FMEA and Fault Tree based Software Safety Analysis of a Railroad Crossing Critical System

Ben Swarup Medikonda[1], P. Seetha Ramaiah[2] and Anu A. Gokhale[3]

[1] Department of Computer Science and Systems Engineering Department of Technology Andhra University

Received: 29 March 2011 Accepted: 24 April 2011 Published: 6 May 2011

Abstract

Software for safety-critical systems must deal with the hazards identified by safety analysis in order to make the system safe, risk-free and fail-safe. Certain faults in critical systems can result in catastrophic consequences such as death, injury or environmental harm. The focus of this paper is an approach to software safety analysis based on a combination of two existing fault removal techniques. A comprehensive software safety analysis involving a combination of Failure Modes and Effects Analysis (FMEA) and Fault Tree Analysis (FTA) is conducted on the software functions of the critical system to identify potentially hazardous software faults. A prototype safety-critical system - Railroad Crossing Control System (RCCS), incorporating a microcontroller and software to operate the train on a track circuit is described.

Index terms— Software safety, safety-critical systems, software faults, software safety analysis.

Introduction safety-critical system is one that has the potential to cause accidents. Software is hazardous if it can cause a hazard i.e. cause other components to become hazardous or if it is used to control a hazard. Software is deemed safe if it is impossible or at least highly unlikely that the software could ever produce an output that would cause a catastrophic event for the system that the software controls. Examples of catastrophic events include loss of physical property, physical harm, and loss-of-life. Software engineering of a safety-critical system requires a clear understanding of the software's role in, and interactions with, the system [1,2]. a) Software-induced failures in real-life Computers are increasingly being introduced into safety-critical systems and, as a consequence, have been involved in accidents. Some well known incidents are the massive overdoses given by the computer-controlled radiation therapy machine Therac-25 [3] with resultant death and serious injuries, during the mid-eighties; European Space Agency's Ariane 5 rocket explosion [4] during lift-off in June 1996, and SeaLaunch rocket failure [5] during lift off in March 2000. Recent examples include the following: on 7 October 2008, Qantas Flight 72 from Singapore to Perth made an emergency landing following an inflight accident featuring a pair of sudden uncommanded pitch-down manoeuvres that resulted in serious injuries to many of the occupants. The Australian Transport Safety Bureau (ATSB) said that incorrect information from the faulty computer triggered a series of alarms and then prompted the Airbus A330's flight control computers to put the jet into a 197-metre nosedive **??**6].

All these examples indicate that accidents still take place despite all the measures taken to prevent them. Since complete elimination of unforeseen hazards is not always possible, what we need is a fail-safe design which, in the event of a failure, allows the system to fail in a safe way, causing no harm or at least the minimum level of danger. To meet the fail-safe requirements, rigorous safety analysis is required to identify potential hazards and take corrective measures during the entire system development life cycle.

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 analyse 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 A 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 a system-level 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 safety approach to a model railroad crossing control system to validate its effectiveness.

We also compare how the safetyspecific software development of a critical system is distinct from the traditional non-safety-specific software development.

The rest of this paper is organized as follows: section 2 describes the Railroad Crossing Control System (RCCS). Section 3 applies the safety analysis using SFMEA and SFTA techniques to RCCS. Section 4 addresses the hardware and software development issues of RCCS. Section 5 presents an analysis of the experimental results and section 6 concludes the discussion.

1 II.

2 Railroad Crossing Control System (RCCS)

Crossing gates on a full-size railroad 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. RCCS is a prototype, real-time, safety-critical railroad crossing control system composed of several software-controlled hardware components.

3 a) RCCS Interfaces

The main interfaces of the microcontroller, which hosts and runs the embedded software, are shown below in Figure **??**. The main inputs to the microcontroller are signals from the 7 sensors on the track, the 2 gates at the railroad intersection, the trackchange lever, and the 3 signal lights. The main outputs of the micro-controller are control signals for the train, Gate1 Gate 2, track change lever, signal lights, LCD display. The values of these output signals are determined using different algorithms combining the input signals that are constantly updated and read by the software.

4 Figure 1. External interfaces of RCCS microcontroller

The main functionality of RCCS is listed in Table **??**.

5 Table 1. RCCS System Functions -Key Areas

6 Safety Analysis of RCCS

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.

7 ? 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.

8 a) 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.

9 b) 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 **??** shows the sample fault tree for the top event Gate Not Closed at the railroad intersection.

10 c) Recommendations to Design and Coding

From the safety analysis we have conducted, the major critical events that might occur and the corresponding safety properties the RCCS software has to implement, and which are controlled by the embedded software in the microcontroller are listed below.

11 Figure 2. Software Fault Tree sample for top event

Gate Not Closed at the railroad intersection

? The software shall make sure that the 2 gates on either side of the railroad intersection operate correctly -ie. opening and closing the gates, at the proper time. The consequences of failure to do so are very severe, since it can result in the train and road traffic collision, leading to death. ? The software shall make sure that the train changes its path from the outer track circuit to the inner track circuit by correctly operating the track change lever at the right time. Failure to do so can have severe consequences leading to collision with another train that may be stationary on the outer track. ? The software shall prevent the running operation of the train if it detects that the gates at the intersection have not been fully closed. ? The software shall prevent the running operation of the train, if the train engine detects any physical obstacle just ahead of it, either at the mid-section of the railroad intersection or at any point on the track path, just ahead of the engine. Failure to do so can lead to collisions.

? The software shall the running operation of the train if a Red signal is displayed in the Signal Light alongside the track. Failure to do so can lead to accidents. ? The software shall prevent the running operation of the train if the train engine is not able to confirm that a green signal has been given to it, to resume running after a previous red signal to stop running. ? The software shall bring the running train to a halt at the location designated as railway station platform, on the track, after every cycle of operation around the track. Failure to do so can cause collision with another train that is passing just ahead on the same track. IV.

12 RCCS Development

RCCS hardware and software development is described in this section. Train: The train is powered by a power supply relay.

When the power is initially switched on, the train begins movement along the track when the metallic wheels of the train receive power. The train comes to a halt at the position where the power to the tracks is switched off.

Sensors: These are used to detect the location of the train on the tracks. Altogether RCCS employs seven sensors. Two pairs of sensors detect the train position before and after the gates. A set of two sensors relate to track change where the track splits into two directions. One sensor gives the train position with reference to the platform, which is the starting point of the train movement. Information from each of the sensors is passed to controller. The safety-specific version of RCCS controller program used the same techniques as the non-safety version with the addition of the following safety-specific analysis: preliminary hazard analysis, and design-level hazard analysis, FMEA and FTA analyses. These techniques target the specification and designs. The goal here is to determine if the inclusion of these methods reduces the number of latent safety-critical faults relative to non-safety specific methods.

The software safety-based development involves preliminary software hazard analysis, which among other things identifies software hazards, ie. the states in the software that can lead to an accident. Without identifying the hazards, we have little assurance that the hazards will not occur. Therefore, preliminary software hazard analysis is an important first step in verifying safety-critical software systems. Once the hazard list exists, the verification process can continue by applying several static and dynamic verification techniques. Static techniques include failure modes and effects analysis (FMEA), and fault-tree analysis (FTA).

After static verification, software engineers must dynamically verify the software's safety (ie. safety testing). Safety-critical testing of RCCS can be done by separating the code into two risk groups. Group one includes hazards that are catastrophic or critical. Group two includes hazards that are marginal or negligible. More testing effort should be spent on those code sections dealing with hazards related to group one.

V.

13 Experimental Results & Analysis

In view of the comprehensive safety analysis, and specification and implementation the safety properties during RCCS design and development, the expected result was that safety-specific RCCS development would produce a software system with fewer latent safety-critical faults than traditional nonsafety specific techniques alone. This is due to the belief that the safety-specific techniques will prevent safetycritical faults in the specifications and designs that the traditional techniques have a tendency to miss. Figure 4 shows the RCCS laboratory prototype developed in the lab.

During the operation of RCCS, the safetyspecific development version of RCCS clearly demonstrated the fulfillment of the safety properties. For example, if the gate at the railroad intersection is not closed at all, or partially closed, as the train is about to pass through the intersection, the controller software makes the train come to a halt. Only after confirming that the gate is fully closed does the software allow the train to pass through the railroad intersection. On the other hand, in the non-safety version of RCCS, the controller software allows the train to pass through the intersection without confirming whether the gate is actually closed or not, assuming that the gate function will operate without failure, leading to a major accident.

Likewise, in the safety-version of RCCS, when the train is changing its track route from the outer loop to the inner loop, the software first confirms whether the track change lever is fully activated and operational. If the track lever is stuck halfway through and the rails connection to the inner loop is incomplete, the software makes the train come to a halt. In the case of the nonsafety version, the software allows the train to change route without confirming the health status of the track lever, leading to an accident. The safety version also demonstrated a preliminary check of the internal health of all the RCCS subsystems -the gates mechanism, track lever operation, sensors, signal light LEDs, displaying the health status on the LCD display panel.

14 Conclusion

This paper discussed a FMEA and Fault Tree based approach to software safety analysis for critical systems. A comprehensive software safety analysis involving a combination of FMEA and FTA techniques was conducted on the software functions of the critical system to identify potentially hazardous software faults. The safety properties of the prototype railroad crossing control system were identified as part of the safetycritical requirements. These safety requirements were incorporated in the design and development of a railroad crossing control system (RCCS). We also briefly compared safety-specific and non-safety specific techniques at developing RCCS. The non-safety version of RCCS broadly focused on achieving the functional behavior of the system. The safety-specific version clearly demonstrated that the software safety properties identified in RCCS specification were fully met in the working system. [1] [2] [3] [4]

Figure 1:

Figure 2:

Figure 3:

3

Figure 4: Figure 3 .

4

Figure 5: Figure 4 .

2

Failure Mode	Possible Causes	Effect	Sever- ity of risk	Prevention And Compensati on
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

Figure 6: Table 2 .

Figure 7: ?

[Leveson and Turner (1987)] 'An investigation of the Therac-25 accidents'. N G Leveson , C S Turner . *IEEE Computer* March 1987. 26 (7) p. .

[DO-178B/ED-12B Software Considerations in Airborne Systems and Equipment Certification (1992)] *DO-178B/ED-12B Software Considerations in Airborne Systems and Equipment Certification*, (RTCA, EUROCAE) December 1992.

[EN50128 Railway Applications: Software for Railway Protection and Control Systems. CENELEC] *EN50128 Railway Applications: Software for Railway Protection and Control Systems. CENELEC*,

[Gray (2000)] Dale M Gray . `www.asi.org` *Frontier Status Report #203*, 19 May 2000.

[Herman ()] Debra S Herman . *Software Safety and Reliability Basics:", (ch.2), Software Safety and Reliability: Techniques, Approaches, and Standards of*, 2000. Key Industrial Sectors Wiley-IEEE Computer Society Press.

[Ieee Std] Ieee Std . *Standard Glossary of Software Engineering Terminology*, 610 p. .

[Ieee Std 1012 ()] Ieee Std 1012 . *IEEE Standard for Software Verification and Validation Plans*, 1986. The Institute of Electrical and Electronics Engineering, Inc. USA

[N ()] *Leveson Safeware: System Safety and Computers*, N . 1995. Addison-Wesley.

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

[Lutz ()] 'Software Engineering for Safety: a Roadmap'. Robyn R Lutz . *Proceedings of the Conference on The Future of Software Engineering*, (the Conference on The Future of Software EngineeringLimerick, Ireland) June 04-11, 2000. p. .

[Tribble ()] 'Software Safety Analysis of a Flight Guidance System'. Alan C Tribble . *Proceedings of the 21st Digital Avionics Systems Conference (DASC'02)*, (the 21st Digital Avionics Systems Conference (DASC'02)Irvine, California) Oct. 27-31, 2002.

[Gleick (1996)] *The New York Times Magazine*, James Gleick . 1st December 1996.