Hazardous Software Development

Corey P. Cunha
Savard Engineering

Developing safety-critical software is often an extremely complicated process, and if managed incorrectly could have the tendency to cause more harm than good. In order to deal with the challenge of writing safety-critical software, certain considerations must be followed. Different case studies will be used in this article to illustrate points about the ethics standards, hazard identification challenges, and aftermath management techniques needed to effectively manage the development and deployment of safety-critical software.

Safety-critical systems are important in everyday life and are used to manage difficult tasks that may otherwise be impossible to do. Many industries use systems that pose potential hazards to the general public, such as systems designed for the aviation and social engineering industries. When systems are developed for use in hazardous situations, development strategies and ethics standards must change to fit the needs of the project in order to ensure the safety of employees and the general public. As the demand for these systems continues to grow, standards and strategies developed to keep safety-critical software safe will continue to evolve.

Case Studies

To illustrate certain points, incidents involving failures in Union Carbide plants, the Patriot Missile Defense System, Iran Air Flight 655, and Therac-25 will be analyzed.

Union Carbide

Around midnight on December 2, 1984, a large amount of water was introduced into a Methyl Isocyanine (MIC) storage tank at Union Carbide's Bhopal, India pesticide plant, producing one of the worst industrial accidents in history. Once mixed with water, the MIC produced a vapor that escaped through the ventilation system into the atmosphere. Quickly afterward, the MIC began to decompose into many different components, including cyanide gas, and "... the immediate death toll from the cyanide release was in excess of 2,000 people" [1]. In the years to follow, the total death and injury counts ranged in the hundreds of thousands.

Though explanations vary, most attribute the disaster to a combination of human error and faulty safety systems. The system also lacked software safeguards that would prevent water mixture if certain hardware safeguards were not in place.

Patriot Missile Defense System

In 1991, a software design error in the Patriot Missile Defense System caused the deaths of 28 U.S. soldiers when a missile was fired at the wrong destination. After investigation, it was determined the error was due to differences in the clocks of the internal missile system and the global system, resulting in a miss when intercepting an incoming missile.

Iran Air Flight 655

Iran Air Flight 655, a civilian air flight traveling on an approved flight path, was shot down by a U.S. guided missile cruiser on July 3, 1988. A U.S. Navy investigation discovered the plane was shot down because of both a bug in the Advanced Electronic Guidance and Instrumentation System (AEGIS) aboard the craft along with bad decision-making skills by the commanding officer.

At no time did the AEGIS system fail completely; the order to fire was given by the commanding officer because of operator error and lack of judgment. Once the aircraft was identified by AEGIS, the system then proceeded to ask the aircraft for its friend or foe (FOF) status. The commercial aircraft then responded with the code *commair*, identifying itself as a commercial aircraft. After trying to confirm the FOF status with repeated radio communication, warning signals were given to have the aircraft change course.

Without any response from the aircraft, AEGIS was instructed to read the FOF signal from the plane again. This posed a problem as AEGIS could not read the FOF signal from the same plane twice without rebooting, causing the system to read the signal of a fighter jet stationed at a nearby air base. With a new signal indicating a military aircraft, the commander issued the order to fire.

Therac-25

Between June 1985 and January 1987 Therac-25, a Medical Linear Accelerator designed by Atomic Energy of Canada Limited (AECL), malfunctioned multiple times. These malfunctions resulted in the deaths of three patients, with many more being injured.

Unlike previous inventions by AECL, the Therac-25 did not use proven hardware with interlocks to ensure safety. Instead, Therac-25 relied completely on software modified from older systems in order to reduce costs [2]. Because of this design choice, there was no safety protocol when the software failed.

Based on user feedback, AECL modified the software before the initial release to include a quick re-entry method that allowed the operator to skip re-entry of certain treatment information [2]. This step was initially used to ensure the patient was getting the correct dosage prescribed. However, the change was not thoroughly tested, and as a result a software bug was never found.

If used extremely quickly, the system would generate an error and subsequently notify the operator of a cryptic error message. After resetting the device to clear the error, the machine would not indicate any dosage was applied, but the patient would receive a hazardous overdose of electrons.

Identifying Hazards

Hazard analysis, though important in every software engineering discipline, has an extremely pronounced role when dealing with safety-critical systems. There are several methods to perform hazard analysis, but before analyzing the hazards they must be identified.

As stated in [3], "There does not appear to be any easy way to identify hazards within a given system." Because of this fact, many identification methods are often used in order to find as many hazards as possible in a given system. The primary techniques used in this process are the Delphi Technique and joint application design (JAD).

Delphi Technique

The Delphi Technique, developed by the

Rand Corporation in the early 1980s, is an identification technique that aims at gathering different ideas from different geographical locations.

The advantages of this technique are numerous. The technique requires anonymity, reducing the chance one voice will be heard over the opinions of others. This technique also eliminates the need for participants to congregate, removing the stress of arguments between participants and the struggle of a group reaching a consensus.

Even with its advantages, the Delphi Technique can be slow to administer. The technique uses questionnaires, polls, and other traditional information-gathering techniques. Because of this limitation, the Delphi Technique calls for multiple iterations, slowing down the process. The process could also be slowed by the amount of workload on the moderator.

JAD

JAD is a hazard brainstorming system pioneered by IBM. Generally, JAD meetings are used in order for a group of people to reach an agreement on what types of hazards could occur and how those hazards could affect the system or a person's quality of life [3].

JAD development, though fast and thorough, was originally designed for other uses then adapted to be used in hazard identification. Because of this, JAD does not seem to identify hazards as well as other development styles. JAD also forces a consensus on the people involved in the process, which is often difficult to reach.

Analyzing Hazards

Hazard analysis is the process of examining all identified hazards, with the goal of locating the points of failure which cause the hazards. There are two different types of analysis: inductive, and deductive techniques. Inductive techniques attempt to start with answers and generate the questions during the analysis process, while deductive techniques look for answers to questions that are pre-defined. This article will focus only on inductive techniques, though, including event tree analysis, failure modes and effects analysis, hazard and operability analysis, and fault tree analysis.

Fault Tree Analysis

Fault tree analysis focuses on one hazard and works to find any events that could trigger the hazard to occur. The hazard in question is placed at the top of the tree with each subsequent event that could trigger the main hazard underneath it.

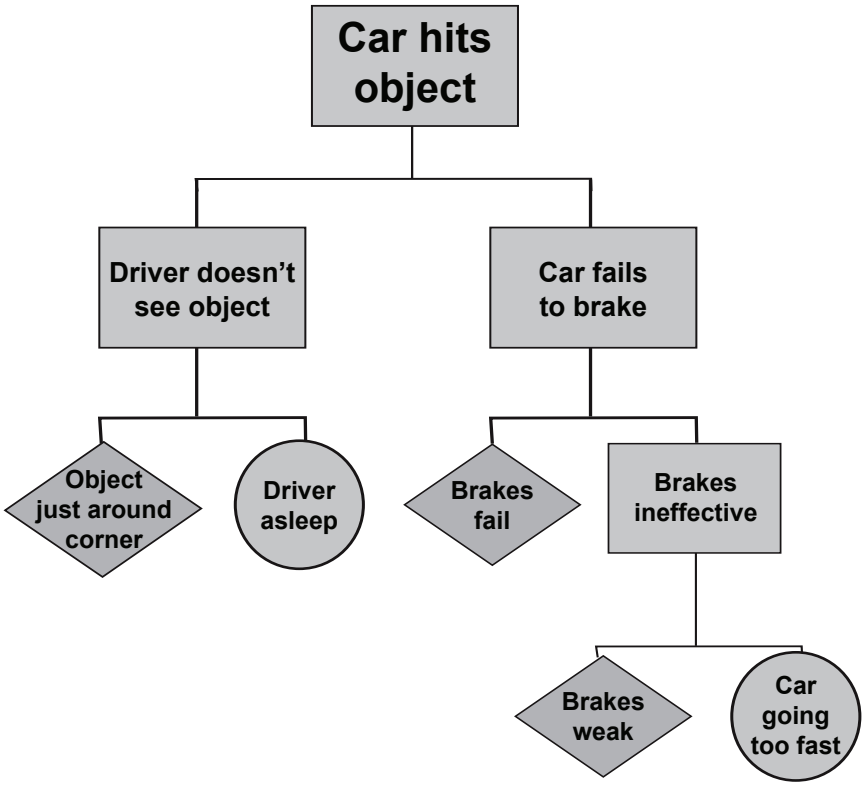


Figure 1: *A Sample Fault Tree Analysis Document*

Figure 1 shows an example fault tree diagram of a car where a vehicle collision is the final outcome. The diagram identifies many points of failure, such as the brakes and the driver's ability. While driver ability cannot be controlled in design, the brake design can now be prioritized to ensure safety.

A fault tree diagram can serve many purposes, the most important being the ability to determine different factors to a system failure. Fault tree diagrams can also be used during all the phases in a software life cycle, reducing the chances of missing a hazard after the planning stages.

The main downside to fault tree analysis is the high cost of producing documentation. Therefore, it is rarely used outside of high-risk situations [3].

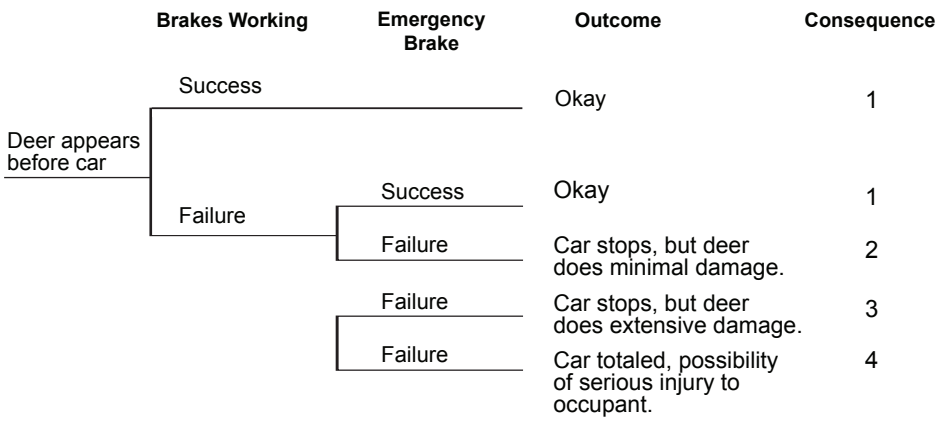
Event Tree Analysis

Event tree analysis has many similarities to fault tree analysis, but instead focuses on finding the outcomes of a system failure. Figure 2 shows an example of an event tree diagram for a car brake system. This specific example looks at a braking hazard, with the main brake and emergency brakes being the points of failure. Branching at each failure point, system developers can see the outcome of almost every failure situation.

Hazard and Operability Analysis

Hazard and operability analysis (HOA), or operating hazard analysis, is a hazard

Figure 2: *A Sample Event Tree Analysis Document*



analysis technique that requires attention during all stages of the software engineering development cycle. By using HOA, a project manager can ensure the maximum number of hazards are identified and corrected before product release [3].

HOA uses two different steps in order to analyze hazards. First, each hazard is either classified into being a human hazard or an environmental hazard. After grouping the hazards, they are analyzed further to generate a list of problems that can result from each specific hazard. Each point in the process is then checked to see if it is a safety-critical concern. If deemed safety-critical, the hazard is analyzed to determine how to minimize the possibility of its occurrence.

This type of hazard analysis could have been used for the system at Union Carbide. By searching for potential hazards throughout every step of the design process, the development team may have found more points of failure in the system and designed safeguards around these failure points.

Failure Modes and Effects Analysis

Failure modes and effects analysis was first developed in 1984 by the Department of Defense as a measure to prevent hazardous failures of critical systems. This method uses a table structure to take every component of the system and evaluate different failure modes for that component. This type of analysis also is used to find [3]:

- The effect of component failure.
- The cause of component failure.
- The occurrence chances.
- The severity of the problem.
- The probability of detecting the problem before the hazardous situation occurs.

Figure 3 shows an example of a failure modes and effects document based around a vehicle tie bar bracket. By analyzing all possible failure modes, designers of the tie bar bracket can see that invalid specifications of certain parts could cause the bracket to fail.

Examples

The blame for Bhopal was primarily placed on the shoulders of the developers of the system, not on the operators themselves [1]. Several ways are listed that the development team could have been better prepared for when working on a safety-critical system. Some points described in [4] include:

- Setting learning objectives for the developers, such as:
  - Assigning a design manager to ensure safety precautions are taken.
  - Creating a safety program to oversee the project.
  - Conducting random audits of software.

"When a safety-critical system does fail, it then becomes the responsibility of the company to follow up and take charge of the problem."

- Making sure employees are receiving education and not *shrugging off* their responsibilities.
- Learning from past mistakes of system failures.
- Including safety-critical engineering topics into software engineering courses.

A couple reasons why safety is often given second-place status to other factors in design is provided in [5]. Because safety issues often are ignored or unknown, these issues generate a substantial amount of risk to product success and personal safety. These are risks outlined in [3] which must be taken into consideration when doing both hazard identification and analysis. Some other risks include the following:

- Developers not conforming to the safety engineering processes due to time restraints.
- Developer lack of safety-oriented training.
- Lack of automated safety shutdown procedures in an emergency situation.

Figure 3: *A Sample Failure Modes and Effects Analysis Document for a Tie Bar Bracket*

Failure Mode	Effect of Failure	Cause of Failure	Occurrence	Severity	Probability of Detection
Bracket fractures	Stabilizing function of tie bar removed. All engine motion transferred to mountings.	Inadequate specification of hole-to-edge distance	1	7	10
Bracket corrodes	As above	Inadequate specification for preparation of bracket	1	5	10
Fixing bolts loosen	As above	Bolt torque inadequately specified	5	5	8
As above	As above	Bolt material or thread type inadequate	1	5	10

Conclusion

Had the design team at Bhopal used one of these development strategies, they could have been more prepared for the conditions which caused the 1984 disaster. If safety education and safety-minded development were taken into account earlier in the design process, the system may have prevented the operator from mixing the solution unless the certain safety conditions were met.

Hazard identification and analysis is extremely important in any safety-critical system, allowing for potential hazards to be found and analyzed before they cause problems in a final product.

These steps also go hand-in-hand with verification and validation (V&V) steps defined by the Institute of Electrical and Electronics Engineers (IEEE) standards document 1012. This document defines guidelines on how to first identify and manage hazards and risks. V&V steps ensure that problems are documented and then followed through, resulting in safer systems.

Codes of Ethics

With the amount of safety-critical systems currently in use, ethics play a larger role in software engineering today than they ever have in the past. The Computer Society, in cooperation from the IEEE, put together a set of ethics codes which all software engineers and designers should follow in order to assure software quality and safety. The standards relating to hazardous development discussed in [4] include, but are not limited to:

- Assuring that software engineers working on a project take responsibility for the code they write or decisions they make. For example, AECL never revealed the developer of the Therac-25 system; therefore, there was never a person to question about the software issues. This is often hard to enforce, as the original engineer may not be part of the company anymore.
- Approving software only if it meets specifications of safety and performance, passes all tests required, and

does not hinder the life or safety of a person.
- Ensuring that any end-user documentation for the software is fully prepared. This was also a problem in the Therac-25 system, as AECL did not include definitions of their error messages inside of the operator's manual. Because of this, many users of the system just bypassed errors the system displayed.
- Disclosing any risks or hazards known about the system prior to the installation of the product.
- Cooperating with any concerns directed at the company regarding safety. This includes fully investigating any safety claims. In the case of AEGIS, the company quickly changed their systems over to the Linux operating system in order to fix rebooting problems. Therac-25 did, in some respects, cooperate with the customers but refused to take any extra actions until it was proved that their product was causing problems.

Besides the examples mentioned, Union Carbide possibly violated ethical standards by not adding safety functions to their software. Problems with the system, though probably known during development, did not implement any safety interlocks to stop chemical mixings without at least one of the safety implementations in place. AECL also violated ethical standards, even going so far as continuing sales of Therac-25 units after multiple accounts of product over dosages. Iran Air Flight 655 was shot down because the developers of AEGIS forced reboots after an active FOF signal was sent to a plane. Because the system was not forced to reboot, this could constitute as an ethical conflict.

Aftermath Management

When a safety-critical system does fail, it then becomes the responsibility of the company to follow up and take charge of the problem. There are many ways to deal with the problem; one of them being a process called closed loop corrective action (CLCA).

CLCA

CLCA is a process developed by the International Organization for Standardization (ISO) standards foundation in order to make sure companies respond correctly to reports of hazardous situations. The usage of the CLCA process may either be triggered by ISO 9000 noncompliance or by concerns from the company itself. Depending on the severity of the problem, the system may even require a complete rewrite of the working build [6]. CLCA, also known as corrective and preventive action, is something that is planned in order to prevent an action from happening in the future. Closed loop just means the development process is closed until problems with the system are corrected. It is described in [6] as:

> The pattern of activities which traces the symptoms of a problem to its cause, produces solutions for preventing the recurrence of the problem, implements the changes and monitors that the changes have been successful.

There are a couple of steps to using CLCA:
1. Identify the problem.
2. Find the root cause of the problem.
3. Define procedures to correct.
4. Implement the new changes.
5. Follow up on the changes.

Results

CLCA processes could have helped in both the Bhopal and Therac-25 incidents. Bhopal could have recalled their entire tank mixing solutions after the incident and created measures that prevented the system from running without safety measures in place.

Therac-25 could have used some sort of CLCA to handle some of the concerns by medical professionals after the system killed three people. AECL denied responsibility at first because they could not reproduce the problem. After the first scare, AECL did not do extensive testing on each claim, stating that the system was completely safe [2].

Conclusion

By using hazard identification and analysis techniques, many of the problems that plagued Union Carbide, the Patriot Missile System, AEGIS, and Therac-25 could have been avoided. These techniques allow for systems engineers and software developers to find and remove hazards and risks before they turn into liabilities. Along with these techniques, CLCA allows for companies to efficiently deal with system failures in an ethical way. Using these preset strategies and learning from past mistakes, companies can avoid lawsuits and make products safer for consumers.◆

References

1. Kopec, D., and S. Tamang. "Failures in Complex Systems: Case Studies, Causes, and Possible Remedies." ACM SIGCSE Bulletin, June 2007, Vol. 39, Issue 2 <http://portal.acm.org/citation.cfm?id=1272905>.
2. Levenson, N.G., and C.S. Turner. "An Investigation of the Therac-25 Accidents." Computer July 1993. ACM Digital Library, Los Alamitos, CA <http://portal.acm.org/citation.cfm?id=161477.161479&coll=guide&dl=>.
3. Thayer, R.H., and M.J. Christensen. Software Engineering, Vol. 1 – The Development Process. 2nd ed. Hoboken, NJ: John Wiley & Sons, 2005.
4. Chambers, L. A Hazard Analysis of Human Factors in Safety-Critical Systems Engineering. Proc. of the 10th Australian Workshop on Safety-Critical Systems and Software. Apr. 2006. Australian Computer Society Sydney. Australia, 2006.
5. Piner, M.G. "Computer Society and ACM Approve Software Engineering Code of Ethics." Computer Oct. 1999.
6. Dodd. B. "Closed Loop Corrective Action." Online Training Materials. Spring 2002 <www.freequality.org/sites/www_freequality_org/documents/Training/Classes%20Spring%202002/Closed%20Loop%20Corrective%20Action.ppt>.

About the Author

Corey P. Cunha is a software developer and project manager for Savard Engineering (SE), an engineering firm focused on design, development, project management, and system analysis, where he designs and codes various systems built on PHP, ASP.NET, and SQL Server for social Web sites. Through SE, he has also built and maintained systems involving Global Positioning System tracking and plotting capabilities. Cunha currently has an internship working with GE Healthcare working in unit test management, test documentation, and enforcing coding standards, and is enrolled in the software engineering program at Champlain College in Burlington, Vermont.

Savard Engineering
75 Ethan Allen DR
Burlington, VT 05401
E-mail: corey.cunha@mymail.champlain.edu