions].

Therefore true understanding is really the only basis upon which one can ensure expected behavior of the system in a real environment.

Summary

1. Perception and the risk of negative evaluations by colleagues and managers plays an important role in decisions to gain system understanding.
2. Despite a shared interest in seeing the system work, individual incentives for team members can conspire to result in sub-optimal decisions leading to poor system understanding.
3. Warnings or so called weak signals from engineers rarely lead to actionable advice for managers unless they are turned into strong signals, the only solution is to develop solid understanding upon which to base decisions.
4. A common mistake is to use prior success as a basis for justifying success in the future, despite lack of solid evidence and true system understanding.

Chernobyl Catastrophe

[26]< include a diagram of the Chernobyl reactor >

On April 26, 1986, the world's worst nuclear accident was set off by massive explosions at the Chernobyl nuclear reactor plant in the Ukrainian republic of the USSR, dispersing nuclear fallout over much of Northern Europe. The fallout from the explosions was estimated to be the equivalent of 10 Hiroshima bombs. Thirty one people died and 299 were injured within a few weeks of the accident, hundreds of thousands of residents in

[26] http://www.world-nuclear.org/ info/chernobyl/inf07.htm

© 2004 Devabhaktuni Srikrishna

email: srikrishna <at> devabhaktuni.us

surrounding areas of Chernobyl were affected, and large tracts of land surrounding the reactor will remain contaminated with radioactivity for several decades.[27] *What advice would you have given Soviet authorities in order to prevent this international calamity?*

In reactors based on nuclear fission, electric turbines are driven by steam produced from heat in the reactor in order to generate electricity. Neutrons drive the nuclear fission, and the presence of neutrons is regulated by insertion of "control" rods into the reactor—the fewer control rods inserted, the more neutrons are available for fission, and the more energy produced by the increased nuclear fission. Water is not only converted to steam as it is circulated inside the nuclear reactor where nuclear fission takes place, but also constitutes the reactor's cooling system. However, two things about the design employed at Chernobyl set it apart from most other nuclear reactors.

First, water being circulated in the Chernobyl reactor would also absorb neutrons due to its circulation in direct proximity to the fuel core. Therefore, in departure from the design of most other nuclear reactors, a loss of water circulation or boiling of water would not only reduce the power produced, but also increase the pace of the fission reaction.

Second, nuclear reactors are typically designed with a protective concrete shell as a last-resort to contain radioactivity in the event of an accident. However, the Chernobyl reactor designed during the Cold War had the dual purpose of producing nuclear weapons material (Plutonium) in addition to producing power. Large cranes were needed to pull the spent fuel out of the reactor for reprocessing at rapid intervals, and therefore it could not support a rigid containment structure.[28]

The explosions were triggered due to an apparently misguided experiment conducted by electrical engineers who did not understand the nuclear physics of the reactor, despite reluctance expressed by operators of the plant.[29] The experiment was designed to see what would happen if steam from the reactor's cooling system stopped flowing to the electricity generating steam turbines. Would the turbines continue to turn based on their rotational inertia and continue to produce enough electricity for the reactor cooling pumps, at least until backup electrical generators could be turned on? By reducing water circulation to the reactor, the experiment inadvertently caused the fission reaction to accelerate out of control.

To make matters worse, the operators turned off the emergency reactor cooling system and disabled other safety measures as well in the course of performing this experiment intended to simulate emergency conditions. Once the reactor core began to melt this was irreversible, and efforts to regain control of the fission failed. Deformed structures blocked attempts to insert the control rods, leading to the explosion.[30]

The design of the reactor and conduct of the experiment both raise serious questions,

1. Was building a reactor without a containment shell truly in the national interest?
2. In contrast to most other reactors, was a reactor design that would accelerate upon failure of cooling systems a wise choice?

---

[27] See "Chernobyl Accident" [p. 529, Schlager] and [p. 161-164, Chiles].
[28] See [McCarthy].
[29] See "Chernobyl" in [Chapter 5, Rhodes].
[30] see [p. 532, Schlager]. A different reason for failure of the control rods is given in [Chapter 5, Rhodes].

© 2004 Devabhaktuni Srikrishna

email: srikrishna <at> devabhaktuni.us

3. Could the goals of the experiment have been served through simulation or perhaps using a turbine that was not coupled to a nuclear reactor?
4. What steps could have been taken to to anticipate the effect of the risky operations that would ensue? Why was the experiment not reviewed by other engineers or experts who understand the nuclear reactions involved?

How Teams Can Avoid Bugs

What Makes Reviews Painful, and How to Make Them Work

Technical reviews are known to generate greater understanding of the system across members of the development team, enabling them to catch and fix bugs early in the development cycle. However, technical reviews have a reputation of leaving a bad taste in people's mouths, are notorious for being difficult, are often reduced to mere formalities, or even avoided altogether in practice. Experienced managers and engineers know that when technical reviews work they can deliver spectacular results[31]. Then why do technical reviews rarely live up to their full potential?

One primary reason is that reviewers tend to be negative upon those being reviewed, especially when bugs are identified. This creates a vicious cycle whereby the reviewed are reluctant to expose their work in a meaningful and engaging way and make the review itself much less useful to the team than it could have been. By eliminating the barriers to conducting effective reviews and encouraging the curiosity of the entire team, technical reviews can be one of the most effective tools in avoiding bugs.

There are many types of technical reviews, ranging from simple discussions using a white board to code reviews to design reviews to formal inspections[32]. For the purposes of this section, we define technical reviews as any activity where a team member solicits feedback from the other about the technical aspects of the system.

The first barrier to reviews is that reviewers who offer negative opinions are perceived by observers as being more intelligent than those who offer positive evaluations, even when the substantive feedback is the same. This was discovered by psychology research in the 1980s. In these experiments [Amabile], subjects were asked to judge various attributes[33] of the authors of book reviews. Participants consistently ranked reviewers who gave negative reviews as more intelligent than those who used positive adjectives to describe the review, even when controlling for the substantive matter and grammatical style of the review.

Not always the case, but technical reviewers are often prone to quickly find flaws and even faults in the work being reviewed without recognizing or acknowledging the positive elements or even offering constructive suggestions. Repeated instances of this behavior

---

[31] For a summary of studies that show the relative cost-effectiveness of catching software defects using reviews versus unit and system testing, see "Why Quality Pays" in [Humphrey].

[32] For a full discussion of the types of reviews using in software engineering environments, see [p. 31-44, Wiegers]. These include inspections, team reviews, walkthroughs, pair programming, peer deskcheck, passaround, and ad hoc review.

[33] Apart from intelligence, these included literary expertise, competence, kindness, fairness, likeability, and open-mindedness.

© 2004 Devabhaktuni Srikrishna

email: srikrishna <at> devabhaktuni.us

will discourage and demoralize those whose work is being reviewed, since they will only be viewed negatively by onlookers. It also creates a vicious cycle whereby a bad review causes authors to fear reviews, and they are not inclined to expose their own work to review in the future. People who subject themselves to reviews are legitimately at risk of looking stupid and exposed in front of colleagues and managers, giving them disincentives to build systems and structures that can be simply explained to and easily understood by others. If systems are to be understood by the team, technical reviews can't be allowed to be a painful or frightening experience, and instead the organization needs to cultivate an environment where reviews are sought after for their benefits.

The second barrier to review has to do with intellectual ownership, which is not necessarily bad. Without it, we would not see a number of contributions from motivated and talented individuals. This is just as much a part of engineering as it is science, wherein engineers take pride in their work. The protective instincts of intellectual ownership also lead the owner to shield their treasure from criticism, and prevent healthy debate that the owner decides is unnecessary.

Isaac Newton aggressively guarded his scientific discoveries from peer-review, and let the details slowly trickle out to his friends over the course of several decades. Apparently, he did this because he feared disputes from those whom he considered would offer inaccurate criticism[34]. He was right in many instances where less qualified and knowledgeable individuals offered incorrect critiques, but it illustrates the resistance to sharing that comes naturally to those who create. The period over Newton's life saw the formation of the Royal Society which permanently established peer-review as a cornerstone of scientific culture. As painful as it was, by proving its value time and time again, peer-review process or scientific results became an essential ingredient of the scientific endeavor.

The value of reviews in engineering environments has also been recognized almost universally by this point, and it may be hard to follow through in practice. The solution is to recognize and focus on the value of reviews, and mitigate concerns about the perceived risk of criticism.

The presence of an unbiased, credible leader who is committed to achieving understanding, and whose judgment can be relied upon is sometimes necessary to ferret out the right answer from the team. Leaders can be useful for,

1. resolving deadlock,
2. mediating and preventing things from getting out of hand,
3. keeping the discussion focused and on track, not allowing the group to stray too far from the topic under review,
4. helping identify the best solution, and convincing the entire team in the process.

If the reviewed and/or reviewers cannot agree, it's possible that the disagreement arose because one of them may be more competent that the other. In that case, it is the job of the senior leader recognize this and communicate the correct picture to others. It may be necessary for judgment to prevail in these cases, and this should ideally be someone who is respected by all parties in the review. Criticism that appears to come from senior

[34] See chapters six and seven of [Gleick], and especially the reference to how Newton regretted revealing his theories of light to the Royal Society and was angered by false counter arguments they attracted from Hooke [p. 85-89, Gleick].

© 2004 Devabhaktuni Srikrishna

email: srikrishna <at> devabhaktuni.us

sources, rather that solely from peers, can be perceived as a hit to the ego—it all depends on the individual personalities involved.

Third, technical reviews can be so boring that they can "put their audience to sleep," in which case they serve little purpose. Technical reviews can only be effective in as much as author or presenter is able to

1. draw attention to relevant areas,
2. appeal to the reviewers' intellect, starting from a high enough level and working down to the lower levels of the technical details.

As antithetical as it might be from an engineer's perspective, reviews actually benefit from a marketing-like approach to interest and engage their audience, and evoke the kind of intense scrutiny that is required. For instance, merely displaying blocks of code during the review can only result in simple syntax and coding style comments but not much more. On the other hand, staying at too high a level offers limited engagement for the reviewers. Engineers need to keep in mind the perspective of the reviewers when presenting code or design in order to make the most of the meeting. Likewise, the reviewers need to understand that they are being drawn in, and will need to ask the relevant questions to uncover further understanding or look in places that the coder/designer did not think to.

Fourth, the "tone" taken by reviewers making them sounds as though they are out of line, dictatorial, or unjustified can lead to things getting personal. Unlike oral communication, people tend to extrapolate the worst possible interpretation from written words or email, whether or not that was the intention. Likewise those whose work is being reviewed need to watch out not to take the suggestions personally, but to use them for coming up with the best solution possible and gain the most understanding. Extra effort to soften the communications helps to ensure reviews can remain productive.

Fifth, a common suspicion of a reviewer's desire to understand the details is that the reviewer has motivations to exercise control. There may be truth to this allegation in some circumstances. But it can also represent a fear of losing control by those who are under the review who are closer to the details in contrast to the reviewer who may be more distanced. Even people who are perhaps unqualified to design the sub-system under review need to understand the implications of that sub-system for the other pieces of the system. Once they understand the implications, they will be in a position to offer constructive feedback with regards to how the sub-system design and its interactions with other parts of the system. This understanding can be insightful and prevent mishaps or upset expectations. Even if the reviewed eventually go against the advice of the reviewers (consensus is preferable), the perspective was heard and understanding within the organization was increased in the process. That way, the reviewed have not lost control, but have gained insight through the eyes of others who may have a different perspective.

Who should be drawn into a technical review? Clearly those who have understanding of the sub-systems and its interacting components are valuable for the review. These are not limited to designers and engineers, but include system testers and product managers. However, some of the most insightful comments can come from people who are not familiar with the sub-system or system under review, simply because they offer a fresh perspective. This is why talented experts who have no direct relation to a project

© 2004 Devabhaktuni Srikrishna

email: srikrishna <at> devabhaktuni.us

are brought in to figure out what happened when a major bug is discovered or an accident happens (e.g. Space Shuttle Columbia and Challenger accident investigations). In effect, they end up increasing the overall understanding of why the system behaved the way it did, and conclude what can be done to fix it. If brought in earlier, these same people could have been equally insightful during design reviews in order to prevent the bug or accident. Leadership is all the more necessary to coordinate a productive discussion among a diverse group of reviewers.

Since the reviewers and those whose work is being reviewed are both stake-holders in the product, they need to agree that they will continue through discussion until they arrive at the right answer. This may take multiple review sessions, and everyone needs to commit to not giving up until their job is done. This assumes there is a right answer, and usually there is such a right answer modulo inconsequential variations in approach. It is important for each participant not to back down on what they recognize as important issues, but also let issues go so long as they are unimportant. Ultimately some debates cannot be resolved by those with opposing opinions, and in which case definitive answers must be had to determine which one is right or if both opinions are right. If all else fails, a moderator may be necessary to make final judgment, usually the lead architect or lead engineer of the product must play this role of a moderator by offering their judgment and communicating it to the team effectively.

Summary

1. Technical reviews are feared and can be discouraging due to the following factors,
    a. In order to be perceived as more competent in a group, reviewers are more inclined to offer negative criticism and find fault with those whose work is being reviewed. This makes it risky for people to subject themselves to review.
    b. Intellectual ownership makes people suspicious of criticism they may receive from others.
    c. Unless presented in an engaging and open manner, reviews often lose the interest of reviewers.
    d. Reviews can be derailed simply because the motivations of reviewers are misunderstood due to factors such as the tone of their voice.
    e. Questions asked during reviews can be misinterpreted as an attempt by the reviewer to exercise control, rather than with the intention of understanding the technical details.
2. Respected leaders who are committed to achieving system understanding are sometimes necessary in reviews to make judgment calls and bring participants onto the same page
3. Technical reviews can benefit from a reviewer audience with diverse background, and not just those who are directly related.
4. To make a technical review successful, participants need to be committed to achieving the right answer by permitting as much discussion and debate as needed to clarify relevant issues, without prematurely curtailing the review.

© 2004 Devabhaktuni Srikrishna
email: srikrishna <at> devabhaktuni.us

The AT&T Switching Collapse

On January 15, 1990, AT&T's national long distance network suffered a nine-hour outage. Estimates of uncompleted calls range from 5 million[35] to 70 million, and AT&T estimates it lost $60 million in lost revenue[36]. The secondary effects not accounted for include the loss to customers and damaged reputation of the AT&T brand.

The collapse was initially suspected to be the work of malicious hackers, it later became clear that a total of 114 devices forming the backbone of AT&T's long-distance network, known as 4ESS switches, went down in rapid succession causing the outage. The outage was triggered by a single switch in New York that was temporarily experiencing heavy load and reset itself, which would have been expected behavior for a switch experiencing this condition.[37]

Several types of messages are sent between the switches comprising the network, including those for call traffic (setup, tear-down) and network control. After the switch reset itself, it would have sent a message explicitly informing neighboring switches that it has come back online allowing the neighbors to update their state, before call traffic messages could be sent. However, a software upgrade to each of the 114 switches during the previous month allowed each switch to detect that another switch has come back online simply based on its sending of call traffic messages alone, without the explicit informational message. But, as part of the upgrade, a one-line bug was introduced in this upgrade, among millions of lines of software code running on the switch.[38]

An unanticipated consequence of this bug was that as the switch came back online and began to rapidly send call traffic messages, neighboring switches would incorrectly process the second message before completing their processing of the first message. This resulted in erasure of critical data, and induced a switch reset. By the same process, these resets triggered resets of their neighbors and so on, bringing the network to collapse.[39]

Finally after enduring a nine hour national outage, AT&T was able to bring the network back to normal by reducing the load on the network and later fixing the bug through another software upgrade.[40] This bug got introduced after significant testing prior to deployment of the software upgrade[41]. These included tests in simulated networks performed by AT&T[42]. The one-line coding error (bug) within a C program was later reproduced by AT&T through simulation[43].

---

[35] see [p. 14, Neumann]
[36] For a succinct description, see [Borisov]
[37] See [p. 211, Peterson]
[38] See [Burke].
[39] To read about the AT&T network collapse and other network collapses in the electrical grid and Internet, see [Gershenfeld].
[40] See [p. 14, Neumann]
[41] See [p. 123, Neumann]
[42] See [p. 214, Peterson]
[43] It was an extra `break` statement within an `if` clause nested in a `switch` clause—see [p. 14, Neumann]

© 2004 Devabhaktuni Srikrishna
email: srikrishna <at> devabhaktuni.us

In a repeat, the frame relay networks of AT&T and MCI-Worldcom, that are relied upon to carry mission-critical traffic over leased lines for American businesses ground to a halt in April, 1998 and August, 1999 respectively. While both networks operated with frame relay switches from different switch vendors, the root cause of both these incidents was shown to be switch software upgrades, each containing bugs that resulted in a cascading network failure.[44]

The presence of the bug brings up the following questions about the system:

1. How did the bug get into the software?
2. Why was the system designed without checks such that a chain reaction of (in this case switch resets) was permitted to cascade and propagate based on a single switch failure?
3. How can any other such bugs be discovered and caught in the switch software? Is post-development testing sufficient?

[44] See [Krause].

© 2004 Devabhaktuni Srikrishna
email: srikrishna <at> devabhaktuni.us

Sobig-F Computer Virus

On August 18, 2003, the Sobig-F computer virus flooded email servers worldwide with 200 million emails. It is hard to accurately calculate the loss of productivity due to this event, and estimates range from $1 billion [Leyden] to as high as $30 billion [Gaudin]. The virus took advantage of vulnerabilities inherent to the pervasive Microsoft Windows operating system and email client programs for Windows such as Microsoft Outlook running on networked PCs, unguarded user behavior, and the insecure SMTP email protocol.

The virus first appears as an email attachment that is executable in Windows (i.e. can be run as a program).[45] The first problem is that the email client allows (tempts) the user to "launch" the executable simply by clicking on it as soon as they read the email, without warning the user or preventing them from running it—these warnings have been introduced in some versions of operating systems and anti-virus software. After clicking on the attachment which triggers the virus to execute itself on the host computer, the second problem is that the virus has access to all files and processing resources without having to gain any permission by presenting credentials of any sort. So the virus then reads various files on the computer's hard disk harvesting it for email addresses. By compromising this ease of use, neither of these problems is as easy to exploit on many Unix-based operating systems such as Linux.[46]

Finally, the virus sends a similar email to each of the addresses by running code that acts as an email (SMTP) server. The steps involved in sending an email using the SMTP protocol are,

1. An email client representing the sender sends the email to its SMTP server.
2. The sender's SMTP server then sends the email to the SMTP server for the destination address.
3. The destination email client downloads the email.

The third problem is that by design, SMTP servers do not check the credentials or authenticity of any peer server trying to send email. Although use of secure extensions to email such as S/MIME that are supported by common email programs and servers might have prevented the propagation of the virus, secure alternatives don't serve a useful purpose unless adopted widely since email is a universal communication medium. So the virus which incorporates code that acts as an SMTP server is not prevented from sending email to the servers that represent the harvested email addresses. The virus, which began as an email, replicates itself in the email inbox of multiple recipients spreading the "infection."

Key questions about the design of the application, operating system, and protocols are brought up:

1. Why does the mail program permit easy execution of executables?
2. Why does the operating system permit any application to access resources without limit?

---

[45] The technical mechanisms exploited by the virus are explained in [Sophos].
[46] See [Granneman].

© 2004 Devabhaktuni Srikrishna
email: srikrishna <at> devabhaktuni.us

3. Why does the SMTP protocol not incorporate authentication of any sort?
4. Which organizations, individuals, and technologies are responsible for email security?

The Bhopal Tragedy

On the night of December 23, 1984, one of the worst, if not the worst ever industrial accident took place in Bhopal, India. A large explosion at a Union-Carbide pesticide manufacturing plant released several tons of a poisonous chemical, methylisocyanate (MIC), into the surrounding atmosphere over a city of 800,000 people.[48] A failure of containment and absence of emergency measures for the community contributed and exacerbated the effects of the accident, which through poisoning killed an estimated 4000 people and injured hundreds of thousands more.[49]

The accident was triggered by well understood chemical reactions when 500 liters of water leaked into a chemical tank containing 40 metric tons of MIC, which caused the chemical explosion that released the deadly chemicals. Although the cause of the water leak is not known[50], a likely trigger for the water leak was a maintenance procedure on December 2, 1984.[51]

Earlier in May 1984, a jumper was installed to connect two different vents that relieve excess pressure in the MIC tank. This was done in order to allow one vent to function while the other was blocked for repairs.[52] Filters connected to one of the vents were being washed with water as part of a "standard" procedure. On December 2, the jumper was opened to allow repairs on the other vent. Water likely flowed along the jumper to the other vent which was known to be leaky from prior work involving liquid nitrogen. Water then leaked back into the MIC tank through that vent. Early signs of trouble from temperature and pressure gauges were ignored due them being known as unreliable by plant employees Changes to the plant or the flow of various liquids within the plant over time was not tracked or logged by sensors.

The vents were designed to carry toxic vapors to a gas scrubber that would have neutralized them, but the scrubber itself had been turned off as part of maintenance procedure described as routine. Even other fail-safe measures, such as a flare tower designed to contain and mitigate the release of the vapors, were also disabled. Despite multiple fail-safe measures designed to ensure safety, true operational safety was missing. This seems to be a pattern in most chemical accidents.[53]

Several aspects of the design and operation of the plant are called into question:

1. When the plant was located in close proximity to an urban area, why was a large MIC tank with potential for large reactive explosions used, as opposed to use of multiple smaller, more isolated tanks?

---

[48] See [Reisch].
[49] See [Lee].
[50] For a discussion of the social, economic, and political circumstances surrounding the tragedy, and an analysis of alternative hypotheses for the causes, see [p. 259-268, Chiles].
[51] See [Kirkland].
[52] See "Toxic Vapor Leak, Bhopal, India" [p. 403, Schlager].
[53] See [Merritt].

© 2004 Devabhaktuni Srikrishna
email: srikrishna <at> devabhaktuni.us

2. Was the installation of the jumper reviewed and undertaken with careful consideration for its future consequences?
3. Why were so many fail-safe measures out of order?
4. Given the known danger of water mixing with MIC, why was there no alert that water was leaking into the MIC tank?
5. Could it be that the existence of multiple fail-safe mechanisms made the plant operational personnel have a false sense of security and neglect operational safety?

© 2004 Devabhaktuni Srikrishna
email: srikrishna <at> devabhaktuni.us

Bibliography

Tassey, G. 2002. *The Economic Impacts of Inadequate Infrastructure for Software Testing,* Gaithersburg, Maryland: National Institute of Standards.
< http://www.nist.gov/director/prog-ofc/report02-3.pdf>

Wikipedia. 2004. “Computer Bug.”
< http://en.wikipedia.org/wiki/Computer_bug >

Ask Oxford. 2004. “Was the first computer 'bug' a real insect?,” Oxford Dictionaries.
< http://www.askoxford.com/asktheexperts/faq/aboutwordorigins/bugs >

Columbia Accident Investigation Board. 2003. *Report Volume I,* Washington D.C.: National Aeronautics and Space Administration.
< http://www.caib.us/news/report/pdf/vol1/full/caib_report_volume1.pdf>

U.S.-Canada Power System Outage Task Force. 2004. *Final Report on the August 14, 2003 Blackout in the United States and Canada:* Causes and Recommendations, Princeton, N.J.: North American Electric Reliability Council.
<http://www.nerc.com/~filez/blackout.html>

Leyden, J. “Sobig-F is dead,” *The Register,* 10 September, 2003
<http://www.theregister.co.uk/2003/09/10/sobigf_is_dead/>

Gaudin, S. “2003 'Worst Year Ever' for Viruses, Worms,” *Internet News,* 23 December, 2003.
<http://www.internetnews.com/dev-news/article.php/3292461>

Sophos. 2004. “Sophos virus analysis: Win32/Sobig-F.”
<http://www.sophos.com/virusinfo/analyses/w32sobigf.html>

Granneman, S. 2003. “Linux vs. Windows Viruses”, *SecurityFocus:* 2 October, 2003.
<http://www.securityfocus.com/columnists/188>

Borisov, N. 2004. “AT&T Failure of January 15, 1990,” 21 June, 2004.
<http://www.cs.berkeley.edu/~nikitab/courses/cs294-8/hw1.html>

Burke, D. 1995. “All Circuits are Busy Now”, November, 1995.
<http://www.csc.calpoly.edu/~jdalbey/SWE/Papers/att_collapse.html>

Gershenfeld, N. 2000. “Everything, the Universe, and Life” *IBM Systems Journal*: 39: 3 & 4.
<http://www.research.ibm.com/journal/sj/393/part3/gershenfeld.pdf>

Krause, J. K. 1999. “How not to handle a network outage” *The Industry Standard,* August 23, 1999, http://www.cnn.com/TECH/computing/9908/23/network.nono.idg/

© 2004 Devabhaktuni Srikrishna

email: srikrishna <at> devabhaktuni.us

Reisch, M. S. 2004. “TWENTY YEARS AFTER BHOPAL,” Chemical & Engineering News, 82 (23): 19-23.
<http://pubs.acs.org/cen/coverstory/8223/8223bhopal.html>

Lee, J. L. 2004. “Bhopal Disaster,” *Trade and Environment Database*, 233.
<http://www.american.edu/TED/bhopal.htm>

Kirkland, Lane. 2004. “The Report of the ICFTU-ICEF Mission to study the causes and Effects of the Methyl Isocyanate Gas Leak at the Union Carbide Pesticide Plant in Bhopal, India, on December 2nd/3rd 1984,” *International Confederation of Free Trade Unions, International Federation of Chemical, Energy, and General Workers Unions,* 26 June, 2004.
<http://www.bhopal.net/oldsite/documentlibrary/unionreport1985.html>

Merritt, C. W. 2004. “Chemical Process Safety at a Crossroads,” Environmental Health Perspectives, 112 (6).
<http://ehp.niehs.nih.gov/docs/2004/112-6/editorial.html>

McCarthy, J. 1996. “CHERNOBYL,” *FREQUENTLY ASKED QUESTIONS ABOUT NUCLEAR ENERGY,* 26 January, 1996.
<http://www-formal.stanford.edu/jmc/progress/chernobyl.html>

Rhodes, R. 1993. *Nuclear Renewal.* New York: Penguin.
Excerpt of Chapter 5, “Chernobyl” provided online at
<http://www.pbs.org/wgbh/pages/frontline/shows/reaction/readings/chernobyl.html>

Berndt, M. R. 2001. “DoD News: Briefing on V-22 Accident by Maj. Gen. Berndt,” *United States Department of Defense News Transcript,* 5 April 2001.
<http://www.defenselink.mil/news/Apr2001/t04052001_t405mv22.html>

Dailey, J. R. 2001. “Report of the Panel to Review the V-22 Program,” *Department of Defense*, 30 April 2001.
<http://www.fas.org/man/dod-101/sys/ac/v22-report.pdf>

Neumann, P.G. 1995. *Computer Related Risks,* Boston, MA.: Addison-Wesley Professional.

Peterson, I. 1996. *Fatal Defect,* New York: Random House.

Schlager, N. 1994. *When Technology Fails: Significant Technological Disasters, Accidents, and Failures of the Twentieth Century,* Farmington Hills, MI: Gale Group.

Chiles, J. R. 2001. Inviting Disaster: Lessons From the Edge of Technology, New York: Harper Collins.

Wiegers, K. E. 2001. *Peer Reviews in Software*, Boston, MA: Addison-Wesley.

Paulk, M.C., et. al. 1995. The Capability Maturity Model: Guidelines for Improving the Software Process, Boston, MA: Addison Wesley.

© 2004 Devabhaktuni Srikrishna
email: srikrishna <at> devabhaktuni.us

Columbia Accident Investigation Board. 2003. *Report Volume I,* Washington D.C.: National Aeronautics and Space Administration.
< http://www.caib.us/news/report/pdf/vol1/full/caib_report_volume1.pdf>

Vaughan, D. 1996. *The Challenger Launch Decision: Risky Technology, Culture, and Deviance At Nasa,* Chicago, IL: The University of Chicago Press.

Levenson, N. 1995. Safeware: *System Safety and Computers*, New York: Addison-Wesley.

Columbia Accident Investigation Board. 2003. *Report Volume I,* Washington D.C.: National Aeronautics and Space Administration.
< http://www.caib.us/news/report/pdf/vol1/full/caib_report_volume1.pdf>

Humphrey, W. S. 2002. *Winning with Software: An Executive Strategy*, Boston: Addison-Wesley.

Wiegers, K. E. 2001. *Peer Reviews in Software*, Boston, MA: Addison-Wesley.

Amabile, T.M. 1983. “Brilliant but Cruel: Perceptions of Negative Evaluators,” *Journal of Experimental Social Psychology*, 19:146-156

Glieck, J. 2003. *Isaac Newton,* New York: Random House.

© 2004 Devabhaktuni Srikrishna
email: srikrishna <at> devabhaktuni.us

Suggested References for the Case Studies

System Bug	Industry	Suggested Reference	Hardware Failure	Software Failure	Network Cascade	Operator Induced
Columbia Accident February 1, 2003	Space Travel	The Columbia Accident Investigation Report	●			
Bhopal Tragedy December 23, 1984	Chemical Process	The Report of the ICFTU-ICEF Mission to study the causes and Effects of the Methyl Isocyanate Gas Leak at the Union Carbide Pesticide Plant in Bhopal, India, on December 2nd/3rd 1984	●			●
Chernobyl Catastrophe April 26, 1986	Nuclear Power	Chernobyl, Nuclear Renewal				●
AT&T Switching Collapse January 15, 1990	Telecom-munications	Everything, the Universe, and Life, IBM Systems Journal			●	
Ariane-5 Software Bug (ESA Report)	Satellite Rocketry			●		
Sobig-F Computer Virus August 18, 2003	Personal Computing	Sophos virus analysis: Win32/Sobig-F		●	●	●
Patriot Software Bug	Missile Defense			●		
Therac-25	Medical Devices			●		
Osprey V-22 Crash December 11, 2000	Aviation	DoD News: Briefing on V-22 Accident by Maj. Gen. Berndt	●	●		
Blackout in the United States and Canada August 14, 2003	Power	Final Report on the August 14, 2003 Blackout in the United States and Canada	●	●	●	●
Hyatt Regency Catwalk Collapse (Petroski)	Civil Engineering		●			
Jack in the Box E. Coli outbreak	Food Processing	Fast Food Nation	●			●

© 2004 Devabhaktuni Srikrishna
email: srikrishna <at> devabhaktuni.us