- **Validation:** those that contribute to validation of the software after it is developed (fault removal and fault forecasting).

3. **The Impairments to Dependability**: faults, errors, failures; they are undesired circumstances causing or resulting from the lack of dependability.

4.1. Fault Avoidance or Prevention:

Fault avoidance or prevention techniques are dependability enhancing techniques employed during software development to reduce the number of faults introduced during construction. This avoidance, or prevention, techniques may address:

A. **System Requirements Specification:** A system failure may occur due to logic errors incorporated into the requirements. This results in software that is written to match the requirements, but the behavior specified in the requirements is not the expected or desired system behavior.

B. **Structured Design and Programming Methods:** These techniques reduce the overall complexity of the software, making it easier to understand and implement and, hence, reduce the introduction of faults into the software.

C. **Formal Methods:** have been used to improve software dependability during construction <u>There are four goals of current formal methods:</u>

1) executable specifications for systematic and precise evaluation,
2) proof mechanisms for software verification and validation,
3) development procedures that follow incremental refinement for step-by-step verification, and

4) every work item, either a specification or a test case, is subject to mathematical verification for correctness and appropriateness.

D. Software Reuse: is very attractive for a variety of reasons,

1) software reusability saves development cost, since it reduces the number of components that must be originally developed.
2) It is a means of increasing dependability because software that has been well exercised is less likely to fail.
3) object-oriented paradigms and techniques encourage and support software reuse.

4.2. Fault Removal

Fault removal techniques are dependability-enhancing techniques employed during software verification and validation to improve dependability by detecting existing faults, and eliminating the detected faults. Fault removal techniques contribute to system dependability using the following:

A. Software Testing: the most common fault removal techniques involve testing. The difficulties encountered in testing programs are often related to the expensive cost and complexity of **exhaustive testing** (<u>testing the software under all circumstances using all possible input sets</u>).

B. Formal Inspection: is another practical fault removal technique that has shown success in many companies. This technique is a difficult process, accompanied by documentation that focuses on examining source code to find faults, correcting the faults, and then verifying the corrections. Formal inspection is usually performed by small peer groups prior to the testing phase of the life cycle.

C. **Formal Design Proofs**: are closely related to formal methods. This technique attempts to achieve mathematical proof of correctness for programs. Using executable specifications, test cases can be automatically generated to improve the software verification process. This technique may be a costly and complex technique to use. However, if performed on a relatively small portion of the code, formal design proofs may be feasible.

4.3. Fault/Failure Forecasting

Fault forecasting, or failure forecasting, includes dependability enhancing techniques that are used during the validation of software to estimate the presence of faults and the occurrence and consequences of failures. Fault forecasting usually focuses on the reliability measure of dependability and is also known as software reliability measurement. Fault forecasting techniques can help predict the effectiveness of additional testing efforts. It includes two types of activities:

A. **Reliability Estimation:** determines **current** software reliability by applying statistical inference techniques to failure data obtained during system testing or during system operation. Reliability estimation is thus a snapshot of the reliability that has been achieved to the time of estimation.

4. **Reliability Prediction:** determines **future** software reliability based upon available software metrics and measures. Different techniques are used depending on the software development stage.

4.4. Fault Tolerance

One way to reduce the risks of software design faults and thus enhance software dependability is to use software fault tolerance techniques. Software fault tolerance techniques are employed during development of the software. They enable a system to tolerate software faults remaining in the system after its development. When a fault occurs, these techniques provide mechanisms to the software system to prevent system failure from occurring. Software fault tolerance provides service complying with the relevant specification in spite of faults by typically using:

A. **Single Version Software Environment:** these techniques are used to partially tolerate software design faults monitoring techniques, decision verification, and exception handling.

B. **Multiple Version Software Environment:** Design diverse techniques are used in a multiple version (or variant) software environment and utilize functionally equivalent yet independently developed software versions to provide tolerance to software design faults. Examples of such techniques include Recovery Blocks (RcB), N-Version Programming (NVP), and N Self-Checking Programming (NSCP).

C. **Multiple Data Representation Environment** Data diverse techniques are used in a multiple data representation environment and utilize different representations of input data to provide tolerance to software design faults. Examples of such techniques include Retry Blocks (RtB) and N-Copy Programming (NCP).

1) <u>Fault-tolerance scope of systems:</u>

Fault-avoidance is difficult, and close to impossible in large and complex systems. This makes fault-tolerance the only realistic alternative for the commonly known classes of systems:

B **General-purpose computer systems**: General-purpose computers in the high-end of the commercial market, employing fault-tolerance techniques to improve general reliability.

C **High-available computer systems**: Systems designed for availability such as a kind of database system or telephone switching system.

D **Long-life systems**: Systems designed for operating for a very long time without any chance of repair. Long-life systems are typical mobile systems where on-site repair is difficult, or maybe impossible.

E **Critical-computation systems:** Systems doing some critical work where faulty computations can jeopardize human life or have high economic impact. This could be the computers in a space shuttle, nuclear plant or air traffic control system, where malfunction can be extremely disastrous.

4) <u>Failures and Failure Modes</u>

Software, especially in critical systems, tends to fail where least expected. We are usually extremely good at setting up test plans for the main line code of the program, and these sections usually do run flawlessly.

Software does not "break" but it must be able to deal with "broken" input and conditions, which are often causes for **software failures**. The task of dealing with abnormal/anomalous conditions and inputs is handled by the **exception code** dispersed throughout the program. Setting up a test plan and exhaustive **test cases** for the exception code is by definition difficult.