Software and System Reliability Considerations for the 21st Century

Dr. Samuel J. Keene

4/19/00 3:34 PM

Y2K and IT shocks

The infamous bug has come and passed. This bug came about because programmers' thought horizon did not reach into the new millenium. When the clock rolled over into the 21st century, software that thought the date was 1900 could disastrously fail. Fortunately, its impact was less than forecasted. Problem prevention has cost about 100B, which is significantly less than many forecasters predicted. Hopefully programmers, or information technology (IT) people, have learned from this traumatic experience. Y2K has sensitized the general public to the greater potential impact of software failures.

Imagine what might have happened if this problem caught us unawares and took its toll. Life as we know it could have been shocked, like the oil embargo of 1974. Many processes would have come to a screeching halt.

There are five exposure areas related to the Y2K type of problem:

1. Operating system embedded dates
2. Application code embedded dates
3. Dates in the embedded software
4. Data (with potential command codes such as 9999 used as a program stop)
5. Embedded or interfacing commercial off the shelf (COTS) software

These all had to be examined to assure the software had date robustness. This type of problem will be avoided if the program requirements are fully developed. The program action must be defined in terms of what the program "should do" as well as what it "should not do". The program requirements should also enumerate the system-desired behavior in the face of operational contingencies. Programs do what we tell them to do. They are scripts to carry out our directions. Our responsibility is to accurately tell programs what we want them to do while removing any logical ambiguities and uncertainties.

In essence, people execute programs every day, in their everyday lives, as they follow someone's directions or try to follow road signs. These directions or road signs are a navigation script. A failure occurs whenever there is ambiguity or someone is somehow misled and goes astray. The directions are clear in the mind of the person who writes them down, but too often, the directions are unclear, confusing or misleading to the person attempting to follow them. This same type of disconnects plague computer programs.

We now have an Information Technology layer that surrounds our life and which we depend upon. Internet use is pervasive and growing. The Internet related companies have rocketed in value over the past several years. Alaska, Colorado, Maryland, Utah and New Hampshire have between 41 and 52% of their population on line. When the internet works, it is multiplies our effectiveness. But when it fails, we are left hapless.

Our offices, businesses and personal lives depend upon computer communications. We feel stranded when that service is not available, even for short periods. The most notorious outage was the 18 hour outage experienced by AOL users in August 1996 [1]. This resulted from an undetected operator error occurring during a software upgrade. The cause of failure was thought to be solely due to the upgrade. This confounding misled the analysts tracking the problem. Thus 18 hours were required to isolate and correct the problem.

More importantly, many of our common services depend upon computer controllers and communications. This includes power distribution, telephone, water supply, sewage disposal, financial transactions, as well as web communications. Shocks to these services disrupt our way of life in our global economy. We are dependent on these services being up and functioning. The most critical and intertwining IT services that envelop our lives, either directly or indirectly are:

<u>Critical deployment areas or high impact areas;</u>

1. Nuclear power
2. Medical equipment and devices
3. Traffic control (air, train, drive-by-wire automobiles, traffic control lights)
4. Environmental impact areas (eg., smoke stack filtration)

5. Business enterprise (paperless systems)
6. Financial systems
7. Common services (water, sewer, communications)
8. Missile and weapons systems
9. Remote systems (space, undersea, polar cap, mountain top)
10. Transportation (autos, planes, subways, elevators, trains, …)

Problem sources

- The main line code usually does its job. **Breakdowns typically occur when the software exception code** does not properly handle an abnormal input or environmental conditions—or when an interface doesn't respond in the anticipated or desired manner. problems resulted when programs failed to gracefully handle the millenium clock rollover. The infamous "Malfunction 54" of Therac 25 radiation therapy machine fried some patients because the software controls did not successfully interlock the high-energy operating system mode. This allowed patients to be exposed to potentially lethal radiation levels.

- **The major cause of software or system problems lies in System Management problems**. These are requirements deficiencies or interface defects. The Martian probe failed in the Fall of 1999 [2]. This failure was due to system management problems. There was a break down in communications between the propulsion engineers and the navigation scientists, at a critical point in the design cycle. The newspaper headlines publicized another program communication problem. NASA and its contractor had unconsciously commingled metric and British dimensional units. The resulting tower of Babel situation helped misguide the $125 million dollar space probe.

  System Management problems are truly communication problems. This author has been a life-long student of system failures (and successes). Today, for the most part, hardware is reliable and capable. Almost all-significant system problems are traceable to a communications break down or too limited design perspective on the programmer or designer's part. The problem is an example of the programmer's lack of an adequate perspective.

- **Beware small changes can have grave consequence**. There is significant error proneness in making small code changes[3]:

Defect rate:	1 line	50%
	5 lines	75%
	20 lines	35%

The defects here are any code change that results in anomalous code behavior, i.e., changes causing the code to fail. Often, small changes are not given enough respect. They are not sufficiently analyzed or tested.

For example, DSC Communications Corp., the Plano Texas Company, signaling systems were at the heart of an unusual cluster of phone outages over a two-month period of time. These disruptions followed a minor software modification. It was reported, "Three tiny bits of information in a huge program that ran several million lines were set incorrectly, omitting algorithms – computation procedures – that would have stopped the communication system from becoming congested, with messages … Engineers decided that because the change was minor, the massive program would not have to undergo the rigorous 13 week (regression) test that most software is put through before it is shipped to the customer". Mistake!

The Ariane 4 code worked fine but it did not scale up satisfactorily to Ariane 5. Ariane 5 design changes were not properly handled by the original code. This is an example of code reuse or using Commercial Off-The-Shelf code (COTS). Reusing COTS code is an increasingly popular way to reduce development cost and time. The code does not have to be reinvented but it needs to be fully analyzed and tested for its new application, paying particular attention to any changes in the hardware or the system from its predecessor application.

Software reliability, especially the system management problem, is the long pole in the tent for developing new systems. It impacts development time, cost and system reliability. A second, increasing problem lies in Electro-Magnetic Interference (EMI). It is an increasingly pervasive threat. This EMI threat is often intermittent, confounding one's ability to trace it down to its root cause and resolve. An example of this can occur during servicing of computer peripherals. The maintenance engineer may service the equipment while it is operating with the doors open. This removes the EMI equipment enclosure. Radiation is broadcast impinging on other operating equipment that potentially disrupts their operations. Then the door is shut on the first

equipment and the radiation problem disappears for the neighboring equipment. Any malfunction, that they experienced, now disappears.

Electro-Magnetic Interference (EMI) is an increasing threat

Equipment reliability is increasingly compromised from electrostatic noise in our environment. The noise sources come from broadcasted energy from direct communication sources (allocated frequency bands) as well as coincidental broadcasted electrical noise from electrical and computer equipment. The broadcast spectrum is fuller and extending up to higher frequencies. There is a significant amount of such energy being transmitted in the Gigahertz range. At 1 GHz, the wavelength is 1/3 meter, or 30 cm. Electronic equipment can be susceptible to a broad range of impinged frequencies.

Although analog electronics are usually more susceptible than digital, even high speed digital electronics can be affected by fields specified as industrial levels, i.e., 10 V/m. Susceptibility in analog electronics often occurs as a result of the modulation of the impinged RF field. That is, a high frequency transmitter can couple significant energy into a product via its apertures (radiated EMI) or its cabling (conducted EMI). Electronic equipments are susceptible at significant fractions of a wavelength (>1/20 wavelength). The transmitted frequency is typically well beyond the bandwidth of most analog electronics. However, if this frequency is modulated at a low frequency (in the kHz range), the modulation frequency is well within the bandwidth of nearly all analog electronics. Thus, the high frequency allows the energy to couple into the product, but it is the low (modulation) frequency that is most often the culprit in an EMI problem.

EMI problems are subtle. For example, one manufacturer's equipment experienced excessive failures in a Paris operating room. On threat of ejection, the supplier's equipment engineer was sent to investigate the problem. This engineer discovered a medical doctor using a cell phone in the operating room. Cell phones operate in the gigahertz range at small enough wavelengths to penetrate normally incidental openings in the surrounding medical equipment chassis and interfere with operations. The equipment covers had to be redesigned to harden them. Higher frequency operation typifies today's systems, making them increasingly vulnerable to Electro-magnetic emissions. Electromagnetic immunity is an ever more important requirement for reliable equipment and systems as radiation

sources (wireless phones, computers) are increasing dramatically. Plus their operating frequencies are higher and thus more intrusive.

Proactive System Reliability Initiatives

- Strong and systematic focus on requirements development, validation, and traceability with particular emphasis on System Management aspects. Full requirement development also specifies things the system should not do as well as those desired actions. Heat seeking missiles should not boomerang. Quality Functional Development (QFD) is one tool that helps assure requirement completeness, as well as fully documenting the requirements development process.
- Plan back up and contingent capabilities for critical failure events.
- Study near misses to further understand how to prevent or mitigate failures, just like we do today in aircraft near misses.
- Institutionalize a "Lessons Learned" data base and use it to mitigate potential failures during the design process.
- Perform a potential failure modes and effects analysis to harden the system against abnormal conditions.
- Perform root cause failure analysis to trace problems down to the underlying cause. Then parse the remainder of the code to ferret out any other faults of the same structure.
- Leverage discovered failures using the Defect Prevention Process, as labeled by IBM [4]. Study and profile the most significant failures. Understand the constructs that allowed the failure to happen. Y2K is an example of a date defect. One should then ask if other potential date roll over problems exist? The answer is yes. For example, the Global Positioning System (GPS) clock rolled over August 22, 1999. On February 29, 2000 the Leap Year exception has to be dealt with. Here one failure can result in identifying and removing multiple faults.
- Develop more effective testing techniques to cover the extensive number of program paths covering the possible input conditions.
- Think defensively. Examine how the code handles inappropriate inputs. Design to mitigate these conditions.
- Perform fault injection into systems, as part of system development, to speed the maturity of the software diagnostic and fault handling capability.
- Build in diagnostic capability. Systems' vulnerability is an evidence of omissions in our designs and implementation. Often we fail to include

error detection and correction (EDAC) capability as high level requirements.

Our electrical power system has is managed over a grid system, which has built-in atomic clocks. These clocks enable EDAC by monitoring time to better than a billionth of a second. This timing capability's primary purpose is for proper accounting of power usage by timing the link the various power sources to the end-user. This also provides a significant EDAC tool for failure events that used to appear simultaneous, can now be differentiated to find the first domino. These precise clocks can also time reflections in long transmission lines to locate the break in the line to within 100 meters.

- Design in opportunistic times to restart and refresh the software during its deployment. This cleans up the system's operational environment by clearing up memory leaks, resource contention, emptying command stacks, for example. This pro-active initiative averts so-called "long latency" failures. This improves the reliability of the software as observed by the user. It is a trade-off. The system takes some planned, but brief outages, to reduce the risk of unplanned outages. This is a common action PC users take. The PC slows down and starts to behave abnormally. The prudent PC operator saves the active data and reboots the system, restoring its normal operation.

The Future

Most companies have back up capabilities for their mission critical applications. For example, they use battery back up and Uninterruptable Power Systems (UPS). This compares to answering machines and digital clocks using battery back up. In critical control situations, each functional department typically uses cell phones to back up the phone line system. System ledgers and journals of transactions are also maintained in real time and are backed up daily. Mission critical data or operations are stored or executed in geographically diverse locations to safeguard against local climate catastrophies.

On the positive side, has quickened our awareness of systems vulnerability. We strive more to build in checks and safeguards. There is also more focus on developing and following best practices, which lead to better reliability. These practices need to be codified and program compliance assured at critical review points in the development cycle.

The biggest development challenge may well be improving our test capability to deal with the burgeoning amount of high function code (Object oriented) that is being developed. Testing such code, with its explosive number of operating paths is a great challenge. Because of this, safety critical code will probably be developed using more formal methods or clean room processes. This development method tests and assures the contingent path conditions as the code is developed. The code and the checking and exception handling are developed simultaneously.

Most of the US was positioned to deal with . There were interruptions to some services but these interruptions were truly limited in time and extent. Note, that not all problems occurred on January 1, 2000. Some will key to other days of the year beyond January 1, 2000.

The rest of the world is not as ready to deal with the millenium calendar problems. This, in turn, can still pose problems worldwide. We are grateful that defense and missile systems gracefully slept through the onset of the new millenium. (Remember what happened due to y1k? … The dark ages.)

Where the world is most weak is <u>not</u> the random failure modes but in the possibility of an organized attack directed at nations (or companies) economic, environmental or enterprise systems. It is conceivable that a well-organized attack could defeat the redundancies. This means that the potential threat of terror must also be considered in the development of 21st century systems.

<u>Our Reliability Mission</u>

Mission critical systems will demand broader attention be paid to all system aspects. The notion of reliability, or more to the point, the definition of failure will expand. Precluding failures will include managing system security, robustness, safety, data privacy, fault mitigation and recovery. The greater reach of software and systems, into the critical areas of our lives, will demand more internal "mistake proofing" and hardening against adverse external factors. System users will increasingly demand their systems behave better (less damaging and easier to recover) under failure conditions. How gracefully critical systems fail will be as important as how much functionality they provide. It is an exciting challenge for those of us who will help carry out and facilitate this role.

References

1. Ramstad, Evan (AP), “Online expectations greater than delivery”, Orange County Registrar, August 9, 1996, Business section.
2. Edwards, William, “Lessons Learned from 2 Years Inspection Data”, Crosstalk Magazine, No. 39, Dec 1992, cite: Weinberg. G., “Kill That Code!”, IEEE Tutorial on Software Restructuring, 1986, p. 131.
3. Oberg, James, “Why the Mars Probe Went off course”, IEEE Spectrum, December 1999, pp. 34-39.
4. Mays, R., Jones, C., Holloway, G., and Studinski, D., “Experiences with Defect Prevention,” IBM Systems Journal, Vol. 29, no. 1, 1990.