Software Fault Tolerance

In the past decade, computer-based systems have increased intensely in scope, complexity, and pervasiveness. Most industries are highly dependent on computers for their basic day-to-day functioning. Safe and reliable software operation is a significant requirement in many types of systems like automobiles, aircraft, medical devices, temperature control and more. The cost and consequences of these systems failing can range from mildly annoying to extremely catastrophic. As software shoulders more of the responsibility for providing functionality in these systems, it becomes more complex and more significant to the overall system performance and dependability. Therefore, in order to ensure that these systems perform as specified, even under extreme condition, it is important to have a fault tolerant computing software or system.

Faults and Its Types:

Fault is a structural imperfection in a software that may eventually lead to the failure of the system on which the software is running. These failures may occur in the system because of an error or defect in the design or because of a simple mistake. However, all these faults can be traced back to two major types of faults that are:

1. **Hardware Faults:**
   The hardware faults are physical faults in the system. These can be easily characterised and predicted by the developer.

2. **Software Faults:**
   The errors detected in the design, coding, or the structure of a program or an application are the software faults. These are usually detected by the testers during the process of testing and are termed as logical errors. However, software faults can be difficult to visualize, classify, detect, and correct.

Ways to Avoid or Prevent Faults:

Faults can occur at any stage of software development process and can cause a minor or major failure. Hence, it is extremely necessary to employ some techniques or ways to ensure the prevention of errors in the software. The fault avoidance or prevention techniques are dependability enhancing techniques employed during software development to reduce the number of faults introduced during construction. These techniques contributes to system reliability through use of structured design and programming methods, use of formal methods with mathematically tractable languages and tools, and software reusability. Elaborated here are the most important ways of preventing as well as avoiding faults in a system:

1. **System Requirements Speci cation:** A system failure may occur due to logical errors incorporated into the requirements. This results in software that is written to match the requirements, but the behaviour specified in the requirements is not the expected or desired system behaviour. This type of error often occurs because software requirements specification lies at the intersection between software engineering and system engineering, and these two disciplines suffer from a lack of communication. Therefore, to prevent such errors, it is really necessary for the programmers to ensure that the specified system requirements fully achieved.

2. **Structured Design and Programming Methods:**
   It introduces the structure to the design to reduce the complexity and interdependency of components. It applies the principles of decoupling and modularization to the software as well as the system design. Each of these techniques reduces overall complexity of the software, making it easier to understand and implement which further reduces the introduction of faults into the software.

3. **Formal Method:**
   Formal methods have been used, particularly in the research community, to improve software dependability during construction. In these approaches, requirements specifications are developed and maintained using mathematically tractable languages and tools.

   Goals of current formal methods:

- To execute specifications for systematic and precise evaluation.
- Proof mechanisms for software verification and validation is provided.
- Develops procedures that follow incremental refinement for step-by-step verification.
- Every work item, be it a specification or a test case, is subject to mathematical verification for correctness and appropriateness.

4. **Software Reuse:**
   Software reusability implies a savings in development cost as it reduces the number of components that are originally developed. It is a popular means of increasing dependability, as software that has been well exercised is less likely to fail. In addition, object-oriented paradigms and techniques encourage and support software reuse, which further increases software reuses' popularity. However, one should recognize that different measures of dependability may not be improved equally by reuse of software.

What is Software Fault Tolerance?

Software fault tolerance is the ability of a software to detect and recover from a fault that is happening or has already happened. These faults are usually found in either the software or hardware of the system in which the software is running in order to provide service in accordance to the provided specifications. Software fault tolerance is a necessary component, as it provides protection against errors in translating the requirements and algorithms into a programming language. But, it does have one disadvantage that is it does not provide explicit protection against errors in specifying the requirements.

Software Fault Tolerance Techniques:

Nowadays, the need to avoid faults is a dominating factor all around the world. Whether you look at a child or an expert, everyone strives for perfection and try their best to avoid as well as prevent errors and mistakes. Similarly, software designers and programmers too, try their best to create software applications with no mistakes or faults. As software development requires huge amount of processing and coding, finding a fault or an error is extremely common during the development process. To ensure that structural and functional errors do not cause any major failure in future and to get faultless results, developers follow the below mentioned techniques and secure the quality and result of their product.

- **Recovery Block:**
  The recovery block operates with an adjudicator which confirms the results of various implementations of the same algorithm. With the assistance of recovery blocks, the system view is broken down into fault recoverable blocks and then the entire system is constructed of the fault tolerant blocks. Each of these blocks consist of at least a primary, secondary, and exceptional case code along with an adjudicator.
  The adjudicator is kept somewhat simple in order to maintain execution speed and aide in correctness. Upon first entering a unit, the adjudicator first executes the primary alternate. If the adjudicator determines that the primary block failed, it then tries to roll back the state of the system and tries the secondary alternate. If the adjudicator does not accept the results of any of the alternates, it then invokes the exception handler, which then indicates the fact that the software could not perform the requested operation. The recovery block method increases the pressure on the specification to be accurate enough to create different or multiple alternatives that are functionally the same.

- **N-version Software:** The N-version software concept attempts to parallel the traditional fault tolerance N-way redundant hardware. In an N-version software system, each module is made up to N different implementations and each of these variants accomplish the same task, but in a different manner. All the versions then submit its answer to the decider, who determines the correct answer and then returns it as the result of the module.

  An important distinction in N-version software is that, the system can include multiple types of hardware using multiple versions of software. The main goal of this technique is to increase the diversity in order to avoid common mode failures. Another benefit of using N-version software is

that it encourages implementation of each of the different versions in as diverse manner as possible, including different tools sets, programming language, and possibly different environment.

The various development groups must have as little interaction related to the programming between them as possible. N-version software can only be successful and successfully tolerate faults if the required design diversity is met. However, the N-version method presents the possibility of various faults being generated, but successfully masked and ignored within the system.

- **Self-Checking Software:**

  Self-checking software is not a rigorously described method, but is a more ad hoc method used in some important systems. Self-checking software provides extra checks, often including some check pointing and rollback recovery methods added into fault tolerant or safety critical systems.

  There are separate tasks executed by self-checking software that are majorly focused in finding and correcting data defects and the options of using degraded performance algorithms. Even though this type of software tolerance technique may not be rigorous methodology, it does provide successful results and has shown to be very effective.

  But, self-checking software does have one disadvantage. Its lack of rigor and unknown code coverage for a fault tolerant system is obviously problematic. Also, without the proper rigor and experiments comparing and improving self-checking software cannot effectively be done.

Conclusion:

Safe and faultless development of a software is a very crucial aspect for programmers and software designers. These experts use various methodologies and techniques to create a product that fulfils all the expectations of the client and is created as per specified requirements. Similarly, to accomplish such a demand of perfect software, developers use Software Fault Tolerance to make the software capable of protecting itself against failures in future. Software Fault Tolerance ensures that whenever a fault occurs in the software of the system on which it is running, it provides mechanisms to prevent system failure. This is accomplished by providing protection against errors in translating the requirements and algorithms into the programming language.