International Journal of Software Engineering and Its Applications
Vol.12, No.3 (2018), pp.19-28
http://dx.doi.org/10.21742/ijseia.2018.12.3.02

Software Failure Analysis using FMEA

Victoria K. Lazarus[1] and Shawulu Hunira Nggada[2]

[1]Department of Computer Science, Faculty of Computing and Informatics, Namibia University of Science and Technology, 13 Storch Street, Windhoek, Namibia
[2]Computer Information Science, Higher Colleges of Technology, Ras Al Khaimah Women's Campus, Ras Al Khaimah, United Arab Emirates
[1]victoria.vicky.lazarus@gmail.com, [2]snggada@hct.ac.ae

Abstract

The reliability of safety-critical systems is of concern, a failure of which would result in injury, death, damage to the environment or financial loss. Such systems have evolved from being largely mechanical to computer driven. The approach to analysing failure of mechanical or hardware architecture of the system is well established in literature and practice. Although the hardware architecture could be adequately analysed for failure using traditional safety analysis techniques, the manner in which the software architecture is to be analysed for failures is fuzzy. This creates additional concern to the reliability of contemporary safety-critical systems. This paper defines an approach to analysing software failure at class diagram using one of traditional safety analysis techniques, failure modes and effects analysis (FMEA). It also demonstrates how to apply the approach to analysing software failure.

Keywords: *Software failure, software quality, failure analysis, reliability, safety-critical systems, FMEA*

1. Introduction

Safety-critical system is one whose failure could result in damage to properties, damage to the environment, financial loss, injuries to humans, injuries to animals, loss of human lives, and loss of animal lives. If the failure of the system could lead to any consequences that is determined to be unacceptable then the system is termed as a safety-critical system [1]. The operations of modern systems are computer controlled and this new approach seems to be the future. Therefore, it is imperative that the software layer of the systems is analysed for failure to allow for correction of foreseen or identified potential hazards associated with the given system. Some systems were once completely mechanical in nature, such as the hydraulic brake system of an automotive system, sequel to the proliferation of advances in computing, the design of such system include a computer controlled option; brake-by-wire (BBW).

According to Lang, Roberts, Jung, Fiedler and Mayer [2], BBW system is used in all common hybrid and electric vehicles produced since 1998. In the automotive industry, BBW technology is the ability to control brakes through electrical means. It can be designed to supplement ordinary service breaks or it can be a standalone brake system. In a BBW car, some sensors are safety-critical components; their failure will result in consequences that are unacceptable. Two examples of such components are brake pedal sensor and wheel speed sensors. The electronic control unit must always be informed of the driver's intentions such as to reduce speed or to stop the vehicle. Therefore, missing the pedal sensor data is a deviation of intended design of the functionality of the vehicle control system.

Received (March 8, 2018), Review Result (July 25, 2018), Accepted (August 2, 2018)

ISSN: 1738-9984 IJSEIA

Although engineers put great amount of efforts into designing systems with improved reliability, failures would still occur because of human behaviour or occurrence of hazard that was not identified and mitigated at design stage. Skarin and Karlsson [3] presented a study of fault injection results on an automotive BBW system to determine if random faults can cause an unsafe situation. They found that 30% of random errors caused erroneous outputs, and of those 15% resulted in critical braking failures. Skarin and Karlsson [3], again in the context of this BBW study, critical failures were either loss of braking or a locked wheel during braking. These sorts of errors can lead to critical operational faults in electronic automotive controls. Typically, systems need to be analysed at design time to find root causes of systems failures. Techniques such as fault tree analysis (FTA) and Failure Modes and Effects Analysis (FMEA) are typical examples of tools that are commonly used.

2. Software Failure

Systems in organisations are developed to perform certain functions specified by the client or as required by the business. However, there can be reasons that some of these organisational functions are not achieved due to some faults which later result in failures. Software failure can lead to degradation in performance to end-users and can result in losses to the business.

An obvious question when studying failures in system software is, what causes system failures? A system failure can occur because of a hardware failure or a severe software issue; causing the system to freeze, reboot, or stop functioning altogether [4]. To evaluate the overall system reliability, all the components in the system should be considered. Some of the causes of hardware and software failures are discussed in the next section.

2.1. How System Failures Occur

An interesting view was expressed by Avizienis, Laprie and Randell [5], that a system failure occurs when the delivered service no longer complies with specifications. The specifications being an agreed description of the system's expected functions and, or services. To understand the analysis of software failure, it is useful to first look at the definition of three system anomalies [6]. These anomalies are fault, error and failure, and are defined according to Storey [7] as follows.

- A fault is a weakness or defect in a system.
- An error is a deviation from the required function of a system.
- A failure occurs when a system fails to perform its required function.

3. Existing Approaches to Improving Software Reliability

According to Cranwell [8], software reliability is a probability that software will not cause the failure of a system for a specified period of time under specified conditions. The probability is a function of the inputs to and use of the system as well as a function of the existence of faults in the software. Various approaches have been used to improve reliability of software through minimising of faults and errors, or tolerance. Few of such are discussed next.

3.1. Defensive Programming

In defensive programming, design patterns, coding styles and software libraries are developed that enable programmers to minimise the time interval between (i) the point of execution at which the symptoms/effects of errors, inherent in the software design and

coding processes, become somehow observable, and (ii) the exact time of execution of the first offensive statement within the source code that actually causes such errors [9].

According to Jones [10], two examples of defensive programming that are now commonplace are (i) watchdog timers which are common component of embedded systems that are designed to restart software and, or hardware after identifying anomalous behaviour, and (ii) checksums which are common elements of packets transferred between nodes to identify errors induced during communication. There is also the risk that the code catches or prevents too many exceptions. In such cases, the error would be suppressed and go unnoticed, while the result would still be wrong.

3.2. Pair Programming

Venkatesan and Sankar [11], describe pair programming as a software failure analysis technique that requires two programmers working on the same problem in front of the same screen using a set of input devices. While one developer is modifying the source code, another is required to perform continuous code-review. Williams, Kessler, Cunningham, and Jeffries [12], suggested that pair programming analysis help to produce high software quality. Williams, *et al.*, [12], again added that, pair programming speeds up the development but generally requires more efforts.

3.3. Exception Handling

Exception handling is the process of responding to the occurrence, during computation, of exceptional conditions requiring special processing often changing the normal flow of program execution [13]. It is provided by specialised programming language constructs or computer hardware mechanisms. The programmer anticipates such deviation from normal or intended operation and then provides mitigating code that would respond to the anomaly during execution. Some programmers write software with error reporting features that collect details that may be helpful in fixing the problem, and display those details on the screen, or store them to a file such as a core dump, or in some cases an automatic error reporting system. An example of an automatic error reporting system is Windows Error Reporting which can automatically phone home and email those details to the programmers. Exception handling gives programmers an alternative way to deal with error-prone situations in their code.

4. Complementary Approach to Improving Software Reliability

One of recent approaches to improving software reliability is by analysing it for failure at top level of software architecture such as use case diagram using failure modes and effects analysis (FMEA) as demonstrated by Nggada [6]. The analysis first established a model for failure analysis at architecture level of software and used BBW as case study where an example use case of the BBW is developed. The analysis involved identification of potential hazards for each actor and use case where the failure analysis is performed on them. The analysis borrowed techniques as used in traditional engineering and thus, mitigating interventions that would reveal further design considerations to improve dependability of software systems is provided. In addition, it would be helpful if the above approach is extended to demonstrating how the technique could be applied to another level of software architecture; such as class diagram.

Riva, Selonen, Systä and Xu [14] stated that class diagram is one of the most commonly used diagrams in object-oriented systems, and models the static design view for a system. Riva, Selonen, Systä and Xu [14] further expressed that the static view mainly supports the functional requirements of a system; the services the system should provide to the end users. A class diagram shows a set of classes, interfaces, and collaborations and their relationships.

Class diagrams also consist of global system description such as the system architecture, and detail aspects such as the attributes and operations within a class as well.

4.1. Case Study

The case study used by Nggada [6], is adopted for this work and its description which is also largely taken from Nggada [6], is as follows. The BBW system is an electric/electronic system which can be controlled by computer. According to Nggada [6], embed of software offers the possibility to introduce functions that were either originally impossible or costly with mechanical or hydraulic system components. It also reduces size and weight.

A BBW system is based on time-trigged protocol, nodes, sensors and actuators. The BBW system transfer electrical signals down to a wire instead of using hydraulic fluid. If no hydraulic back-up is available, it is a pure Brake by Wire system (Electro Mechanical Brake system) and it is very important that the system must continue to function in the event that fault occurs.

The simple form of the BBW is as shown in Figure 1 and it is described as follows. The BBW consists of a central controlling unit known as vehicle control unit (VCU) and one brake control unit (BCU) per wheel. The VCU reads as input the braking pressure applied on the brake pedal. It then processes this pressure to send signal to each BCU about the amount of braking pressure to be applied on the respective wheels. Each BCU further processes this signal taking into account wheel conditions in order to establish the needed amount of braking pressure.

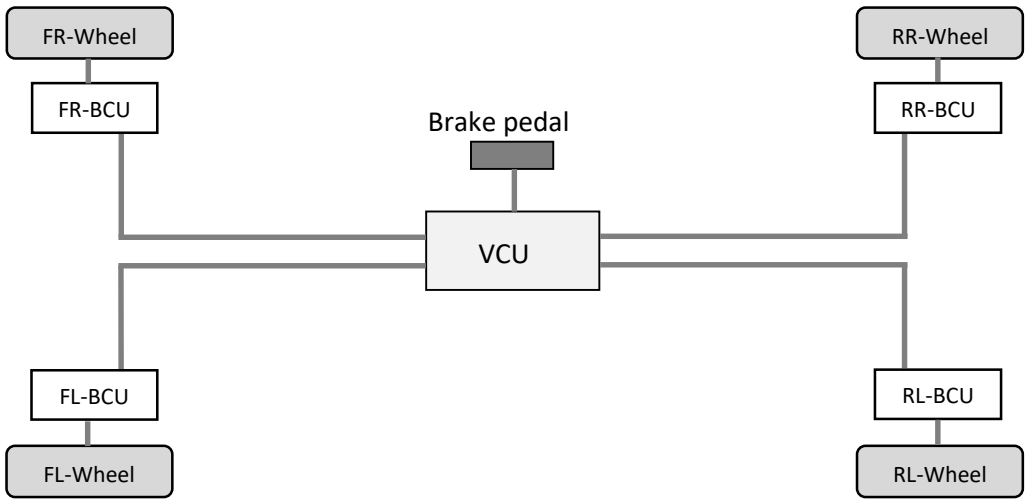

Figure 1. Simple BBW System Adapted from Nggada [6]

Where:

FL-Wheel refers to front left wheel

FR-Wheel refers to front right wheel

RL-Wheel refers to rear left wheel

RR-Wheel refers to rear right wheel

FL-BCU refers to front left wheel brake control unit

FR-BCU refers to front right wheel brake control unit

RL-BCU refers to rear left wheel brake control unit

RR-BCU refers to rear right wheel brake control unit

4.2. Class Diagram

Dependability analysis is performed on a system model and therefore, in order to perform the analysis on the class diagram, such class diagram needs to be established. The class diagram of the BBW is as seen in Figure 2. The relationship between the Driver and the VCU is one to one. The driver sends braking request including braking pressure to the VCU. Similarly, the VCU sends the braking pressure to the respective BCUs and applies the brake through the respective BCUs; a generalisation relationship exists between the VCU and BCU which indicates that there are four BCUs which are types of the VCU. The BCU sends the braking pressure to the wheels to stop or slow down the car. Each BCU is attached to its corresponding wheel and where each corresponding wheel is a type of an abstract wheel. Each implementation of the wheel has its record of wear and tear status and tyre pressure. Additionally, each implementation applies the actual brake through actuators which does not appear in the Figure. It is also worth mentioning that all the classes including their attributes and operations for the BBW will be impossible to represent in this paper due to complexity and space. However, those which appear are considered sufficient to demonstrating the applicability of FMEA on class diagram.

4.3. Failure Analysis at Class Diagram

In this work FMEA is used to analyse the BBW system's braking failure mode at the level of the system architecture in the form of class attributes and methods. The approach employed in the analysis using FMEA is similar to that used in the failure analysis of hardware systems; classes are used in place of components. The storage of message or certain class characteristics is within the attributes of the class and therefore, the possible failure of such attributes is also of interest. Similarly, communication within a class and, or among classes is possible via operations of classes, hence, the analysis of such is also of interest. Failure of attributes and, or operations could impact on the host class and, or the system. Therefore, effect of such failure on the class and the system is analysed.

A number of different failure modes for attributes and operations as specific to the given case study were identified as seen in Table 2 under "Failure" column. It should be noted that various failure modes within the context of a given problem may vary. Additionally, these should correlate with the system top event (system level failure mode). Although class level failure modes for the BBW were identified, this does not imply that they have been identified exhaustively. This justifies the need to involve several dependability experts in the failure analysis of a given system. However, those identified in this paper are sufficient to demonstrating the applicability of failure analysis at class diagram.

The software failure analysis of the BBW in Table 2 contains the BBW classes which are (Driver, VCU, BCU and Wheel). Each class has attributes and, or operations on which failure could be analysed. The analysis further indicates effect that the failure has on the component and the system, and finally the mitigation to the cause of failure.

The analysis in Table 2 reveals that class attributes are critical to the correctness of system operations. Quite a number of the failures point to the attributes, even operations specific failures.

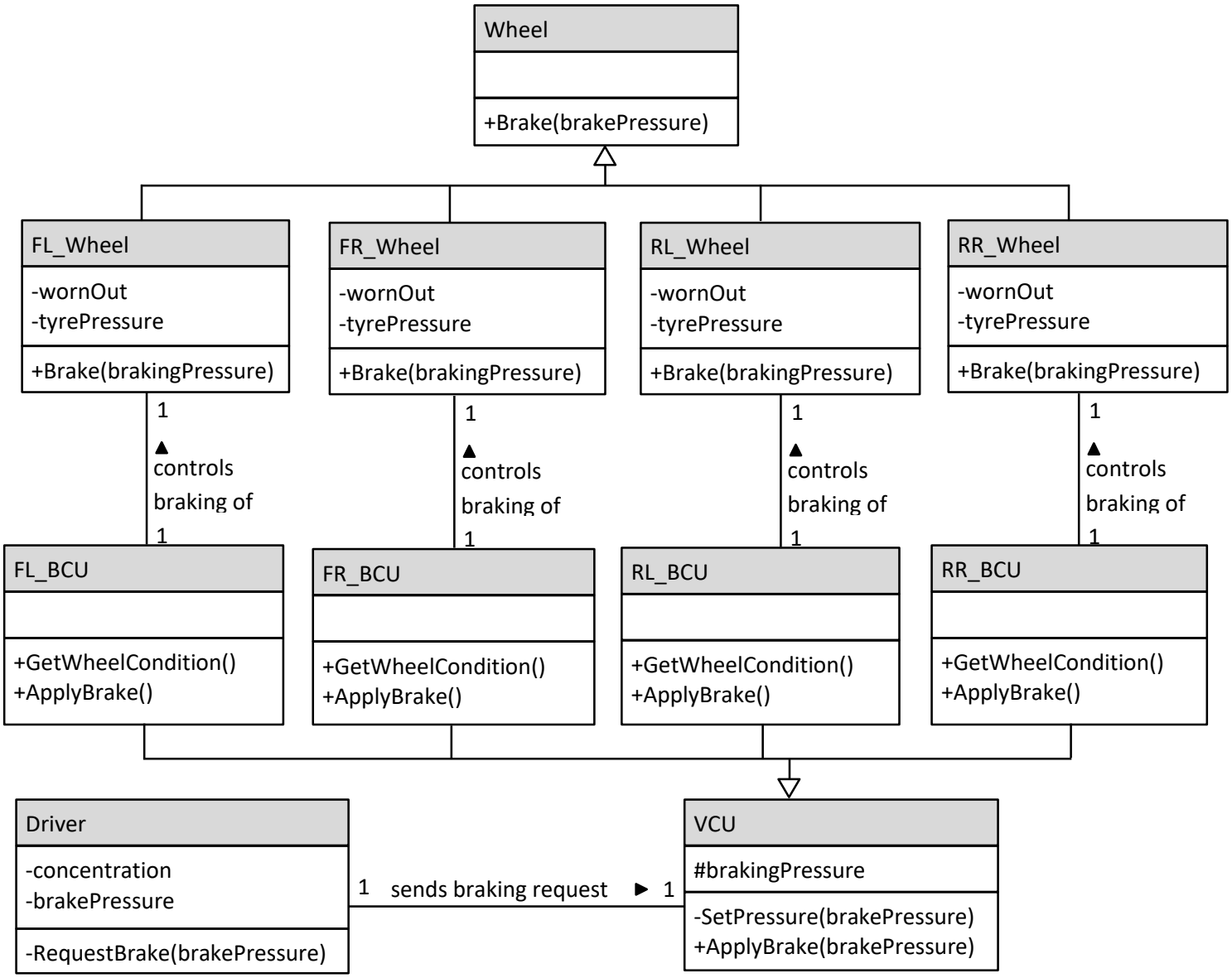

Figure 2. Simple BBW Class Diagram

Table 1. BBW Failure Analysis of its Class Diagram

System Failure Mode: Braking Failure					
Class	**Attributes/ Operations**	**Failure**	**Effect the Class**	**System Effect**	**Mitigation**
Driver	concentration	Low value	None	Likely Accident	Provision of a device to monitor the level of concentration to alert the driver.
		High value	None	Appropriate braking	None
	brakePressure	Omission	None	No retardation	Provision of a device to monitor the level of Concentration and surrounding objects (vehicles, etc) to alert the driver.
		High value	None	Sudden braking	Provision of a device to monitor surrounding objects and to apply appropriate braking.
		Low value	None	Slow braking	Provision of a device to monitor surrounding objects and to apply appropriate braking.
	RequestBrake()	Argument out of range	Loss of appropriate value for braking pressure as applied by the driver	Application of inappropriate brake or no brake at all	Since this is critical, a code that will validate the value would be necessary. In the case of an invalid range, the use of a device that will suggest the hazard (e.g. distance) and apply the appropriate braking would be helpful.
VCU	brakingPressure	Omission	No appropriate value for braking pressure	No retardation	The use of sensor to suggest the hazard (e.g. distance) and apply the appropriate braking.
		High value	None	Sudden braking	Provision of a device that can detect surrounding objects and apply appropriate braking.
		Low value	None	Slow braking	Provision of a device that can detect surrounding objects and apply appropriate braking.
	SetPressure()	Argument out of range	Loss of appropriate value for braking pressure as applied by the driver	Application of inappropriate brake or no brake at all	Since this is critical, a code that will validate the value would be necessary. In the case of an invalid range, the use of a device that will suggest the hazard (e.g. distance) and apply the appropriate braking would be helpful.
	ApplyBrake()	Argument out of range	Loss of appropriate value for braking pressure as applied by the driver	Application of inappropriate brake or no brake at all	Since this is critical, a code that will validate the value would be necessary. In the case of an invalid range, the use of a device that will suggest the hazard (e.g. distance) and apply the appropriate braking would be helpful.

FL_BCU	GetWheelCondition()	No condition information	Unaware of the condition of the wheel; e.g. wear and tear, tyre pressure, etc	No application of appropriate brake to the wheel taking into account its condition	An additional operation such that it would routinely check the condition could be created. If the condition information on demand fails, the last known value could be used. Additionally, should the routine check fail to obtain condition information after few checks, the driver should be alerted and advised for maintenance.
	ApplyBrake()	Braking pressure value out of range	Loss of appropriate value for braking pressure as applied by the driver	Application of inappropriate brake or no brake at all	Since this is critical, a code that will validate the value would be necessary. In the case of an invalid range, the use of a device that will suggest the hazard (e.g. distance) and apply the appropriate braking would be helpful
FR_BCU	Reuse(FL_BCU)	Reuse(FL_BCU)	Reuse(FL_BCU)	Reuse(FL_BCU)	Reuse(FL_BCU)
RL_BCU	Reuse(FL_BCU)	Reuse(FL_BCU)	Reuse(FL_BCU)	Reuse(FL_BCU)	Reuse(FL_BCU)
RR_BCU	Reuse(FL_BCU)	Reuse(FL_BCU)	Reuse(FL_BCU)	Reuse(FL_BCU)	Reuse(FL_BCU)
FL_Wheel	wornOut	Braking problems on the wheel	Braking malfunctions	Irregular retardation	In addition to the routine check for condition information that is recommended previously, an additional operation that would validate the condition of the wheel in terms of acceptable state should be provided. If the condition is unacceptable, then immediate maintenance should be recommended or ground the vehicle. However, initial warnings must have been issued when the condition approaches the threshold
	tyrePressure	Incorrect pressure	Braking malfunctions	Irregular retardation	Same as in above (wornOut)
FR- Wheel	Reuse(FL_Wheel)	Reuse(FL_Wheel)	Reuse(FL_Wheel)	Reuse(FL_Wheel)	Reuse(FL_Wheel)
RL- Wheel	Reuse(FL_Wheel)	Reuse(FL_Wheel)	Reuse(FL_Wheel)	Reuse(FL_Wheel)	Reuse(FL_Wheel)
RR- Wheel	Reuse(FL_Wheel)	Reuse(FL_Wheel)	Reuse(FL_Wheel)	Reuse(FL_Wheel)	Reuse(FL_Wheel)

5. Conclusions

Achieving acceptable level of software reliability is paramount to software related industries. However, software and hardware are becoming indispensable; therefore, hardware reliability is also of interest. For hardware systems, established approaches or techniques on how failure could be analysed exists, this is however absent in software. An established procedure on how software could be analysed for failure right from the infancy stage of development would be helpful in minimising the occurrence of software failures.

This paper utilised an established approach, failure modes and effects analysis (FMEA) used in hardware failure analysis to establish similar approach to analysing software systems. The performed analysis in this work demonstrates that the use of FMEA to analyse failure of software systems is possible and the results of the analysis such as mitigations would reveal further design considerations to improve dependability of software systems. Thus, the contribution of this work is that it has demonstrated that software failure analysis could be performed at class diagram using FMEA.

References

[1] J. C. Knight, "Safety Critical Systems: Challenges and Directions", Proceedings of the 24th International Conference on Software Engineering, **(2002)**, pp. 547-550.
[2] H. Lang, R. Roberts, A. Jung, J. Fiedler and A. Mayer, "The Road to 12V Brake-by-Wire Technology", VDI Berichte, **(2006)**, pp. 55-71.
[3] D. Skarin and J. Karlsson, "Software Implementation Detection and Recovery of Soft Errorsin a Brake-by-wire System", In the 7th European Dependable Computing Conference (EDDC-07), **(2008)**, pp. 145-154.
[4] I. Sommerville, "Software engineering", 9th Ed, Pearson, New York, **(2011)**.
[5] A. Avizienis, J. C. Laprie, B. Randell and C. Landwehr, "Basic Concept and Taxonomy of Dependable and Secure Computing", IEEE Transactions on Dependable and Secure Computing, vol. 1, no. 1, **(2004)**, pp. 11-33.
[6] S. H. Nggada, "Software Failure Analysis at Architecture Level using FMEA", International Journal of Software Engineering and Its Applications, vol. 6. no. 1, **(2012)**, pp. 61-74.
[7] N. Storey, "Safety-Critical Computer Systems", Addison Wesley Longman, London, **(1996)**.
[8] R. M. Cranwell, "Ground Vehicle Reliability – Design for Reliability", DoD Maintenance Symposium, Orlando, **(2007)**, Retrieved from http://www.sae.org/events/dod/presentations/2007.htm.
[9] A. Jonsson, U. Winberg, C. G. Lignell, C. M. Lee and P. Larsen, "Method and Apparatus for Handling Processing Errors in Telecommunications Exchanges", U.S. Patent 5,594,861, **(1997)**.
[10] M. T. Jones, "Defensive Programming", **(2005)**, Retrieved from http://www.drdobbs.com/defensive-programming/184401915.
[11] V. Venkatesan and A. Sankar, "Adoption of Pair Programming in the Academic Environment with Different Degree of Complexity in Students Perspective - An Empirical Study", International Journal of Engineering, Science and Technology, vol. 2, no. 9, **(2010)**, pp. 4791-4800.
[12] L. Williams, R. R. Kessler, W. Cunningham and R. Jeffries, "Strengthening the Case for Pair Programming", IEEE Software, vol. 17, no. 4, **(2000)**, pp. 19-25.
[13] W. Weimer and G. C. Necula, "Exceptional Situations and Program Reliability", ACM Transactions on Programming Languages and Systems, vol. 30, no. 2, **(2008)**.
[14] C. Riva, P. Selonen, T. Systä and J. Xu, "UML-based Reverse Engineering and Model Analysis Approaches for Software Architecture Maintenance", Software Maintenance, 2004. Proceedings of the 20th IEEE International Conference, **(2004)**, pp. 50-59.

Authors

Victoria K. Lazarus, received Bachelor Honours Degree in Business Information Systems from the International University of Management, Windhoek, Namibia, in 2012, worked with the ministry of education in Namibia as an IT Technician from 2012 to 2013. Since then she has been working with International University of Management and she is currently a lecturer at the institution. She pursued a Masters degree in Computer Science at Namibia University of Science and Technology.

Shawulu Hunira Nggada has PhD in Computer Science and PGDip in Research Training both from University of Hull - UK, MSc Software Engineering from University of Bradford - UK and B. Tech (Hons) in Computer Science from Abubakar Tafawa Balewa University, Nigeria. He has Chartered IT Professional membership of BCS The Chartered Institute for IT, Chartered Engineer membership of Engineering Council – UK. He is a Senior Member of IEEE. He is currently a Faculty at Higher Colleges of Technology, UAE. He previously held a Faculty position at Namibia University of Science and Technology.