Software Failure Analysis at Architecture Level using FMEA

Shawulu Hunira Nggada

Department of Computer Science, University of Hull, HU6 7RX, United Kingdom
shnggada@googlemail.com, s.h.nggada@2007.hull.ac.uk

Abstract

The advancement and proliferation of information technology has made it possible for specified functions of systems including safety-critical systems to be software driven. Traditional failure analysis techniques existed before computers and are widely used in the failure analysis of hardware. Typically, hardware failures are random while software failures are systematic and this makes software failure analysis difficult to be addressed. However, similar approaches used in hardware failure analysis can be applied in the failure analysis of software at its architecture level. Such analysis informs design modifications in software and likely hardware to mitigating design weaknesses. This paper investigates this approach by employing the use of FMEA and emphasizes on the commencement of failure analysis at early system design stage. Thus, weaknesses in the design can be identified early and necessary interventions taken. The FMEA investigates failure of each entity of the architecture relative to a defined system top event.

Keywords: *Software engineering, failure analysis, use case, software architecture*

1. Introduction

Advancement in computing has made it possible for hand held devices, household equipments, automobiles, nuclear power plants, aircrafts, spacecrafts, etc to have embeds of programmable devices. In other words, these systems are computer controlled through software (computer programs). The level of computer control will vary from one system to another depending on the scale of computer control required and, or size of the system. The scale of computer control required and the size of the system both add to the design complexity of the system. In some of these systems, a failure in the software will have impact on the system's reliability (e.g. mobile phones and digital cameras) while in some systems it has severe impact on their safety (e.g. automobile and aircrafts). This also suggests that in general, a failure can have a negative impact on at least one of three dependability attributes (reliability, availability and safety) of the system in question.

Systems like aircrafts, nuclear power plants, etc that cause hazards for people and the environment are termed as safety-critical systems [1]. Although it is logical to invest more in the failure analysis of safety-critical systems, in general an in-depth failure analysis of any given system will reduce manufacturing cost that may be incurred at the following development phases; design, implementation and post-implementation (e.g. customer support and maintenance – repair and replacement). It is therefore of significant importance that systems are analyzed for potential failure from the design stage to completion. The commencement of failure analysis right from the infancy or early design stage will help ensure that identified weaknesses in the design are mitigated. The analysis of failure should normally be reviewed following mitigation. This way, potential hazards which might have previously been overlooked are likely to be identified.

According to Haapanen and Helminen [2], the failure modes of the constituent components of mechanical and electrical systems are normally well understood. This is because the reasons for failures are known and their sequences may be studied; some of these reasons are wear, aging and unanticipated stress. However, this does not suggest that the failure analysis of such systems is always easy, but in essence is straightforward. In contrast, the failure modes of software for software-based systems are generally unknown.

Software engineering does not only advocate for the development of software that meet user requirement but also one which is dependable as is the case for safety-critical systems. This paper investigates the failure analysis of software at its architecture level by employing a traditional failure analysis technique used for mechanical, electrical and electronic systems. In section 2 a review on system failure analysis via traditional technique such as failure modes and effects analysis (FMEA) is discussed. It also contains investigation on how FMEA could be applied in software systems. Section 3 investigates the architecture of software, a model on which this work attempts to apply FMEA. It also establishes and proposes a failure analysis model for software architecture. Section 4 presents the case study on which the software failure analysis is performed – brake by wire system. The analysis is contained in section 5 and conclusion is drawn in section 6.

2. Systems Failure

There is currently no study in literature which suggests that systems can be designed and operate 100% failure free. Certain failures occur due to design flaws and oversight in hazards identification. The identification of system and component failure modes and their causes is subject to the judgement of the system engineers involved in their failure analyses. In addition, as the size or the volume of operation of a system increases its design complexity increases and so does the difficulty in identifying design weaknesses (flaws and hazards). Having ascribed the occurrences of certain failures to design flaws and oversight in hazards identification, some failures occur due to human factor. However according to Leveson [3], all human behavior is influenced by the context of the system being interacted with, and operators of such system are often at the mercy of the design of the automation they use. Hence, many occurrences of systems failure could be as a result of poor design.

Although the focus of this paper is on software failure analysis using FMEA, first the use of traditional FMEA is explained using a simple hardware example.

2.1. Failure Modes and Effects Analysis (FMEA)

FMEA is a systematic procedure for the analysis of a system to identify the potential failure modes, their causes and effects on system performance [4]. A typical system has one or more potential failure modes. Anticipating these failure modes which is central to the analysis needs to be carried out extensively in order to prepare a list of maximum potential failure modes [5]. The causes of these modes could be seen at component levels which propagate through the system and eventually leading to system failure. To illustrate failure analysis using FMEA where similar approach is employed in analyzing software failure in section 5, Figure 1 is here considered. Figure 1 is a simple light system which in this case is sufficient for the required illustration.

Figure 1 consists of a battery which powers the lamp to glow when the switch is closed. Port B-Port1 of the battery is connected to the switch through Line 1, and the switch to port L-Port1 of the lamp through Line 2. Port L-Port2 of the lamp is connected to B-Port2 of the battery through Line 3. Assuming the objective of the design of the simple light system in

Figure 1 is to provide a certain intensity level of glow then its failure mode can exist in several forms such as no glow, low glow, high glow, intermittent glow, etc. One system failure mode "no glow" is here considered where the components are analyzed relative to this failure mode. The FMEA analysis is presented in Table 1.

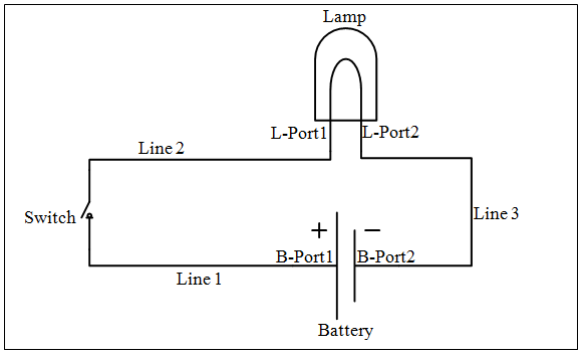

Figure 1. Simple Light System

Table 1 – Basic FMEA Analysis of Figure 1

System Failure Mode: *No Glow*			
Component	**Potential Causes**		**System Effect**
Battery	Battery dead		No flow of current
	No connection	B-Port1 – Switch	Open circuit
		L-Port2 - B-Port2	Open circuit
Switch	Opened		Open circuit
	Switch faulty		Open circuit
	No connection	B-Port1 – Switch	Open circuit
		Switch - L-Port1	Open circuit
Lamp	Lamp dead		No flow of current
	No Connection	Switch - L-Port1	Open circuit
		L-Port2 – B-Port2	Open circuit

Table 1 shows the failure mode being considered for the system and its components. It also shows for each component the potential causes that lead to the failure mode. Finally it shows the effects of these potential causes on the system. Among other attributes, a more detailed FMEA analysis will include preventive or mitigating measures for the potential causes. A modification of the system design will be based on the mitigating measures and when they are included in the modification, FMEA is reapplied. This way, FMEA is an iterative process that is updated as the design evolves [4]. Failure analysis in other fields of engineering using FMEA is well established. This is however not true in software engineering. In order to address this, the manner in which software can fail is discussed next.

2.2. Failure in Software

The manner in which software fails differs from that of hardware. Sequel to the fact that hardware (e.g. in mechanical, electrical and electronics) failure analysis has existed for long, engineers are accustomed to the failure behaviour of these systems than to software. Additionally unlike hardware failure, software failure occurs due to systematic faults and in which case it may be difficult to apply statistical analysis. To understand the analysis of software failure, it is useful to first look at the definition of three system anomalies; fault,

error and failure. The definitions of these anomalies are clearly stated in [1] and are here used. A *fault* is a weakness or defect in a system while an *error* is a deviation from the required function of a system. A *failure* occurs when a system fails to perform its required function. These anomalies are not only confined to system level but also to component and subroutine levels which will eventually affect the system's normal operation. To further clarify these definitions, a simple example is here considered. Assuming the code (in C++) below is a function that is intended to return the complete result (including fraction part) of the division of two integers i.e. if param1 = 3 and param2 = 2 then the result of the division param1/param2 should give 1.5 and not 1.

```
double twoIntDivision(int param1, int param2)
{
    return param1/param2;
}
```

As one would expect, the above function will always return an integer quotient of the division because the integer division would require a typecast that is same as the function return type. Thus, this could be a design weakness and therefore it is a fault lurking in the system. A call to the function with arguments whose division does not evaluate into a whole number will result into an error; a deviation from the required function. If the evaluation of this function is used somewhere in the system, for instance in a financial calculation, this could lead to a system failure; e.g. financial loss. It is also clear that the existence of a fault will lead to an error when a demand in function is requested from the code segment containing the fault. The error in-turn causes a system failure. In general, software is implemented based on its design. Hence, a software failure is the product of a design flaw (fault or weakness). It is therefore imperative that the design of the software is analyzed for potential failures right from the infancy stage of the design. This paper addresses the analysis at the earliest possible architecture level of the software and thus software architecture is discussed next.

3. Software Architecture

Software architecture refers to a structured conceptual representation of a software system. Such representation includes different entities, nature of data used and the interactions among these entities. One goal of software design is to derive an architectural rendering of a system which serves as a framework from which more detailed design activities are conducted [6]. This implies that an earliest level of software architecture should define the top hierarchical or modular components of the system that are sufficient to represent the system. The details of each modular component could then be addressed in further design. Also according to Pressman [6] the software architecture is not the operational software but a representation that enables a software engineer to (i) analyze the effectiveness of the design in meeting its stated requirements, (ii) consider architectural alternatives at a stage when making design changes is still relatively easy, and (iii) reducing the risks associated with the construction of the software.

The involvement of a user of the system right from the conceptual view of the system will help in reducing later adjustments to the design of the system. Hence two levels of architectural designs can be enumerated; a *user* and *developer* level. At the user level, the design is in the highest possible architectural representation of the system which can easily be communicated to the user. This way the user can easily point out some design oversights. The developer level involves a detailed technical representation of each of the modular entities

found at the user level. Hence the user level provides an insight into the technicalities required for the system. Example of user level architecture is *use case diagram* while that of developer level is *class diagram*, *state diagram*, *sequence diagram*, *interaction diagram*, etc. In practice, the choice and number of architectures for a software project is problem specific and may also rely on developer intuition or experience. Failure analysis should be performed at all levels of software architecture being considered for a given project. For instance the topmost level (i.e. level 1) could be *use case diagram*, followed by *class diagram* (i.e. level 2), etc as intuitively identified by the software engineer concerned. A failure analysis model for software architecture is established, proposed and presented in Figure 3.

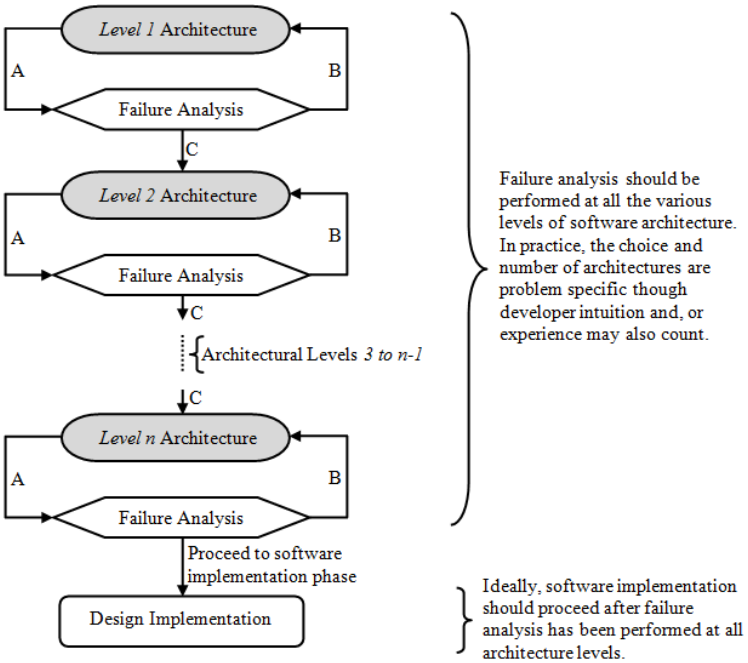

Figure 3. Failure Analysis Model for Software Architecture

Descriptions of arrows used in Figure 3

A: Perform failure analysis for the current architecture

B: Modify and evaluate system architecture in according with failure analysis

C: Proceed to the next architecture level

At each architecture level of design, failure analysis is performed and where design weaknesses or flaws are indentified the system is evaluated for quality/design objectives taking into account the proffered mitigations (interventions). If *j* is the *j-th* architecture level, then the software design proceeds to architecture level *j+1* once failure analysis has been performed and the system architecture has also been modified and evaluated at level *j*. This paper focuses on the failure analysis of a software system at the user level architecture and the brake by wire system is here considered as a case study to demonstrate the applicability of FMEA on the user level architecture.

4. Case Study – Brake by Wire System

The brake by wire (BBW) system in the context of automotive systems refers to the concept where mechanical or hydraulic system is replaced by electric/electronic systems [7]. The electric/electronic systems are computer controlled and hence are made up of embedded software. 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. However, with the brake by wire system still being new, a mechanical/hydraulic system may be used in conjunction with the brake by wire system. This can be used as a backup to strengthen safety measures. Two types of brake by wire systems exist, the *wet brake by wire* and the *dry brake by wire* system [8]. The former is a combination of the electronic brake system and the hydraulic brake system as a backup while the latter represents systems consisting of electronic brake system where no master cylinder or hydraulic lines are needed and therefore there is no mechanical backup. Sequel to the focus of this paper on software failure analysis, the dry brake by wire system is used and subsequent references to brake by wire will imply the dry brake by wire system. The challenge of computer controlled systems is that they introduce new modes of failure that is unfamiliar in hardware failure analysis. To demonstrate the software failure analysis of the BBW, BBW is first introduced and then its user level software design is presented from where the analysis is conducted.

The brake by wire system considered in this paper is similar to the one described in Wilwert et al [9]. The BBW is designed to increase the quality of braking by reducing the stopping distance. The simple form of the BBW is as shown in Figure 3 and 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. One of environmental advantages or friendliness of the BBW is that no braking fluid is necessary [8, 9].

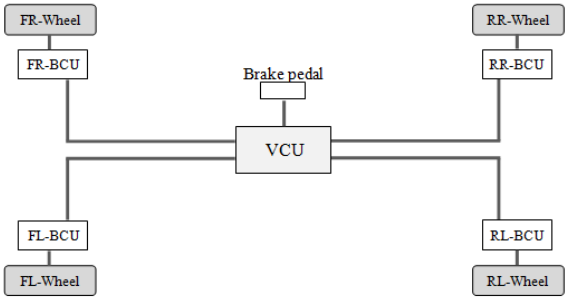

Figure 3. Simple Brake by Wire System

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

— Data (or signal) channel

5. Analysis and Results

To analyse the BBW system, a use case diagram (i.e. level 1 architecture) for the system needs to be drawn. This is presented in Figure 4. It consists of an actor (which is a driver) and 15 use cases. The prefixes FL, FR, RL and RR which appear in some of the use cases infer *front left*, *front right*, *rear left* and *rear right*. Each of the entities (actor and use cases) of the use case diagram will be analyzed for software failure. To proceed with the analysis, a system failure mode needs to be defined. This paper defines a system failure mode referred to as *braking failure*. It should be noted that this is different from *brake failure* in that the term brake failure may refer to the inability of the brake system to deliver its function on demand. However, the manner in which the demand is requested may as well count. To this effect braking failure would mean that one of the following occurs when the brake is applied, (i) vehicle stops too early, (ii) vehicle stops too late, and (iii) the brake system fails to deliver its function – implying brake failure as explained earlier. It can be observed that the FL-Wheel, FR-Wheel, RL-Wheel and RR-Wheel are specialized form of the Wheel use case. Though all are wheels one cannot ascertain or trust the user that all the wheels will completely be of same specification. For instance the user may use different tyres, tyre aging may also vary, etc. For similar reason the BCU consist of specialized BCUs since all operate on their respective wheels. In accordance with the new ISO 26262 standard for automotive safety, the analysis could be performed at the software architecture design phase of the "product development at software level" [10]. The analysis is thus presented in Table 2.

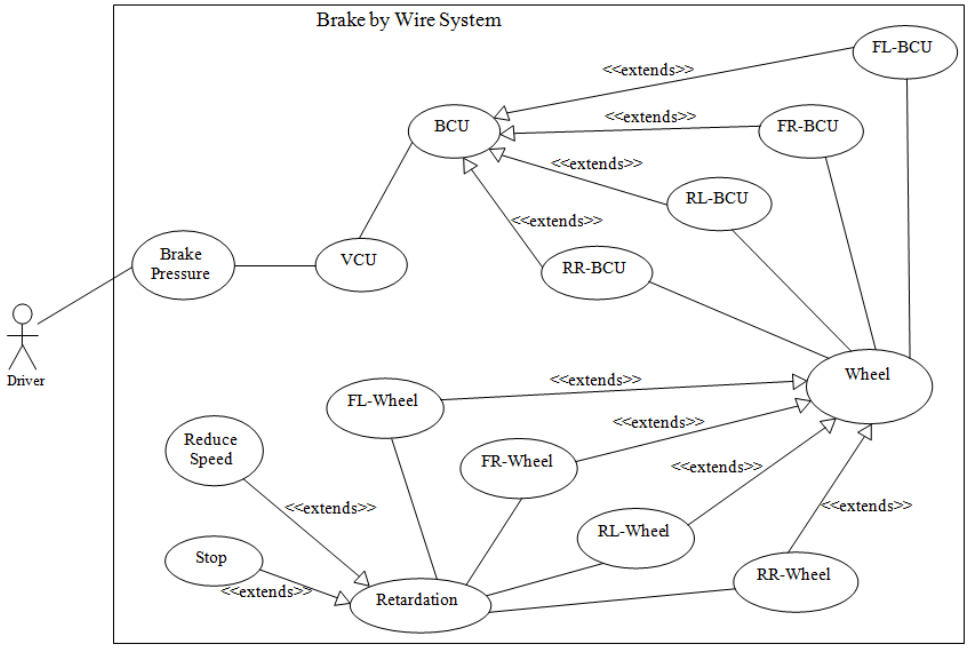

Figure 4. Use Case Diagram of Brake by Wire System

Table 2 – Software FMEA of BBW

System Failure Mode: *Braking Failure*				
Entity	**Potential Cause**	**System Effect**	**Mitigation**	**Remark**
Driver	Brake not applied – i.e. omission of input	No retardation	Provision for a function that can detect an object that is in line of motion will be helpful. The detection should be relative to the speed of the vehicle. The detection can be done by a smart sensor. A warning function should be called to alert the driver	This is an example of a scenario where certain safety measures are beyond system control, and the required intervention is in the hands of the user whom the system has no control over [11]. For instance the driver decides whether or not to apply brake
Brake Pressure	Low pressure input	Late retardation	Late retardation may result into accident. Similar to the above, the possibility of including a smart sensor that can spot object in the line of motion and compensate required pressure to retard the vehicle appropriately will be helpful	This is a case where software failure analysis informs the possibility to modifying and evaluating the hardware design according to the design objectives. Software and hardware failure analysis team should therefore communicate their results of analysis to one another. The software implementation will therefore need a function that can detect low pressure, speed of vehicle, object distance, and another to generate the required brake

				pressure. Hence this is a clear fact that analysis at the use case level will help inform the next design level such as class diagram
	High pressure input	Early retardation	Early retardation may as well lead to accident, for instance a moving vehicle behind may brake late and run into the vehicle in its front. A rear smart sensor will be helpful to detecting the distance of the object behind	This is similar to the above, however a function to detect the high pressure is required. Safety design measures then begins to be more complicated as in both low and high pressure values the detection of object behind and in front of the current vehicle is helpful to be used as an input to compensate required pressure
	Omission of input value	No retardation	Any function developed to apply the brake on wheels should check and ensure that the brake pressure is not null or zero. The provision for emergency braking may be considered. When a null or zero value is detected the emergency braking function would be evoked.	The provision of emergency or backup brake system is the case in the *wet brake by wire* system.
VCU	Invalid (out of range) input	High or slow retardation	The range of the brake pressure value should be	This will entail the creation of a function that

			checked before outputting to the BCU. When the value is invalid and on detecting an object in the line of motion, a configured braking is applied. In the absence of a detected object, an emergency brake should be activated	checks the validity of brake pressure that arrives at the VCU. The invalid input could be triggered by external interference
	Omission of input	No retardation	When the brake pressure to the VCU is either null or zero then the mitigation is same as above	Omission of input may occur when a function in the VCU is called without a brake pressure data. This could also be triggered as a result of external interference
BCU	Invalid input	High or slow retardation	Same as for the VCU	The output of the VCU becomes the input of the BCU. If the output of the VCU is mitigated as above, one could say that there is no need to implement checks for the BCU's input. However, it cannot be guaranteed that the value will arrive at the BCU in a valid form. It is therefore essential that the BCU implements its own function to validating the input.
	Omission of input	No retardation	Also same as for the VCU	

FL-BCU	Reuse(BCU)	Reuse(BCU)	Reuse(BCU)	As mentioned earlier these are specialized form of the BCU. The *Reuse()* is here used to refer to adopting the failure modes and mitigations for the use case appearing in the parenthesis - "()".
FR-BCU	Reuse(BCU)	Reuse(BCU)	Reuse(BCU)	
RL-BCU	Reuse(BCU)	Reuse(BCU)	Reuse(BCU)	
RR-BCU	Reuse(BCU)	Reuse(BCU)	Reuse(BCU)	
Wheel	Non specified size	Unequal application of brake on all wheels, hence the effect is poor retardation	Before generating the right amount of brake pressure by the BCU, it is thus necessary that the BCU inquires of the properties of the wheel concerned. Such properties could be size, pressure, wear, etc.	This will entail having a function that obtains such properties and used in determining the right brake pressure to apply on the wheel concerned
	Worn out tyre			
	Incorrect tyre pressure			
FL- Wheel	Reuse(Wheel)	Reuse(Wheel)	Reuse(Wheel)	Being specialized forms of the Wheel, all these use cases adopt the failure modes and mitigations of the Wheel
FR- Wheel	Reuse(Wheel)	Reuse(Wheel)	Reuse(Wheel)	
RL- Wheel	Reuse(Wheel)	Reuse(Wheel)	Reuse(Wheel)	
RR- Wheel	Reuse(Wheel)	Reuse(Wheel)	Reuse(Wheel)	
Retardation	Number of wheels that received brake request are less than 4 or none at all	Poor retardation	Count of wheels that receive brake request should be made and this should equal 4. Any less, then communication to the VCU should be made to re-issue a configured braking to compensate the deficiency	Such counter should be re-initialized after completion of each brake to avoid overflow i.e. for a vehicle with 4 wheels, the count should not exceed 4

Reduce Speed	Failure in the brake actuator on wheels	Poor or no retardation	A provision for a function that can read the health of the brake actuators for each wheel will be beneficial. When the health is poor, the driver may be alerted for maintenance.	This may involve the use of a sensor and a function that can read the sensory data.
Stop	Failure in the brake actuator on wheels	Poor or no retardation	Same as above	Same as above

As mentioned earlier, the outlined mitigating measures could be used to further evaluate the use case design i.e. to see whether extra use case(s) is or are needed to improve the system. More interestingly, the development of the next level of architecture such as class diagram can incorporate the mitigating measures. This could be in the form of class attributes and methods (member functions) that will implement the prevention of the defined failure mode. The demonstration of this is however out of the scope of this paper and is left for further work.

The analysis in Table 2 has shown that hardware design modifications could be informed by the result of software failure analysis, for instance the introduction of sensors. The analysis has also shown that functions required to prevent failure can be identified at the user level architecture. In a further design, these identified functions could be included in the class diagram of the system.

In a typical system development environment, teams of engineers may be given different failure modes to work on where firstly each engineer within a team will work independently. Secondly a collective review and collation in each team is performed and thirdly the work of all teams is reviewed.

6. Conclusions

Typically, failure modes and effects analysis (FMEA) is used in addressing the failures and mitigating interventions for hardware systems. It is unclear how FMEA could be used to analyze software systems. This can be attributed to the difference in the way software and hardware fail and also since FMEA was developed to analyzing hardware failure. This paper has investigated the possibility of using FMEA in the failure analysis of software systems by considering as a case study the brake by wire (BBW) system which is a recent design consideration in the automotive industry. In both software and hardware systems, failure analysis should begin from the infancy stage of design through to completion. Therefore this paper demonstrated the use of FMEA in analyzing software system at the top level software architecture - use case diagram. The paper then establishes and proposes a failure analysis model for software architecture. Also, the use case diagram of the BBW was drawn from where the FMEA analysis of the BBW was performed. The analysis shows that the use of FMEA to analyzing software systems is possible and that results of the analysis such as

mitigating interventions would reveal further design considerations to improve dependability of software systems.

References

[1] N. Storey, Safety-Critical Computer Systems, Addison Wesley Longman , London **(1996)**

[2] P. Haapanen, and A. Helminen, "Failure mode and Effects Analysis of Software-based Automation Systems", STUK-YTO-TR 190, Helsinki, 2002, Available: http://www.fmeainfocentre.com/handbooks/softwarefmea.pdf , Accessed: **(2011)** July 3.

[3] N. Leveson, "A New Accident Model for Engineering Safer Systems", Safety Science **(2004)** Vol. 42, No. 4, pp. 237-270

[4] G. Cassanelli, G. Mura, F. Fantini, M. Vanzi, and B. Plano, "Failure Analysis-assisted FMEA", Microelectronics and Reliability, Vol. 46, Issues 9-11 **(2006)** pp. 1795-1799

[5] V. Ebrahimipour, K. Rezaie, and S. Shokravi, "An Ontology Approach to Support FMEA Studies", Expert Systems with Applications, Vol. 37, Issue 1 **(2010)** pp. 671-677

[6] R.S. Pressman, Software Engineering: A Practitioner's Approach, 5th Ed, McGraw-Hill Series in Computer Science, New York **(2001)**

[7] P. Sinha, "Architectural Design and Reliability Analysis of a Fail-Operational Brake-by-Wire System from ISO 26262 Perspectives", Reliability Engineering & System Safety, Vol. 96, Issue 10 **(2011)** pp. 1349-1359

[8] H.T. Dorissen, K. Dürkopp, "Mechatronics and Drive-by-Wire Systems Advanced Non-contacting Position Sensors", Control Engineering Practice, Vol. 11, Issue 2 **(2003)** pp. 191-197

[9] C. Wilwert, N. Navet, Y.-Q. Song, and F. Simonot-Lion, "Design of automotive X-by-Wire systems," in The Industrial Communication Technology Handbook, R. Zurawski, Ed. Boca Raton, FL: CRC **(2004)**

[10] IDRA, ISO 26262 The merging Automotive Safety Standard [online], Available: http://www.siliconindia.com/events/siliconindia_events/Softec_Conf_Pune/Shrikant.pdf, Accessed: **(2011)** October 21, SiliconIndia Website.

[11] N.G. Leveson, System Safety Engineering: Back to the Future, Aeronautics and Astronautics, Massachusetts Institute of Technology **(2002)**

Author

Shawulu Hunira Nggada obtained a B. Tech (Hons) in Computer Science from Abubakar Tafawa Balewa University, Nigeria and MSc in Software Engineering from the University of Bradford, UK. He is currently a PhD student at the University of Hull, UK and working on multi-objective optimisation of system maintenance. He is also a Charted IT Professional with the British Computer Society. Some of his research interests are system dependability, software engineering, green computing and programming languages.