# System Documentation

# ABSTRACT

It is known that in today’s life computer applications plays an essential role in modern society, including in medical devices, e-commerce, aircraft control system and many other applications. Failures of such systems may cause very severe consequences. Due to limitations of human capability, it is impossible to design a system which is free from errors. Although software developers try they best to ensure that the system is clean before it goes in to use, but the analysis show that many application failures in today’s experience are caused by software errors which inherent from design faults during software development. Therefore, the need for software fault tolerance becomes increasingly important. To ensure the development of highly dependable systems, different author have suggested various techniques for building software fault tolerance. This paper will review the three methods for software fault tolerance which are recovery blocks, N-version programming, and self-checking software and suggesting the best method among them.

**KEYWORDS**: software fault tolerance, N-version programming, recovery blocks, self-checking software.

**METHODOLOGY**: The methodology used to conduct this research is a Literature Review. Different survey and reviews have been done through some published papers from journals and international conferences that involve the research about designing a software fault tolerance.

# INTRODUCTION

Fault tolerance is the ability of a system to cope with internal error and continue to perform its task correctly. The ultimate goal of fault tolerance is to boost the dependability of a system [2,9]. Dependability is the ability of a system to deliver its required services to its users. Reliability, availability and safety are desirable features for any computing system, and are primary attributes of dependable system. As computer systems become more applicable in our today’s life, the dependability of these systems becomes a critical issue. For example, in airplanes, e-commerce system, heart pace-makers or other safety critical applications, a system failure may cause a large damage into people’s life. The failures of those system may be caused by a number of reasons but most of the reported system failure in today’s experience are caused by software fault. For example, in the research done by [2] show that 60-90% of current computer errors are from software faults. Although all designers wish to develop a system which operate without a failure but the history shows that such a goal is not achievable because it is practically impossible to develop a perfect system. No matter how carefully those systems are tested, debugged, and verified, design bugs will still remain in the system and hence later causes a system failure. In order to achieve the goals of designing a software which will be free from failure, a certain measure should be taken into considerations to handle those faults which remain in the system after its development. Therefore, the idea toward designing software fault tolerance became very crucial and many researchers have proposed a number of approaches for designing software fault tolerance. According to [2,3,6,9] some of approaches for tolerating software fault are include recovery block, N-version programming (NVP), Consensus recovery block(CRB), Distributed Recovery Block (DRB), N-self checking version programming(NSCP), Roll-Forward Check Pointing Scheme (RFCS), Extended Distributed Recovery Block (EDRB). However, this paper will not propose the new technique for designing software fault tolerance but instead it will review the existing techniques such as N-version Programming, Recovery Block and Self-checking software and evaluating the effectiveness of those techniques in ensuring fault tolerance in software system.

# SOFTWARE FAULT TOLERANCE TECHNIQUES

Software fault tolerance is the use of techniques that enable the system to deliver a service at an acceptable level of performance and safety after a design fault becomes active. Software fault tolerance techniques are designed to allow a system to tolerate software faults that remain in

the system after its development. These techniques are employed during the development, of the software. When a fault occurs, these techniques provide mechanisms to the software system to prevent system failure from occurring. According to [9], software fault tolerance approaches are categorized in to two group which are design diversity (multiple version) and single design.

**Design Diversity Software Fault Tolerance Approach**

Design Diversity or multiple version software fault tolerance is based on the use of two or more versions of a piece of software which are executed either in sequence or in parallel. In this approach components of a system are built through independent designs but deliver the same service. The fundamental assumption of design diversity is that components build differently will fail independently. Thus, if one of the redundant version fails, at least one of the others will provide an acceptable output. Examples of such techniques include recovery blocks, N-version programming

**Single-Design Software Fault Tolerance Approach**

Single version techniques focus on improving the Fault Tolerance of a single piece of software by adding mechanism into the design, targeting the detection, containment, and handling of errors caused by the design faults. [3] The goal of single version techniques is to determine the fault occurrence in a system. Among others, single-version software fault tolerance techniques include considerations on program structure and actions, error detection, exception handling, checkpoint and restart, process pairs, and data diversity.

## N-VERSION PROGRAMMING

N-version programming (NVP), also known as multi-version programming is a fault tolerance technique where multiples version of software system is developed, these versions are independently generated from the same initial specifications. With N-Version Programming, NVP, independent development teams use the same specification to generate multiple implementations. During development the design teams are kept separate and do not share their designs nor do they discuss the specification’s meaning with each other. In this technique the design teams use different algorithms and different programming languages to produce multiple versions. These versions are either run sequentially in a single processor or parallel in a loosely coupled processor and then submits its answer to voter or decider or decision mechanism (DM) which determines the correct answer and returns the final result if it exist. The voter uses the result of majority to make decision of the final result. According to [5, 9], some of the generalized selection algorithms are Formalized majority voter, Generalized median voter, Formalized plurality voter and Weighted averaging techniques. Other voting techniques are based on Neural network and Genetic algorithm techniques. They are implemented such that their performance is related to the application and the particular characteristic of the software versions. It is hoped that by performing the N designs independently it will overcome the design faults present in most software, since the same mistakes will not be made in all the modules The voting module will be able to detect a fault because the same fault is not expected to occur in all the modules [5].

The General syntax of NVP:

**run version 1, version 2, ..., version n**

**if (Voter (Result 1, Result 2, ..., Result n))**

**return Result**

**else**

**failure exception**

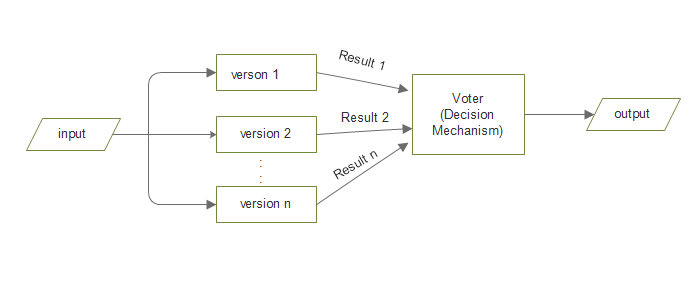
****

Figure : N-Version Programming Structure

The basic elements of the N-version programming approach are:

* ***The initial specification***- this is the functionality that a software system is desired to provide
* ***N software versions***- software modules which all are independently generated from the initial specification;
* ***A decision mechanism(voter****)* - a mechanism which take in the results from each version as input and make decisions on the final result.
* ***A supervisory program***- this is a software structure used to drive the N versions and the decision mechanism.

From the figure-1 the same input data is entered in all versions (i.e. version 1, version 2, .., version n ) of the software and each version execute that input data independently and the results obtained is sent to the voter which determine the final output by comparing the results from all the versions based on majority vote.

## RECOVERY BLOCKS

Recovery blocks is a software fault-tolerance technique which combines the basics of checkpoint and recover schemes [2,7,9,]. N versions of a software block are provided and a set of acceptance tests is used. One version of the block is designated as the primary and the rest are alternate versions. The Checkpoints are created before a version executes and are needed to recover the state of the software system after a version fails to produce a desired operation. The Acceptance Test (AT) need not be an output-only test and can be implemented by various embedded checks to increase the effectiveness of the error detection [9]. To perform a recovery block operation, the primary versions is executed and an acceptance test is run to determine whether the version has performed a valid operation. If the primary version fails to completeor fails the acceptance test, the system state is restored back to its current state before entry into the primary version, it does so by using checkpoint and an alternate version is performed. If the primary version passes the acceptance test, all alternates are ignored and if of all of alternate versions fails to pass the acceptance test, then the entire recovery block is considered to have failed, so the block in which it is embedded fails to complete [7].

The general syntax of RB is:

**ensure Acceptance Test**

**by Primary Alternate**

**else by Alternate 2**

**else by Alternate 3**

**...**

**else by Alternate n**

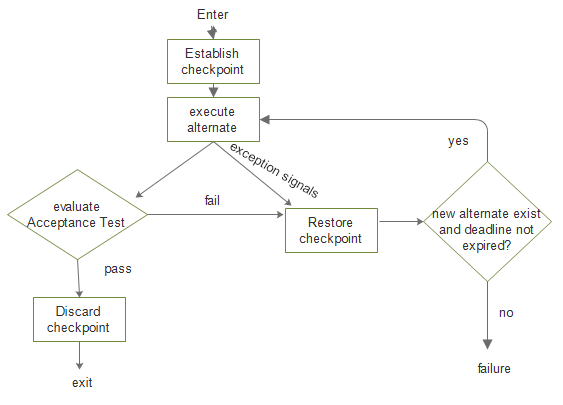
**else failure exception**

Figure : Recovery Block Structure

Consider the figure-2 which demonstrate how the RB techniques work

From figure-2, recovery blocks use an Acceptance Test (AT) to accomplish fault tolerance. Before execution of any alternate the system enters into checkpoint to save the current stable state of the system. The primary alternate start to execute and the result is checked to ensure whether it pass the acceptance test or not. If it passes the AT others alternates are ignored and the final result is produced and if it fails, the system restores back to its stable state using checkpoint and other alternate is switched on until the acceptance test is passed. If all the alternates did not pass the Acceptance test, then the system enter into a failure.

## SELF-CHECKING SOFTWARE

It is proposed that one among the techniques for implementing software fault tolerance is to detect the errors in the program code that could propagate in the system component and causes fault that may result to system failure. Therefore, some researchers have suggested the implementation of Self-checking software. Self-checking software is one which can be able to detect software errors and stop the propagation of those errors from affecting other component in the software. Self-checking software uses program redundancy to check its own behavior during program execution to determine whether the output produced by a program module is valid or not. By checking the behavior of the constituent modules in a software system it is possible to detect errors that remain in software during development, thereby improving its reliability. Since the ability of a self-checking software is to detect error in itself depends on the error-trapping capability of the formulated normal behavior. There is the need for the formulation of a set of standards, for verifying the normal behavior of a module [4]. Self-checking techniques can be implemented in the program to check the control sequence, the function, and the both input and output data of a process. These self-checking techniques should be implemented at the initial stage of program development. According to the research of [4,9] the checks are categorized in diagnostic, replication, timing, reversal, coding, reasonableness, and structural checks.

# RESULTS AND DISCUSIONS

After reviewing a number of papers which is published by different authors, the following results were obtained concerning the effectiveness of N-version Programming, Recovery Block and Self-Checking software:

**N-Version Programming** is based on design diversity technique, which use the assumptions that multiple version of software implemented differently will fail independently, thus ensuring that if one among the version fails other versions will continue to deliver a required functionality. Due to the research done by different authors such as [5,] it is observed that N-version programming technique can be able to tolerate the design faults presents in the software if and only if the design diversity is implemented properly. It is advisable that each version of the module should be implemented using different tool sets, different developing teams different programming languages, and possibly different environments. However, the cost of using NVP has become another important issue. [6] It is very expensive to produce designs and implementations of N versions of a given program instead of a single one. Also, on designing N-version programming, requirements specifications of the software should be taken in to considerations. The specifications must be clear and unambiguous so as to guarantee that different developing team will produce the versions which provide the same functionality otherwise the design diversity will not be achieved and hence software fault of the system will still cause system failure.

Not only that but also the problem with this approach is the use of Decision Mechanism or Voter which generate the final result based on the majority voting. Voter may produce incorrect decision when majority of outputs are incorrect since it is not clear that n-modules will not have correlated faults [7]. Moreover, due to research done by some authors it shows that the NVP techniques cannot be applicable to the situations in which distinct multiple solutions exist.

**Recovery Block**

**Self-Checking Software** technique is considered to be cost-effectiveness techniques for tolerating software faults. This technique is based on the single design techniques where mechanism is added to the software to ensure it will be able to detect errors and control the propagation of that error from affecting the other components of the system. However, this technique lacks a number of research done on its effectiveness of tolerating fault. The experiment conducted in the research done by [8] was based on how to design the programs with self-checks and determine how effective these checks were in detecting errors in the programs. In that experiments a number of programs were developed by different participants, and some errors were added on that programs. The aims of the experiment were to detect those errors presented on that programs. But the result obtained show that out of 60 known errors only 20 errors were able to be detected. This implies that the self-checking software does not guarantee to detect errors. But according to that author one among the reasons which causes those errors not to be detected is incorrect placements of self-checks code in the programs. Therefore, it is suggested that more training or experience might be helpful in the design of software fault tolerance using this technique.

# CONCLUSION AND FUTURE WORK

Due to highly increasing of the use of computer system, to make those systems dependable is the goals that is needed to be achieved no matter what. Since its clear that due to the limitations of human capabilities it is impossible to design a system which is free from errors. Therefore, in order to handle those errors that remains in the software after its development, the need for making software tolerating fault that may leads to system failure become an interested issue. Different researchers have suggested a number of approaches that would make the goal of getting dependable system be achieved. The approaches are categorized on design diversity and a single design. Design diversity is based on the assumption that software implemented differently will fail independently and hence it would be able to mask the fault occurring in the system since if one version fails another one will provide a required service. Single Design is based on adding some mechanisms in a single software that would make it able to detect errors in the program code and preventing the propagation of that errors from affecting other components in the system. This paper review the three approaches for designing software fault tolerance which are N-version Programming, Recovery Block and Self-Checking software. N-version Programming and Recovery Block are based on Design Diversity while Self-Checking Software is based on a single Design. The result shows that all the discussed approaches do not guarantee to tolerant faults in software by 100% efficiency, each one has its own strength and weakness on its own way. Therefore, the choice of these approaches depend upon the nature of the application software, and it is advisable to use more than one approaches on guarantying fault tolerant in software system.

The further study of the research is based on suggesting on the new fault software approaches that would

.

# REFFERENCE

1. Lee and R. K. Ayer, "Faults, Symptoms, and Software Fault Tolerance in the Tandem GUARDIAN90 Operating System", IEEE 1993, pp. 20-29.
2. Dr. K. C. Joshi, “Approaches of Software Fault Tolerance Computing ", International Journal of Advanced Research in Computer Science and Software Engineering, Vol 5, Issue 9, September 2015.
3. Jashan Deep and Rajiv Mahajan, “An Efficient Approach For Software Fault Tolerance In Parallel Computing”, International Journal of Computer Engineering & Science, March 2014.
4. P. K. Lala and K. C. Yarlagadda, “On Self-checking software design”, IEEE, 1991.

1. Bharathi V, “N-Version programming method of Software Fault Tolerance: A Critical Review”, National Conference On Nonlinear Systems & Dynamics, NCNSD-2003
2. Anjushi Verma, Ankur Ghaartan and Tirthankar Gayen, “Review of Software Fault-Tolerance Methods for Reliability Enhancement of Real-Time Software Systems”, International Journal of Electrical and Computer Engineering (IJECE), Vol. 6, No. 3, June 2016, pp. 1031 – 1037.
3. Sunita and Shekher Dahiya, “Review Paper on Recovery of Data during Software Fault”, International Journal of Innovative Research in Advanced Engineering (IJIRAE), Volume 1 Issue 2 (April 2014).
4. NANCY G. LEVESON, STEPHEN S. CHA, JOHN C. KNIGHT AND TIMOTHY J. SHIMEALL, “The Use of Self Checks and Voting in Software Error Detection: An Empirical Study”, IEEE TRANSACTIONS OFSOFTWARE ENGINEERING. VOL 16. NO 4. APRIL I990.
5. Goutam Kumar Saha, “Approaches to Software Based Fault Tolerance – A Review”, Computer Science Journal of Moldova, vol.13, no.3(39), 2005.
6. Lance Fiondella and Panlop Zeephongsekul, “Recovery Block Fault Tolerance Considering Correlated Failures”, IEEE 2014.
7. Laprie, J., C., and K., “Definition and Analysis of Hardware- and Software-Fault-Tolerant Architectures,” IEEE Computer, Vol. 23, No. 7, 1990, pp. 39-51.