Devin Delfino

COMP 401: Senior Seminar

Project 02: Ethics in Safety-Critical Systems

As technology continues to become more advanced and central to society, it is a direct consequence that software is adopting a more prominent role in the operation of various systems, including those that may come with significant dangers and risks. These are known as safety-critical systems, which are computer-controlled systems whose failure can result in serious physical harm (or even loss of life) to human beings, serious damage to the environment and property, or failure to complete an important mission. Some examples of these types of systems include weapon systems, nuclear power plant control, fly-by-wire aircrafts, and heart pacemakers [3]. Using software to operate such a product, although it may be more efficient, decreases the overall safety of the system due to the difficulty in evaluating the functionality and reliability of complicated software [1]. It is clear that a failure in any one of these systems mentioned would produce catastrophic consequences, so there must be significant efforts to minimize the risk of error in the software that controls them. When it comes to designing and implementing software for safety-critical systems, there are various ethical decisions that must be made in order to ensure the safest possible product while still maintaining other desired qualities, such as cost-effectiveness, usability, and availability.

According to Bowen, as ethical questions arise in the development and implementation stages of a safety-critical system, it is the software development team's responsibility, specifically the software engineers, to make the ethical and rational decisions that will produce the safest product for the customer [1]. It is the software team's job to determine which technique or approach will be most effective while developing the system at hand. It is common for companies to have a certain standard regarding object-oriented approaches or specific programming languages to be used in their software, which should not be the reason a particular approach is chosen for a safety-critical system. The technique chosen for the system should reflect the trusted approach that has been proven to be effective in the past and comes with the minimum amount of risk [1].

The software engineers also thoroughly need to examine the requirement specifications of a system to ensure that all possible outcomes are accounted for. Most of the time failures occur during a safety-critical system due to incomplete requirements rather than coding errors, where the system encounters conditions that have not been tested or predicted [3]. The engineers need to work with the ones who designed the requirements of the system in order to handle all cases within the software, especially if the details of the system are beyond the level of expertise of the developer. In addition to analyzing the requirements of the system, the software engineers must also rigorously test these requirements, both functional and safety, which is done mostly using formal methods.

Formal methods are mathematically-based techniques that use logical reasoning to model, design, and analyze computer systems [2]. These methods help understand the functional behavior of the system, as well as guarantee that the system acts according to the requirements that have been chosen. For example, Fault Tree Analysis is a method used to evaluate the reliability and safety of the system by organizing possible causes (nodes) of a specific system failure (root) into a tree structure [2]. In this method, a path from a leaf of the tree to the root of the tree is determined by a series of logical operators, representing the transition from a possibly problematic event that occurs in the system to a subsequent event, eventually leading to the failure. This way, the engineers can use a cause-and-effect analysis to understand each possible failure to the best of their ability and act accordingly to prevent them from happening.

When it comes to formal methods, it is important for the engineering team to introduce them into the design process as early as possible, especially prior to the actual programming stages [1]. Usually, serious errors aren't found until some preciseness is introduced into the process, whether it is using a programming language to actually begin implementing the system or formal methods of using mathematics to accurately model the system. Since formal methods cost less than actual implementation, using these techniques while the engineering team is still designing the system using informal language and diagrams can help discover errors sooner in the process (and ultimately saving time and money) [1].

Overall, the safety and success of the system is heavily dependent on the ethical judgment calls of the software engineers designing and developing it, whose decisions are naturally more astute and consistent with more experience and skill. Therefore, guaranteeing that the technical skill, mathematical knowledge, intuition, and experience of the engineers implementing the safety-critical system are of the highest quality can pay great dividends in the attempt to control the risk of failures. Great technical skill and experience will allow for the engineer to analyze the problem and approach it using the safest, most appropriate technique (one that perhaps they have worked with before), as well as make the right calls when faced with a difficult decision [1]. A strong mathematical background will allow for the engineers to directly utilize and understand formal methods to validate the system requirements, which reduces the amount of information that must flow between two parties with differing background knowledge, which would occur if mathematical experts were needed to perform the tasks that software engineers were incapable of doing themselves. This flow of information across bodies can be very difficult when the system is extremely complex, making the loss of information a huge pitfall and cause of error that could be alleviated if the software engineers had a more balanced skill set [1].

In order to prepare software engineers to be more qualified to contribute to software for safety-critical systems, Bowen suggests adding more ethics and mathematics courses to undergraduate computer science curricula [1]. This way, software engineers will have been previously exposed to material that will allow them to transition more smoothly into roles that demand a consistency in ethical decision making and mathematical confidence. A lot of what makes a software engineer more qualified to work on safety-critical systems can only be learned from experience, so it is beneficial to expose future developers to the tool sets required to succeed in these demanding projects during their early education.

Bowen also mentions that there are currently no special certifications required of software engineers to develop and maintain software that controls safety-critical systems [1]. Many other (non-software) engineering roles require various types of documented qualifications in order to contribute to a specific job or project, depending on the associated risks and sensitivity. Rather than trusting that a software engineer will be honest and not attempt to work beyond his or her known limitations, perhaps introducing some sort of formal certification process will ensure that only the most qualified and experienced software engineers are able to work with safety-critical systems [3].

Although it may be impossible to completely eradicate the risk of failure from software that operates a complex system, a safety-critical system is considered 'safe' when it is highly unlikely that it will produce an output that will cause the system to fail [3]. The software engineers that implement this software are responsible for selecting the most effective techniques, evaluating the completeness of functional requirements, and verifying that the system successfully meets the safety requirements. There are many desired qualities that factor into the production of software, including cost, usability, availability, and time, but in a system that has the potential to inflict serious or even fatal harm to humans, property, or the environment, safety must be valued above all the rest.

"Program testing can be used to show the presence of bugs, but never to show their absence!"

- Edsger W. Dijkstra

References

[1] Bowen, J. The ethics of safety-critical systems. *Commun. ACM 43*, 4 (April 2000), 91–97.

[2] Liu, S., Stavridou, V., Dutertre, B., and Stavridou, S. L. V. The practice of formal methods in safety critical systems. *Journal of Systems and Software 28* (1995), 28–77.

[3] Muftic, A. Ethics and safety-critical software systems. Master's thesis, Mälardalen University.