Waves of Faults in Embedded System and Way out through Fault Tolerance

Jigna B. Prajapati
Research Scholar, J.J.T.University

Dr.N.K.Modi
Prof. & Head,
S.V.Institute of Computer Studies
Kadi, Gujarat, India

Savan K.Patel
Asst.prof, Acharya Motibhai Patel
Institute of Computer Studies,
Ganpat University, Kherva, Gujarat, India

ABSTRACT

Software has rapidly become an important and indispensable element in many aspects of our daily lives. If such element is not running as on our need, we have to go through the problems about it. In initially, the paper focus on the different types of faults, their impact and fault classification. Faults are subdivided into different activities such as fault prediction, fault detection, fault prevention, fault correction etc. Here we study the faults in context boiler system. The concern thing is Faults classification as external, location, duration, and effect, permanent, temporary and may more. Any fault arise within system can be avoid, prevent or removed. Then we propose the different fault tolerance techniques to deal with different faults.

General Terms

Software Faults t**o**lerance

Keywords

Software Faults, Faults Comparison, Faults tolerance**,** Design diversity

1. INTRODUCTION

In recent years, the production of reliable software for control systems in real time has become a major industry interest. We hope that these systems function reliably, even under extremely harsh conditions. However, no matter how well you test, debug and verify modularize, design errors still plague our software.

The term error often is used in addition to the terms fault and failure. Often, errors are defined to be the result of faults, leading to failures [1]. Informally, errors seem to be a passive concept associated with incorrect values in the system state. However, it is extremely difficult to develop unambiguous criteria for differentiating between faults and errors. Many researchers refer to value faults, which are also clearly erroneous values. The connection between error and failure is even more difficult to describe.

We substitute the term fault for the common uses of the term error. Generally, references to the term "error" in the literature can be fitted to the context of this document by substituting the term "fault."

Fault can be of different types as system boundaries, dormant, phenomenological, Permanent - Temporary, physical faults, Intentional, Accidental, Interaction faults [2] and many more. When such faults arise within our system, that should be avoided, prevented or removed which can performed with the use of fault tolerance.

Fault-tolerance is the property that enables a system to continue operating properly in the event of the failure of (or one or more faults within) some of its components. If its operating quality decreases at all, the decrease is proportional to the severity of the failure, as compared to a naively-designed system in which even a small failure can cause total breakdown [3]. Fault-tolerance is particularly sought-after in high-availability or life-critical systems [3, 4]. In any applications, operational reliability is of paramount importance. Therefore, to achieve ultra-reliability in industrial computing, it is necessary to adopt the strategy of defensive programming based on redundancy. This is referred to as fault-tolerant software.

2. FAULT CLASSIFICATION & IT'S IMPACT

Faults may be classified based on Locality:-atomic component, composite component, system, operator, and environment where faults reside in some specific location, the combination of more than one component, faults arise from any environmental causes, or any user-operators [2] Cause:- design, damage where problems arise by problematic designing of system, application or software. Duration:-transient, persistent where faults occurred either temporary or permanently. Effect: - on System State crash, amnesia, partial amnesia, etc. [5,6]

Faults can be classified according to their phase of specification fault, creation(design fault), implementation fault or occurrence, system boundaries:-internal, external where functionalities provided up to minimum and maximum, domain hardware or software, phenomenological cause, intent, and persistence. The discussion below is focused on software fault classification based on their recovery strategies [2, 5].

Physical faults: Permanent, internal, physical faults. This class concerns those faults that have their origin within hardware components and are continuously active Temporary, internal, physical faults (intermittent faults) [2, 6].

The impact of any faults is to take system in non working state. The fault can lead the either system failure or component failure. If occurred fault within system is not breaking down the working state but it may lead another fault. Locality faults may be within one component which cause failure to another component and become composite component failure. Cause failure as design faults are remain forever in system where we don't have chance to prevent or correct such faults later on. Faults on duration are sometime permanent and sometime temporary. Permanents faults need to redesign or another design of the same piece of software. Temporary fault cause temporary failure of temporary improper outcome which can be working property on usual state. User faults, operator faults, documentation faults are accidental faults which may arise or may not. If such faults not arise then system will work properly.

3. FAULTS IN EMBEDDED SYSTEM

To help understand these definitions, consider the example of Traffic system. Sometime after developing such system, we can point out that a precedent for using this as an example exists here comparing practices in traffic system design with practices in software design.

When designing the Software to control boiler the designer must consider details regarding requirements, and the environment in which the Boiler System would be operated. Suppose system allowed 180 f. How the fault that led to the failure? There are lots of possible answers to this:

To help understand these definitions, consider the example of Traffic system. Sometime after developing such system, we can point out that a precedent for using this as an example exists here comparing practices in traffic system design with practices in software design.

When designing the Software to control boiler the designer must consider details regarding requirements, and the environment in which the Boiler System would be operated. Suppose system allowed 180 f. How the fault that led to the failure? There are lots of possible answers to this:

A. The designer of the System did not allow for appropriate temperature setting. This could be:
    a) A specification fault if the XYZ department did not anticipate that more than 180 f would required need to use the boiler, or
    b) A design fault if the specification called for it being able to keep 180 f .
    c) An implementation fault if we didn't correctly follow the design.

B. The boiler user ignored a "Temperature Limit" sign. This would be a user fault.

C. A worker for the XYZ department posted an erroneous "Temperature Limit" sign. This would be an operator fault.

D. The people preparing the documentation for the boiler system mistakenly indicated that the boiler would support 280 f, when in fact it was only designed to support 180 f. The XYZ department erected a 180f "Temperature Limit" sign. This would be a documentation fault, followed by an operator fault.

E. By any natural effect, if system would damage or crashed that would be environmental faults.

As same, consider the same boiler with a improper temperature. There is no failure involved if the boiler continues to carry the temperature requested of it in spite of this fault. It may be the result of normal wear and tear. However, a thorough analyzing of the boiler system might discover that the temperature in the system a faulty strut, From the point of view of the boiler system analyzer, the strut would have failed. This component failure is an internal fault.

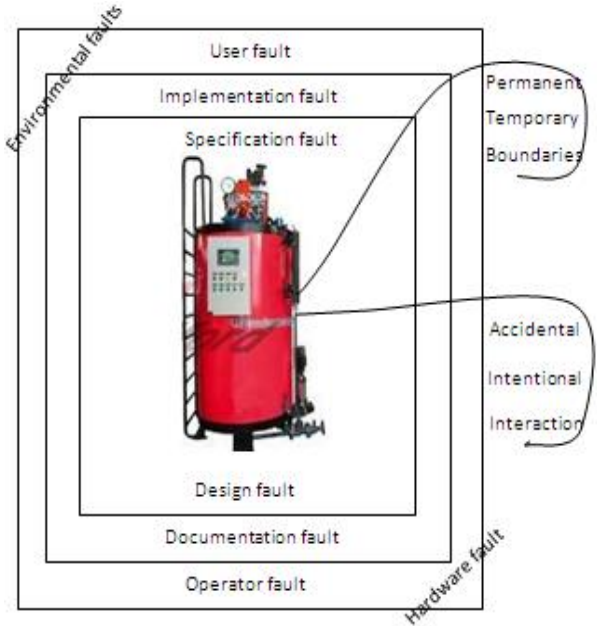

Fig: 1 Fault classification on Boiler System

Scenarios like this can be generated ad infinitum. Note that a fault does not lead to a failure unless the result is observable by the user, and leads to the boiler system becoming unable to deliver its specified service. This means that one person's fault is another person's failure. For instance, in example 4 above, from the point of view of the department the erroneous documentation was a fault that led to an operator failure. From the point of view of the user of the boiler system the erroneous documentation was a documentation fault that led to an operator fault which led to a boiler system failure. Consider a computer system running a program to control the temperature of a boiler by calculating the firing rate of the burner for the boiler. If a bit in memory becomes stuck at one that is a fault. If the memory fault effects the operation of the program in such a way that the computer system outputs cause the boiler temperature to rise out of the normal zone, that is a computer system failure and a fault in the overall boiler system. If there is a gauge showing the temperature of the boiler, and its needle moves into the "yellow" zone (abnormal, but acceptable), that is a symptom of the system fault. On the other hand, if the boiler explodes because of the faulty firing calculation, that is a (catastrophic) system failure.

4. FAULT TOLERANCE TECHNIQUES

Fault prevention aims at preventing the occurrence or introduction of faults. Techniques in this category include, e.g., quality assurance and design methodologies; Fault removal aim to remove faults after the development stage is completed. This is done by exhaustive and rigorous testing of the final product [7].

Fault avoidance/prevention includes design methodologies which avoid the faults which may not have fault solution [7, 8].

Fault tolerance makes the assumption that the system has unavoidable and undetectable faults and aims to make provisions for the system to operate correctly even in the presence of faults [7, 8, 9].

These techniques are divided into two groups as Single version and multi-version software techniques [10]. Single version techniques focus on improving the fault tolerance of a single piece of software by adding mechanisms into the design targeting the detection, containment, and handling of errors caused by the activation of design faults. Single version techniques are Error detection, Exception handling, Data diversity, Process pair, etc. Multi-version fault tolerance techniques use multiple versions (or variants) of a piece of software in a structured way to ensure that design faults in one version do not cause system failures[10,11,12]. A characteristic of the software fault tolerance techniques is that they can, in principle, be applied at any level in a software system: procedure, process, full application program, or the whole system including the operating system Also, the techniques can be applied selectively to those components deemed most like to have design faults due to their complexity. Multi-version fault tolerance techniques are as Recover block, N-version programming [11, 13].

5. DISCUSSION

A system fails because of incorrect specification, incorrect design, design flaws, poor testing, undetected fault, environment, substandard implementation, aging component, operator errors or combination of these causes. Though programming bugs is considered to be an important reason of the most system failures at present but the recent studies suggest that soft errors are increasingly responsible for system downtime [2]. Computing system is becoming more complex and is getting optimized for performance and price but not for availability. This makes soft errors an even more common case. Using denser, smaller and lower voltage transistors has the potential threats to be more susceptible to such increased transient errors. Soft errors are the errors, which occur because of the unintended transitions of logic state in a circuit typically caused by external source of ionizing radiations.

To deal with errors in fault tolerance system classified as roll-forward and roll-back. Roll forward means to take the system to some specified location to resume the errors. Rollback means to take system to some earlier version [2, 5, and 6]. Here, we analyzed different faults in the mentioned embedded system which can be managed by the fault tolerance techniques.

We start with specification faults, (A.a) which can be managed by the rechecking design specification. Design faults (A.b) can be managed by Design diversy (NVP or RcB). NVP use multiple versions of the same requirements, where we can divert to another version on chosen design. In NVP if one of the design fail, at least one alternate version will work [14]. Implementation faults (A.c) which need to check properly before getting in use. We can use Verification, error detection and check point techniques. User faults (B) are of different types but generally it will wrong range of data which can be solved by input limit checking, exception handling. If user gives improper input so it would be handled by exception or checking the limit. Operator faults (C) which may work n improper instructions which can be managed by n-self checking or data diversy[12,14]. If any hardware or part of hardware would fail because of any reason we can apply hardware fault tolerance. Hardware fault tolerance gives alternate component for failed component [12].

6. CONCLUSION

As we discussed faults reside and arise from requirements of the system to implementation of an embedded system. Different types of Faults found during the process of system usage. Analysts would think and act on that which will consider as design fault later on. Such faults will recover by another design of that part of the system known as recovery block or N-version programming. Any wrong data will feed up (user faults) in system will handle by exceptional handling. If more resources are required and any faults occurred due to such reasons then we can apply Processor pair. So, in this way we can handle different faults for embedded System with the use of above mentioned fault tolerance techniques. Thus we can improve software reliability. Software fault tolerance techniques provide protection against errors in translating the requirements and algorithms into a programming language, but do not provide explicit protection against errors in specifying the requirements. Software fault tolerance techniques have been used in the aerospace, nuclear power, healthcare, telecommunications and ground transportation industries, among others.

7. REFERENCES

[1] Software faults prediction using multiple classifiers, Computer Research and Development (ICCRD), 2011 3rd International Conference on, 11-13 March 2011,504-510p

[2] Jean-Claude Laprie. Dependability: Basic Concepts and Terminology in English, French, German, Italian and Japanese, volume 5 of Dependable Computing and Fault Tolerance. Springer Verlag, Wien, 1992,16-17p.

[3] Randell B. "System Structure for Software Fault Tolerance" IEEE Transactions on Software Engineering 1975 SE-1(2) 220–232p.

[4] Laprie J. C. et al. Hardware and software fault tolerance: definition and analysis of architectural solutions in Proceedings of 17th International Symposium on Fault-Tolerant Computing, Pittsburgh. 1987. 116-121p.

[5] Jean-Claude Laprie, Dependability—its attributes, impairments and means. In B. Randell, J.-C. Laprie, H. Kopetz, and B. Littlewood, editors, Predictably Dependable Computing Systems, ESPRIT Basic Research Series, pages 3–18.Springer Verlag, Berlin, 1995.

[6] Jean-Claude Laprie. Dependability of computer systems: from concepts to limits.
In Proc. of the IFIP International Workshop on Dependable Computing and Its Applications (DCIA98), Johannesburg, South Africa, 1998.

[7] Malicious- and Accidental-Fault Tolerance for Internet Applications Conceptual Model and Architecture David Powell and Robert Stroud, 23-24p. Available: http://www.Joomla.org/core-features.html

[8] CSE 598D: Software Fault Tolerance Instructor: Mahmut Kandemir, 23-24p

[9] Intrusion-Tolerant Architectures:Concepts and Design byPaulo Esteves Vessimo, Nuno Ferreira Neves, Miguel Pupo Correia

[10] P. A. Lee and T. Anderson. Fault Tolerance: Principles and Practice Second Edition, Springer-Verlag. 1990.

[11] A SURVEY OF SOFTWARE FAULT TOLERANCE TECHNIQUES byZaipeng Xie, Hongyu Sun and Kewal at 0.

[12] Software Fault Tolerance: An Evaluation by Anderson, T., Barrett, P.A., Halliwell, D.N. Moulding, M.R. In Software Engineering, IEEE Transactions on,2006,1502 – 1510p

[13] Software Fault Tolerance Techniques & Implementation by Laura L Pullan, pages 68,72-76

[14] Avizienis A. "The N-Version Approach to Fault-Tolerant Software" IEEE Transactions on Software Engineering 1985. SE-11(12) 1491-1501p.