International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395 -0056
Volume: 02 Issue: 09 | Dec-2015 www.irjet.net p-ISSN: 2395-0072

Analyzation of Safety Verification at different stages of Safety Critical System Development

K. Amarendra
Research Scholar,
Department of CSE,
GITAM University,
Visakhapatnam

Prof. Dr. P. Seetharamaiah
Professor Emeritus,
Department of CS&SE,
Andhra University,
Visakhapatnam

Prof. Dr. J.A. Chandulal
Professor, Department of CSE,
GITAM Institute of Technology,
GITAM University,
Visakhapatnam

***Abstract*— In many important applications of computers today the most difficult problem that must be faced is how to obtain sufficiently safe operation. Due to the growing increase in computer-related technologies, industry is continuing to put greater demands on software-controlled systems. These demands sometimes place software in total or partial control of critical system functions such as navigating planes, determining radiation dosages, shutting down nuclear reactors, and identifying military targets. A fault in such a critical system can result in catastrophic consequences such as death, injury, or environmental harm. In order to detect and prevent such faults, researchers have developed safety standards, safety analysis techniques, and fault-tolerant techniques.**

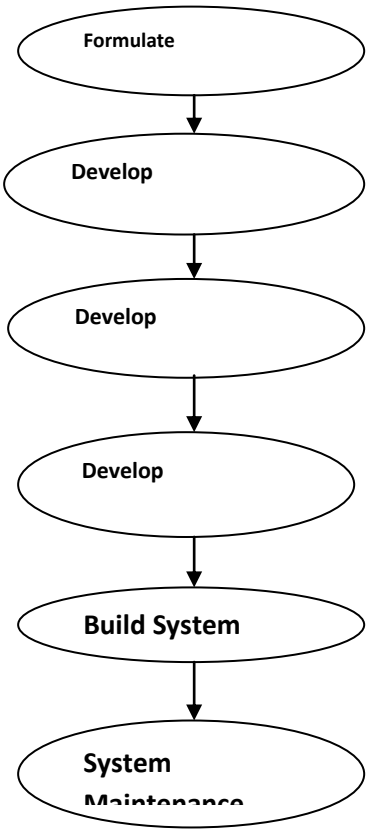
***Keywords*— Safety critical system, Hazard, Fault- Tolerant, Safety Standards, Static and Dynamic Verification.**

Introduction

Ensuring the correctness of computer systems is a complex task of paramount importance, especially when such systems control and monitor life-critical operations. The verification of industrial computer systems is particularly difficult due to their size and complexity. The most frequently used methods, simulation and testing, are not exhaustive and can miss important errors. While the use of both methods can increase the reliability of the application, they cannot fulfill the verification needs of modern complex safety-critical systems. Formal methods are an additional methodology to tackle this problem. Formal verification tools allow an exhaustive search to be automatically performed on the state space of the system, avoiding the shortcomings of both simulation and testing.

The increase in software-controlled systems is due to many factors such as cost, flexibility, and reliability. Research in software safety falls into two categories: (1) improving software safety before releasing the product by using verification techniques and (2) improving software safety after releasing the product by using fault-tolerant techniques. For verification techniques, most researchers concentrate on static methods, which analyze a software system's safety without executing it, and ignore dynamic methods, which analyze a system's safety by executing it. This research concentrated on developing a methodology framework that combines static-verification, dynamic-verification, and fault-tolerant concepts for verifying safety-critical software systems.

This research's methodology combines software-safety methods into a comprehensive whole for the purpose of verifying safety-critical software systems. For clarification purposes, this document treats the words approach, technique, and method as having synonymous definitions. A methodology brings structure, guidance, and specific techniques all together in order to improve a given process - in this case, the process is software-safety verification. This research deals with Safety Verification and validation Methodology(SVVM). Below diagram represents Standard phases for System development showing general exit and entry conditions.



Deficiencies within software safety: As normal for relatively new fields, software-safety methods and practices have deficiencies in many areas. Within software engineering, researchers have been looking into safety-related issues for approximately the past decade. Their research focused mainly on techniques for statically verifying and modeling safety-critical software systems and providing standards for developing such systems. However, safety standards are often too vague and have few and scattered guidelines.

II.SAFETY ISSUES

Safety Vs Reliability: Reliability deals with making sure a software system has no faults. Howden refers to a fault as an error in the software, and a failure as the erroneous behavior resulting from a fault. Reliability concentrates on removing all faults from the software without respect to the fault's severity or consequences; therefore, any fault degrades reliability to some degree.

Safety, on the other hand, deals only with faults that can cause accidents. The United States' Department of Defense (DoD) defines the term accident as "an unplanned

event or series of events that results in death or major injury to personnel or damage to the launch vehicle, experiments, spacecraft, associated support equipment, or facilities."The DoD defines a major injury as "any injury which results in admission to a hospital such as bone fracture, second or third degree bums, severe lacerations, internal injury, severe radiation exposure, chemical or physical agent toxic exposure, or unconsciousness.

Risk Analysis : The Electronic Industries Association (EIA) defines a hazard as an inherent characteristic of a thing or situation that has the potential of causing an unexpected,

unplanned, or undesired event or series of events that has harmful consequences such as injury, death, environmental harm, or illness Risk then is a function of a hazard's seventy and the probability that the hazardous event will occur.

Hazard Analysis :

A hazard is a situation that poses a level of threat to life , health, property or environment. Most hazards are dormant or potential , with only a theoretical risk of harm. A hazardous situation that has come to pass is called an incident . Hazard analysis includes hazard identification, Categorization, Resolution, Documentation and review.

III. SAFETY VERIFICATION STANDARDS

In dealing with system safety , we have to concentrate both on software safety and hardware safety. In this paper we mainly discuss about relevant standards and techniques dealing with software-safety research. Specifically, this research

discusses fault tolerance, static verification, dynamic verification.

Research in software safety falls into two categories:(1) improving software safety before releasing the product by using verification techniques and (2) improving software safety after releasing the product by using fault-tolerant techniques. For verification techniques, most researchers concentrate on static methods, which analyze a software system's safety without executing it, and ignore dynamic methods, which analyze a system's safety by executing it. This research concentrated on developing a methodology framework that combines static-verification, dynamic-verification, and fault-tolerant concepts for verifying safety-critical software systems.

Static Verification:

For safety-critical software systems, static processes should be in place to insure that the output from each phase satisfies the previous phase's safety requirements. Different techniques are described as follows.

Failure-Modes-and-Effects Analysis (FMEA) : This static technique concentrates on analyzing a component's potential failures, each failure's effects on the system, and

the immediate causes for each failure.

Fault-Tree Analysis (FTA): Engineers can use this static technique to analyze the causes for any specific system condition. For safety-critical systems, engineers use FTA to analyze the causes for hazards.

*Event-Tree Analysis (ETA):*This technique traces an event using forward analysis in order to determine its consequences on the system . ETA is similar to FMEA since both use forward analysis in order to determine an event's effects on the system.

Petri nets : A Petri Net is a mathematically-based static model for representing and analyzing system flow and control.

Safety Standards:

Governments and industry have developed several standards that address various aspects of safety-critical systems.

*MIL-STD-882B :*This requirements standard describes what the Department of Defense (DoD) expects from the government and contractors who are building safety-critical systems. The standard gives a general introduction to safety issues such definitions of important terms, the importance of managerial participation and support for safety, and an outline for a system-safety developmental team.

MTL-STD-1574A : As the forward for this standard mentions, it is a tailored application of MDL-STD-882B. The standard contains the normal definitions for safety-related terms, guidelines for a system-safety organization, guidelines on contractors and subcontractors, and functional guidelines for various system-safety groups.

*DOD-STD-2167A :*This standard is meant to help establish requirements for defense-related software system. The standard outlines what developers must do for the following areas: (1) software management, (2) software engineering, (3) formal qualification testing, (4) software-product evaluations, (5) software configuration management,(6) transitioning to software support.

DO-178A: Various working groups from organizations around the world, who are members of the radio technical commission for aeronautics (RTCA), helped to develop this standard to recommend methods and techniques for the orderly development and management of airborne software system.

SEB6 –A: This standard which the Electronic Industries Association (EIA) created, provides basic information and guidelines for carrying out the activities and tasks in MIL-STD-882B, DOD-STD-2167A, and MIL-STD-1574AJ The standard contains practical information for all life-cycle phases from requirements.

Dynamic verification:

Dynamic verification checks the software's internal consistency (during run time) against its specification and design; therefore, making sure that critical software components satisfy specific entry and exit criteria. Dynamic-verification methods that apply to safety-critical software are normally in addition to standard testing practices such as statement coverage, branch coverage, etc.

Self-checks: This technique is a classification of dynamic fault-detection categories, which various fault-tolerant techniques use. For example, N-version programming uses a replication self-check while a recovery-block approach uses replication and either a reversal or reasonableness self-check.

*Statistical Testing: S*tatistical testing is to calculate the software's failure probability. Statistical testing, in order to be effective, requires an accurate input distribution function and a method for randomly generating and automatically verifying test cases.

IV. DEVELOPMENT

There are many software fault removal techniques in literature. The most frequent classification is by differentiating between static and dynamic techniques [8]. Different authors focus on probabilistic based approaches (like the Markov modeling method), or statistical, approaches like statistical testing, software reliability models [9]. However most of the fault removal techniques are non-probabilistic. In some standards, static techniques require formal methods and proofs based on mathematical demonstrations. Other standards and literature classify these techniques in functional and logical terms [10] or by just mentioning functional testing like in [11] or structural testing, like in [12].

None of the fault removal techniques like algorithm analysis, control flow analysis, Petri-Net analysis, reliability block diagrams, sneak circuit analysis, event tree analysis, FMEA and FTA can be considered apt and complete in all respects, when used in isolation. A way out of this is to analyze how to combine individual techniques so that the fault removal process is significantly improved. One of the most effective combinations is FMEA+FTA. The literature [9,10] already mentions that FTA technique can be associated effectively with other practices like FMEA. Their greatest advantage is in combination with each other. FMEA concentrates in identifying the severity and criticality of failures and FTA in identifying the causes of faults. FMEA technique is a fully bottom-up approach and FTA has a fully complementary top-down approach. Moreover, these two techniques are directly compatible with system level techniques.

In this paper, we propose an implement an integrated approach to software safety analysis for critical systems that combines two existing fault removal techniques – FMEA and FTA to identify and eventually remove software faults at successive software development phases. We have applied our integrated safety approach to a model railroad crossing control system to validate its effectiveness. We also compare how the safety-specific software development of a critical system is distinct from the traditional non-safety-specific software development.

Railroad Crossing Control System (RCCS)

Crossing gates on a full-size railroads are controlled by a complex control system that causes the gates to be lowered to prevent access to the crossing shortly before a train arrives and to be raised to allow access to resume after the train has departed. This requires the detection of approaching trains or the manual actuation of the crossing gates by an operator. RCCS is a prototype, real-time, safety-critical railroad crossing control system of limited complexity. It is composed of several software-controlled hardware components.

RCCS System Functions:

• Control the overall operation of train on the track circuit.

• Control the opening and closing of Gate 1 and 2 at the railroad intersections

• Control the track lever to change the track route from the outer to the inner loop

• Check the internal health of all the subsystems

• Control the train operation at the Signal Lights

• Monitor all the sensors on the track circuit

RCCS System Operations:

When RCCS is first switched on, the controller does a preliminary check of the normal working status of all the subsystems involved – the driver circuitry, the sensors, the gate assemblies, and the train signals. If all the components are found to be in normal working condition, it executes the code related to normal operation. Initially, the train starts from the platform location and is programmed to run on the outer track. After it completes this cycle, it changes direction and runs on the inner track. This change is facilitated by the track-change level which is present at the intersection of the outer track and inner track. Along the track, the two gates Gate 1 and Gate 2 are automatically lowered when the train nears the railroad intersection and raised when the train leaves the intersection. Whenever the signal lights display Red, the train comes to a halt and resumes running only after a Green signal is given. Whenever the train detects any physical obstacle on the track, the train comes to a halt.

Figure 5 shows the block diagram of RCCS. If the train passes Sensor2 positioned prior to gate, a signal is sent to the controller indicating the approaching train. The controller then sends a signal to the gates assembly, causing the gate arms on either side of the road to close. When the train finally has passed Sensor3, which is positioned just beyond the gate crossing section, a corresponding signal is sent to the controller, which in turn triggers both the gate arms to open simultaneously.

Figure 5. RCCS block Diagram

V. SAFETY ANALYSIS AND RESULTS

The safety analysis of RCCS software functions takes place in three sequential steps.

• *Software Failure Mode and Effects Analysis (SFMEA):*

This analysis is performed in order to determine the top events for lower level analysis. SFMEA analysis will be performed following the list of failure types. SFMEA will be used to identify critical functions based on the applicable software specification. The severity consequences of a failure, as well as the observability requirements and the effects of the failure will be used to define the criticality level of the function and thus whether this function will be considered in further deeper criticality analysis. The formulation of recommendations of fault related techniques that may help reduce failure criticality is included as part of this analysis step.

• *Software Fault Tree Analysis (SFTA)*

After determining the top-level failure events, a complete Software Fault Tree Analysis shall be performed to analyse the faults that can cause those failures. This is a top down technique that determines the origin of the critical failure. The top-down technique is applied following the information provided at the design level, descending to the code modules . SFTA will be used to confirm the criticality of the functions (as output from SFMEA) when analyzing the design and code (from the software requirements phase, through the design and implementation phases) and to help:

- Reduce the criticality level of the functions due to software design and / or coding fault-related techniques used (or recommended to be used)

- Detail the test-case definition for the set of validation test cases to be executed.

• *Evaluation of Results :*The evaluation of the results will be performed after the above two steps in order to highlight the potential discrepancies and prepare the recommended corrective measures. Recommendation can be given to design and coding rules.

SFMEA Analysis of RCCS

The SFMEA, a sample of which is shown in the Table 2 below presents some software failure modes defined for RCCS. The origin and effects of each failure mode are analyzed identifying the top level events for further refinement, when the consequence of this failure could be catastrophic for this system. Three top events were singled out for further analysis of failure mode Gate not closed as train is passing through railroad intersection.

Failure Mode	Possible Causes	Effect	Sever-ity of risk	Prevention And Compensation
Gate not closed as train is passing through	a) sensor not detected by s/w b) gate motor mechanism is defective c) s/w gives wrong command d) s/w gives right command at wrong time	Train collision with passing road traffic leading to accidents	Critical	Software first checks the working status of gates each time the train is about to cross the gates
Track change lever is not acti-vated to change train route	a) sensor is not detected by s/w b) track lever motor mechanism is defective c) s/w gives wrong command to lever d) s/w gives right command at wrong time e) s/w fails to give a command to acti-vate lever	Train fails to change its path from the outer track circuit to the inner track circuit leading to accident	Critical	Software first checks the working status of the track lever each time the train is about to enter the inner track loop
Control program software is corru- pted	a) logic fault b) interface fault c) data fault d) calculation fault e) memory fault	Unpredictable sequence of opera-tion leading to accident	Critical or Catast-rophic	Algorithm logic is verified for accuracy. Data Structures and Memory overflow is checked.

SFTA Analysis of RCCS

The fault tree is a graphical representation of the conditions or other factors causing or contributing to the occurrence of the so-called top event, which normally is identified as an undesirable event. A systematic construction of the fault tree consists in defining the immediate cause of the top event. These immediate cause events are the immediate cause or immediate mechanism for the top event to occur. From here, the immediate events should be considered as sub-top events and the same process should be applied to them. All applicable fault types should be considered for applicability as the cause of a higher level fault. This process proceeds down the tree until the limit of resolution of tree is reached, thereby reaching the basic events, which are the terminal nodes of the tree.

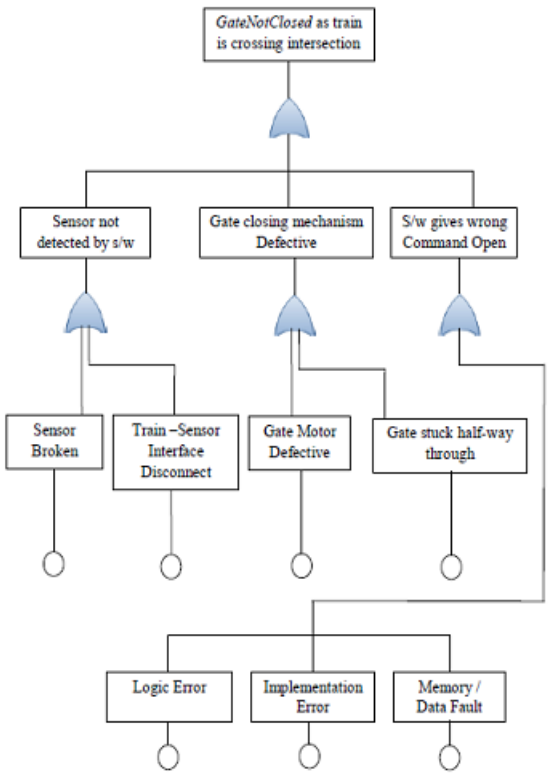


Figure 9.16: Software Fault Tree Sample for the Failure Mode

A new technique was developed based on a combination of two existing techniques: the failure modes and effects analysis (SFMEA) and fault tree analysis (SFTA). These two techniques complement each other very well: SFMEA is a bottom-up approach that concentrates on identifying the severity and criticality of the failures and SFTA as a fully complementary top-down approach that identifies the causes of the faults. It is possible to integrate both techniques with commonly used techniques at system level. The resulting new technique can be shown to combine nearly all aspects of existing fault removal techniques. This should enable, at least theoretically, coverage of a large number all software failure modes and fault types that occur in real time critical software applications.

VI. CONCLUSION

This paper gives brief description about static and dynamic verification methods for safety critical systems. It also studies about fault tolerant techniques. The methodology follows a life-cycle approach to verification by supplying methods and guidelines for preliminary hazard analysis, high-level-design hazard analysis, detailed-design hazard analysis, and code-level hazard analysis. The methodology contains several testing and coverage techniques along with guidelines for dynamic verification, which is an area that research largely ignores in spite of its importance.

References

[1] MIl-STD-1574A (USAF), "System Safety Program for Space and Missile Systems," Dept of Defense, US Govt. Printing Office, 1979

[2] P. V. Bhansali, "Software Safety: Current Status and Future Directions" *ACM SIGSOFT Software Engineering Notes,* Volume 30 Number 1, page 1, January 2005

[3] John C. Knight, "Safety Critical Systems: Challenges and Directions", *Proceedings of the 24th International Conference on Software Engineering (ICSE),* Orlando, Florida, 2002

[4] MIL-STD-882C, System Safety Program Requirements 1993, http://eic.ipo.noaa.gov/IPOarchive/MAN /doc124.pdf

[5] N.G. Leveson, Safeware: *System Safety and Computers,* Addison-Wesley, Reading, MA, USA, 1995.

[6] W.R. Dunn, Practical Design of Safety-Critical *Computer Systems*, Reliability Press, 2002.

[7] M.S. Jaffe and N.G.Leveson, "Completeness, robustness, and safety in real-time software requirements specification", *Proc. of the*

11th International Conference on Software engineering (ICSE), Pittsburgh, USA, pp. 302-311, 1989

[8] Raghu Singh. "A Systematic Approach to Software Safety". *Proceedings of Sixth Asia Pacific Software Engineering Conference* (APSEC), Takamatsu, Japan ,1999.

[9] T. Shimeall and N. Leveson, *An Empirical Comparison of Software Fault Tolerance and Fault Elimination*, IEEE Transactions On Software Engineering, vol. SE-17, no. 2, pp. 173-183, 1991.

[10] P. Rodríguez Dapena. 'How are static fault removal techniques verifying software safety and reliability?' Joint ESA-NASA Space-FlightSafety Conference. ESA. 06-Nov-2001

[11] *Software Safety*. NASA-STD-8719.13A NASA Technical Standard. September 15, 1997 Replaces NSS 1740.13 dated February 1996. http://swg.jpl.nasa.gov/resources/index.shtml

[12] L. M. Ippolito, D. R. *Wallace A Study on Hazard Analysis in High Integrity Software Standards and Guidelines*. National Institute of Standards and Technology. NIST IR 5589. January 1995