Abstract
The paper discusses the ways and means of improving the reliability of safety software. Safety software should have high reliability and low probability of failure on demand. This paper proposes some of the techniques that will improve the reliability of safety software.

1.0 Introduction
In the modern era, hardware is anything you can hit with a hammer; software is what you can only curse at. This is not Murphy’s Law. Hardware design and development has been prevalent since the Stone Age while software design and development is only a few decades old. Its age speaks to its maturity.

Safety systems relate to human life, and the most unwanted component of the safety system is software. Software has been branded as the most unreliable component in any safety system because of its user random and unpredictable failures.

This paper proposes certain means and ways of improving software reliability for safety systems.

2.0 Description
A safety-critical system is a system where human safety is dependent upon the correct operation of the system. It can also be defined as failures in systems that can lead to the death of one to potentially several million people.
2.1 Software Function Error Seeding

Error seeding is an age old technique sparingly used in the system testing environment. Artificial errors are introduced into the system. Testing stops when a certain percentage of them are detected.

Detected seeded errors/total seeded errors = detected non-seeded errors eliminate spaces around slash / total non-seeded errors

\[ X = Y \left( \frac{A}{B} \right) \]

An estimate of total non-seeded errors can be obtained

Where

- \( X \) = Total non-seeded errors (Undetected error)
- \( Y \) = Detected non-seeded errors
- \( A \) = Total seeded errors
- \( B \) = Detected seeded errors

This technique gives a fair idea of about the residual errors remaining in the system level and can also be used as criteria to stop the system testing. The concept of error seeding at the system level can also be used in the software unit/function level testing with some modifications.

The behavior of a software function is based on the cyclomatic complexity (CF) of the function and the average cyclomatic complexity of the file (AvCf) in which the function resides. The probability of errors in a function increases exponentially as the cyclomatic complexity of the function increases, as defined by the following graph.

The average cyclomatic complexity is also an important parameter, as the software function is not independent, and it interacts with other functions within the file. The cyclomatic complexity of other functions also has an “error” effect in the function we are analyzing.

The assumptions for this kind of approach are:

- Software is modular.
- Use of global variables is restricted.
- Software function interfaces within a file are well documented.

Based on the assumptions and the parameters defined, we now define a parameter known as NK’s parameter.

NK’s parameter is defined by the ratio of the cyclomatic complexity of the function analyzed to the average cyclomatic complexity of the file (*.c, *.cpp, *.ada, etc.) where the analyzed function resides.

\[ NK = \frac{CF}{AvCf} \]

Another interesting parameter that determines the effectiveness of the error seeding for a software function is the number of errors that need to be injected in the function. The number of errors that need to be injected into the function depends on CF, AvCf, input parameters and the number of return states from that function.

\[ \text{Errors to be injected} = (NK)^*(\text{input parameters for the function} + \text{number of return states from that function}) \]

Using the above defined constants and assumptions, the residual errors in the software function using the error seeding for software technique is

\[ X = (NK) \cdot \frac{Y}{A/B} \] : substituting the values from the above expression

\[ X = ((NK)^2) \cdot \frac{Y}{B} \cdot (\text{input parameters for the function} + \text{number of return states from that function}) \]

\[ X = ((NK)^2) \cdot \frac{Y}{B} \cdot \text{Interface Complexity} \]

This will give a fair idea of what function to concentrate more on to discover the hidden errors.

Advantages

- Improves software reliability.
- Improves the error detecting capability.
- Residual testing estimates to discover the hidden problems becomes easy.

Disadvantages
Expensive and time consuming.

Source code gets instrumented.

Path Forward
You can arrive at a more accurate expression for NK’s parameter by considering an exponential factor for the relationship between the cyclomatic complexity and the probability of errors.

2.2 Traceability Analysis

Traceability analysis is a technique that provides a path to the validation and verification of stakeholder requirements to ensure that their needs are met by the systems delivered. Vertical slice design analysis and horizontal slice design analysis can be performed with the traceability information.

2.2.1 Vertical & Horizontal Slice Design Analysis

The vertical slice design analysis takes care of the sequential flow of the development process in the “V” model. The VSA technique is a method to assess the development strategy and quality of software. It can also be applied to system, product and hardware development. It is important that the VSA is carried out at a number of stages throughout the development lifecycle of a product and that the results are fed back into the design and development structure. This being the case, it can be seen that development techniques are consistent throughout the design lifecycle, and that items of concern can be countered.

The scope of the analysis is to carry out a detailed analysis of a selected part of the design to ascertain whether this has been developed and documented to the required integrity level of the product. The analysis should be based on a snapshot of the design assessing small vertical sections, examining all aspects of one particular requirement.

The analysis is not intended to cover the complete design; however, the level of detail of analysis performed is dependent upon the degree of complexity of the software, and the degree of confidence gained as a result of initial assessment.

A horizontal slice design analysis can also be performed to ensure that the design through testing is adequate enough to satisfy the integrity level of the software. This kind of analysis will increase confidence that the testing is adequate to satisfy the safety requirements.

Traceability analysis (vertical and horizontal) provides us with the following benefits:

- Identification of most important or riskiest paths
- Locations of interactions
- Completeness / omission
- Identification of re-test areas
- Impact of change
- Discovery of root cause of faults, failures

2.3 Software Fault Tree Analysis

Fault tree analysis (FTA) is a top-down, deductive analytical method. In FTA, initiating primary events such as component failures, human errors, and external events are traced through Boolean logic gates to an undesired top event such as an aircraft crash or nuclear reactor core melt. The intent is to identify ways to make top events less probable, and verify that safety goals have been achieved. The purpose of performing a software fault tree analysis (FTA) is to demonstrate the functional safety of software by tracing the cause of system hazards into the logic of the code and determine whether a path exists through the code that could cause a hazardous output.

FTA may be qualitative or quantitative. When failure and event probabilities are unknown, qualitative fault trees may be analyzed for minimal cut sets. Quantitative FTA is used to compute top event probability, and usually requires sophisticated tools. A software fault tree analysis determines what the most likely failure events are pertaining to software. Then it determines what single or multiple point failures could produce these top level events. FTA’s are useful when designing or testing the error handling part of software.

Software FTA is intended to be an extension to traditional system-level FTA, and hence software and system fault trees can be integrated together with little difficulty. Software FTA can be initiated at an early stage in the software development lifecycle, i.e., once the software architecture has been defined together with initial module structure and sequence diagrams.

The basic software failure events are:

- False alarms
- Insufficient error handling
Sequencing or order
Timing
Outputs are incorrect
Outputs are valid but not expected

Software fault trees are most effective when performed during each phase of the lifecycle starting with system requirements.

2.4 Software FMEA

Failure mode and effects analysis (FMEA) is a bottom-up, inductive analytical method that may be performed at either the functional or piece-part level. The software FMEA should be used as an analysis tool to assess the consequences of single point failures within the software, to identify hazards that have not previously been identified and to determine the levels of mitigation present within the product and system.

The software FMEA can be considered as a process FMEA as well as a product FMEA.

Process FMEA: The FMEA performed on software can improve the development process as identified by the failure modes. The development process could be an update to coding standards, implementing static and dynamic analysis as a part of the development process, and improving the development process and procedures.

Product FMEA: The identified failure modes could also result in implementing more diagnostics and defensive programming techniques to improve the error handling. The product FMEA will result in a safe shutdown of the system for any random unsafe faults.

Software FTA when used along with software FMEA early in the lifecycle will be much more effective.

2.5 Development Tool certification and Validation

Assessment and evaluation of COTS software products has become a compulsory and crucial part of any COTS-based software system lifecycle. The risk of selecting a product with unknown quality properties is no longer acceptable. Assessment of software COTS components is an essential part of component-based software development. Poorly chosen components may lead to solutions of low quality and that are difficult to maintain.

Safety-critical systems often make use of operating systems. Safety-critical software continues to make its way into military embedded application environments beyond avionics flight systems such as weapons systems and communications equipment. Often, all of these systems operate together as a single system. This makes it increasingly important that each of the components (including COTS) meets the most stringent and rigorous requirements for safety-criticality, since, if one system fails, there could be failures or vulnerabilities in the entire system.

COTS software can include:
- Any component re-used from previous releases.
- Open source tools and methodologies
- Software components that form part of a program.
- Standalone programs and utilities.
- High-level services that interact with multiple programs.
- COTS assessment should be performed to ensure:
  - It implements its stated functionality
  - No hazardous side-effects are added by the COTS functionality.

The main types of assessment that can be carried out on COTS software are:
- Development process assessment
- Proven in use
- Black & white box assessment
- Assessment based on any safety standards (e.g., IEC 61508)

In the present day, almost all the safety-critical tool vendors have their development process as defined by a safety standard (e.g., IEC 61508) and get the tool also certified as per the safety standard. Using a certified tool will improve the reliability of the software being developed.

Any tool that has been developed in-house has to be verified and validated before being used in any development activity. This will ensure that reliability of the development process is increased.
3.0  Conclusion

The primary key for successful development of any safety system depends on the development and organizational process as defined by the safety standards. Improving the reliability of a software component in the safety system contributes to reducing the risks further and improves the fault avoidance and control capabilities of the system. Some of the techniques presented in this paper help improve the software reliability.

The techniques defined will reduce the probability of failure of demand further down the road. They can be embedded as a part of the development process for any safety systems development.

4.0  References

- IEC 61508
- CENELEC EN 50128

About Cognizant

Cognizant (NASDAQ: CTSH) is a leading provider of information technology, consulting, and business process outsourcing services, dedicated to helping the world’s leading companies build stronger businesses. Headquartered in Teaneck, New Jersey (U.S.), Cognizant combines a passion for client satisfaction, technology innovation, deep industry and business process expertise, and a global, collaborative workforce that embodies the future of work. With over 50 delivery centers worldwide and over 118,000 employees as of June 30, 2011, Cognizant is a member of the NASDAQ-100, the S&P 500, the Forbes Global 2000, and the Fortune 500 and is ranked among the top-performing and fastest growing companies in the world.

Visit us online at www.cognizant.com for more information.