The purpose of this panel is to present different perspectives and opinions regarding the issues surrounding why software should or shouldn’t be entrusted with critical (high consequence) functionality.

1 High Consequence Systems

Definition 1 high consequence system: A system in which significant loss\(^1\) results from a failure.

For numerous practical reasons, when designing a system, there is generally a limitation placed on the resources that can be expended in order to (1) increase the reliability of the system’s design, and (2) provide convincing evidence that the system’s design and implementation is reliable. It is typical for performance, inclusive of reliability, to be compromised due to cost and schedule constraints.

Because of these practical considerations, in a well designed system, the level of resources that one is willing to expend in order to provide sufficiently convincing assurance that a particular failure will not occur should be commensurate with the severity of the loss resulting from that failure. Thus the greater the consequence associated with the failure, the greater the level of assurance needed that the system will not experience such a failure. A difficult question that arises at this point is: “How does one measure level of assurance?”

Historically, a relatively unambiguous way to measure level of assurance has been in terms of the number of test cases to which the system has been subjected.

\(^{1}\)This work was supported by the United States Department of Energy under Contract DE-AC04-94AL85000. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy.

\(^{1}\)The term loss can refer to anything from “loss of money” to “loss of national security” to “loss of life.”
DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, make any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.
DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.
consequence application should be challenged.

2 High Consequence Software Systems

Because of their complexity, the ability to extensively analyze software systems is considered by many to lie beyond current technologies. This resistance to analysis increases dramatically when the analysis includes the environment in which the system is expected to operate safely. Due to the low failure rates demanded by high consequence software systems, high assurance must be provided that the system will not experience a failure with respect to a wide environmental spectrum (e.g., lightning strikes, power failures, fire, etc.). As the number of environments that must be considered in the analysis increases so too does the difficulty of the analysis.

In the limit, the combined complexity of a software system and its environment spectrum is such that it can be (and has been) argued that a software system should not be entrusted with high consequence functionality because it cannot be sufficiently analyzed.

However, it can also be argued that viewing a system solely from the perspective of its potential for failure in some sense provides only half of the picture. A better perspective of the overall value of a system can be obtained by also taking into account the positive benefits derived from proper functioning of the system. Informally stated then, the true value of a system should be determined by measuring the potential of a system to do “bad” and weighing it against its potential to do “good”. From this it follows that the decision to support a specific functionality in a system design should take into account the consequence associated with failures that can result from providing this functionality as well as the benefits derived from this functionality.

Along these lines, a primary reason for using software in a system design is to obtain a highly beneficial refined functionality that cannot be realistically obtained using other technologies. Oftentimes the choice a system designer is faced with is to either realize this functionality through software or to accept a system with reduced functionality. The difficult question to answer at this point is: “Does enhancing a system with a refined functionality increase the net value of the system?”

2.1 Example

Consider a situation where two armies are engaged in a war. The first army is equipped with missiles that have highly intelligent software-based guidance systems. Suppose that the guidance system is so complex that extensive analysis is only able to provide convincing evidence that a high consequence failure will occur with a probability of $10^{-4}$. Note that in reality, the actual system may have a lower probability of experiencing a high consequence failure. The evidence provided by the analysis only provides convincing evidence that the failure probability of the system is no worse than $10^{-4}$.

The missiles of the second army have a very simple, but crude, guidance system that has a high consequence failure probability of $10^{-7}$. Both armies have 1000 targets of critical value. An army looses the war if it is the first to have all of its 1000 targets destroyed. Which army would you join?

References

