ISSN : 2347 - 8446 (Online)
ISSN : 2347 - 9817 (Print)
Vol. 2, Issue 4 (Oct. - Dec. 2014)
International Journal of Advanced Research in Computer Science & Technology (IJARCST 2014)

# A Survey of Software Fault Tolerance, Reliability and Safety

[I]Anuj Verma, [II]Disha Guleria, [III]Karuna Lakhanpal

**Abstract**

*This survey reviews the state of system reliability, safety and fault tolerance of the system. The system will provide the desired level of reliable service. The proposed work where it discusses why some assumptions underlying hardware fault tolerance do not hold for software. Presently our inability is to produce error- free software, software fault tolerance is and will continue to be an important consideration in software systems. Software safety applications for some is more important than reliability and fault tolerance techniques used. The fault tolerance for system dependability must be validated to ensure that its redundancy has been implement correctly and system will provide the desired level of reliable service.*

**Keywords**

*Fault tolerance, Fault tolerance process, Software Fault Tolerance, Reliability, Software Reliability, Safety, Software Safety.*

## I. Introduction

The software has seen many changes since its inception. It has evolved over the period of time against all circumstances. The major problem of software industry is its inability to develop bug free software. If software developers are asked to certify that the developed software is bug free, no software would have ever been released. It defines that the aim is the production of quality software that is delivered on time, within budget and that satisfies its requirements. People makes errors and a requirements error may be magnified during design, and amplified still more during coding. If it could not be detected prior to release, it may have serious implications in the field. An error may lead to one or more faults. It is more precise to say that a fault is the representation of an error, where representation is the mode of expression, such as narrative text, data flow diagrams, ER diagrams, source code etc. Defect is good synonym for fault. If fault is in source code, we call it a bug.

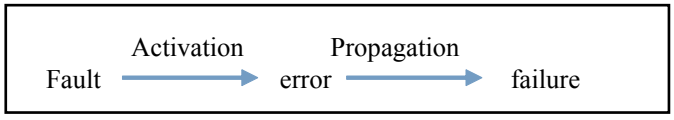


Fig.1: The fundamental chain of dependability threats leading to a failure

The need to control software fault is one of the most rising challenges facing software industries today. Fault is the defect in the program that, when executed under particular conditions, causes a failure. A fault can be the source of more than one failure. It is obvious that fault tolerance must be a key consideration in the early stage of software development. There exist different mechanisms for software fault tolerance, among which:

- Recovery blocks
- N-version software
- Self-checking software

### A. Fault Tolerance

Fault Tolerance features basic allow the computer keep executing with the presence of defects. these systems are usually classified as either highly reliable or highly available. It 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. Fault tolerance is particularly sought after in high-availability or life-critical systems.The most important requirement of design in a fault tolerant computer system is making sure it actually meets its requirements for reliability. This is done by using various failure models to simulate various failures, and analyzing how well the system reacts.

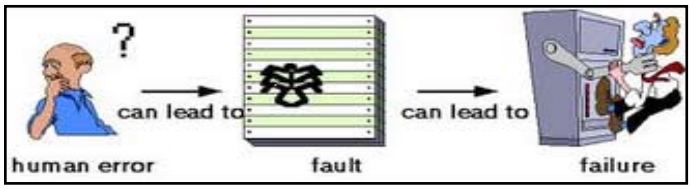


Fig. 2: Fault occur

A process is said to be fault tolerant if the system provides proper service despite the failure of the process. Fault tolerance is the property that enables a system to continue operating properly and it must continue to operate without interruption during the repair process.

### B. Reliability

Reliability is often misunderstood in the software field since software does not break in the physical sense. Software reliability is defined as the probability that a software system fulfills its assigned task in a given environment for a predefined number of input cases, assuming that the hardware and the inputs are free of error. It is the probability of a failure free operation of a program for a specified time in a specified environment. This tends to change continually test periods. It is generally measured by the number of failures per 't' hours of operation, mean time to failure or probability of failure free operation in a specified time under specified environment.

The reliability characteristics relating to capability of software to maintain its level of performance under stated conditions for a stated period of time are:

- Maturity- Attributes of software that bear on the frequency of failure by faults in the software.
- Fault Tolerance- Ability to maintain a specified level of performance in cases of software faults or unexpected inputs.
- Recover ability- Capability and effort needed to reestablish level of performance and recover affected data after possible failure.

### C. Software System Safety

Software system safety is an element of the total safety and software development program, cannot be allowed to function independently of the total effort, here the both simple and highly integrated multiple systems are experiencing an extraordinary growth in the use of computers and software to monitor and

ISSN : 2347 - 8446 (Online)
ISSN : 2347 - 9817 (Print)

control safety-critical subsystems or functions. Software Specification error, design flaw, or the lack of generic safety-critical requirements can contribute to or cause a system failure or erroneous human decision. Here, to achieve an acceptable level of safety for software used in critical applications, software system safety engineering must be given primary emphasis early in the requirements definition and system conceptual design process. Safety-critical software must then receive continuous management emphasis and engineering analysis throughout the development and operational lifecycles of the system. It is directly related to the more critical design aspects and safety attributes in software and system functionality, whereas software quality attributes are inherently different and require standard scrutiny and development rigor. Hazard analysis of software safety required for more complex systems where software is controlling critical functions generally are in the following sequential categories and are conducted in phases as part of the system safety or safety engineering process: software safety requirements analysis; the design analyses of software safety (top level, detailed design and code level); and software safety test analysis, and software safety change analysis. If these "functional" analyses of software safety are completed the software engineering team will know where to place safety emphasis and what functional threads, functional paths, domains and boundaries to focus on when designing in software safety attributes to ensure correct functionality and to detect malfunctions, failures, faults and to implement a host of mitigation strategies to control hazards. Security and various protection of software technologies are similar to software safety attributes in the design to mitigate various types of threats vulnerability and risks.

Software has been built into more and more products and systems over the years and has taken on more and more of the functionality of those systems. The question is: how dependable is the functionality provided by software? The traditional approach to verification of functionality - try it out and see if it works - is of limited value in the case of software which can be much more complex than hardware. Software safety has evolved to be a parallel effort to the development of the software itself. The System Safety engineer is involved in each step of the software development process identifying which functions are critical to the safe functioning of the greater system and tracing those functions down into the software modules which support them.

## II. Conclusion

Here size and complexity of software systems were increasing which makes it hard to prevent or remove all possible faults. The faults that should remain in the system can eventually lead to a system failure. The techniques of fault tolerance are introduced for enabling systems to recover and continue operation when they are subject to faults. There are many fault tolerance techniques are available but incorporating them in a system which were not always trivial. In this thesis, here introducing the application methods and tools for the fault tolerance techniques to increase the reliability and safety of software systems.

## III. Acknowledgment

Thanks to my Guide and family member who always support, help and guide me during my dissertation. Special thanks to my father who always support my innovative ideas.

## References

[1]. *D. E. Bernard, E. B. Gamble, N. F. Rouquette, B. Smith, Y. W. Tung, N. Muscettola, G. A. Dorias, B. Kanefsky, J. Kurien, W. Millar, P. Nayal, K. Rajan, and W. Taylor. Remote Agent Experiment DS1 Technology Validation Report. Ames Research Center and JPL, 2000.*

[2]. *M. C. Boden. Benefits and Risks of Knowledge-Based Systems, 1989. (ISBN 0-19854-743-9).*

[3]. *J. J. Borrelly, E. Coste-Manière, B. Espiau, K. Kapellos, R. Pissard- Gibollet, D. Simon, and N. Turro. The ORCCAD Architecture. The International Journal of Robotics Research, 17(4):338–359, April 1998.*

[4]. *J. Carlson and R. R. Murphy. Reliability Analysis of Mobile Robots. In Proceedings of the 2003 IEEE International Conference on Robotics & Automation, pages 274–281, Taipei, Taiwan, September 14-19, 2003.*

[5]. *M. Daran. Modelling Erroneous Software Behavior and Application to Testset Validation by Fault Injection. PhD thesis, Institut National Polytechnique de Toulouse, 1996. LAAS Report No. 96497 (in French).*

[6]. *O. Despouys. An Integrated Architecture for Planning and Execution Control in Dynamic Environments. PhD thesis, Institut National Polytechnique de Toulouse, 2000. LAAS Report No. 00541 (in French).*

[7]. *C. Dousson. Evolution Tracking and Chronical Recognition. PhD thesis, Université Paul Sabatier, 1994. LAAS Report No. 94394 (in French).*

[8]. *M. S. Feather and B. Smith. Automatic Generation of Test Oracles - From Pilot Studies to Application. In Proceedings of the 14th IEEE Automated Software Engineering Conference (ASE-99), Cocoa Beach, Florida, October 12-15, 1999.*

[9]. *B. Tekinerdogan. Asaam: Aspectual software architecture analysis method. In Proceedings of the 4th Working IEEE/IFIP Conference on Software Architecture (WICSA), pages 5–14, Oslo, Norway, 2004.*

[10]. *B. Tekinerdogan, H. Sozer, and M. Aksit. Software architecture reliability analysis using failure scenarios. In Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture (WICSA), pages 203–204, Pittsburgh, PA, USA, 2005.*

[11]. *B. Tekinerdogan, H. Sozer, and M. Aksit. Software architecture reliability analysis using failure scenarios. Journal of Systems and Software, 81(4):558–575, 2008.*