**Software Issues**

**Standards**

A **software standard** is a [standard](https://en.wikipedia.org/wiki/Technical_standard), [protocol](https://en.wikipedia.org/wiki/Communications_protocol), or other common format of a document, file, or data transfer accepted and used by one or more [software developers](https://en.wikipedia.org/wiki/Software_developers) while working on one or more than one [computer programs](https://en.wikipedia.org/wiki/Computer_program). Software standards enable interoperability between different programs created by different developers.

Software standards consist of certain terms, concepts, data formats, document styles and techniques agreed upon by software creators so that their software can understand the files and data created by a different computer program. To be considered a standard, a certain protocol needs to be accepted and incorporated by a group of developers who contribute to the definition and maintenance of the standard.

Some developers prefer using standards for software development because of the efficiencies it provides for code development[[1]](https://en.wikipedia.org/wiki/Software_standard#cite_note-1) and wider user acceptance and use of the resulting application.

For example, the protocols [HTML](https://en.wikipedia.org/wiki/HTML), [TCP/IP](https://en.wikipedia.org/wiki/TCP/IP), [SMTP](https://en.wikipedia.org/wiki/SMTP), [POP](https://en.wikipedia.org/wiki/Post_Office_Protocol) and [FTP](https://en.wikipedia.org/wiki/FTP) are software standards that application designers must understand and follow if their software expects to interface with these standards. For instance, in order for an email sent from [Microsoft Outlook](https://en.wikipedia.org/wiki/Microsoft_Outlook) can be read from within the [Yahoo! Mail](https://en.wikipedia.org/wiki/Yahoo!_Mail) application, the email will be sent using SMTP, which the different receiving program understands and can parse properly to display the email. Without a standardized technique to send an email, the two different programs would be unable to accurately share and display the delivered information.

Open vs Closed Standards

A standard can be a closed standard or an [open standard](https://en.wikipedia.org/wiki/Open_standard). The documentation for an open standard is open to the public and anyone can create a software that implements and uses the standard. The documentation and specification for closed standards are not available to the public, enabling its developer to sell and license the code to manage their data format to other interested software developers. While this process increases the revenue potential for a useful file format, it may limit acceptance and drive the adoption of a similar, open standard instead.

**Software Reliability:**

Software Reliability is the probability of failure-free software operation for a specified period of time in a specified environment. Software Reliability is also an important factor affecting system reliability. It differs from hardware reliability in that it reflects the design perfection, rather than manufacturing perfection. The high complexity of software is the major contributing factor of Software Reliability problems.

Software Reliability is not a function of time - although researchers have come up with models relating the two. The modeling technique for Software Reliability is reaching its prosperity, but before using the technique, we must carefully select the appropriate model that can best suit our case. Measurement in software is still in its infancy. No good quantitative methods have been developed to represent Software Reliability without excessive limitations. Various approaches can be used to improve the reliability of software, however, it is hard to balance development time and budget with software reliability.

A good software reliability engineering program, introduced early in the development cycle, will mitigate these problems by: Preparing program management in advance for the testing effort and allowing them to plan both schedule and budget to cover the required testing.

Continuous review of requirements throughout the life cycle, particularly for handling of exception conditions. If requirements are incomplete there will be no testing of the exception conditions.

SoHaR software reliability engineers are experienced in all the stages and tasks required in a comprehensive software reliability program. We can support or lead tasks such as:

1) Reliability Allocation

2) Defining and Analyzing Operational Profiles

3) Test Preparation and Plan

4) Software Reliability Models

**Reliability Allocation**:-

Reliability allocation is the task of defining the necessary reliability of a software item. The item may be part of an integrated hardware/software system, may be a relatively independent software application, or, more and more rarely, a standalone software program. In either of these cases our goal is to bring system reliability within either a strict constraint required by a customer or an internally perceived readiness level, or optimize reliability within schedule and cost constraints.

SoHaR will assist your organization in the following tasks:

Derive software reliability requirements from overall system reliability requirements.

When possible, depending on lifecycle stage and historical data, estimate schedule and cost dependence on software reliability goals.

Optimize reliability/schedule/cost based on your constraints and your customer's requirements,

**Defining and Analyzing Operational Profiles**:-

The reliability of software, much more so than the reliability of hardware, is strongly tied to the operational usage of an application. A software fault may lead to system failure only if that fault is encountered during operational usage. If a fault is not accessed in a specific operational mode, it will not cause failures at all. It will cause failure more often if it is located in code that is part of an often used "operation" (An operation is defined as a major logical task, usually repeated multiple times within an hour of application usage). Therefore in software reliability engineering we focus on the operational profile of the software which weighs the occurrence probabilities of each operation. Unless safety requirements indicate a modification of this approach we will prioritize our testing according to this profile.

SoHaR will work with your system and software engineers to complete the following tasks required to generate a useable operational profile:

Determine the operational modes (high traffic, low traffic, high maintenance, remote use, local use etc).

Determine operation initiators (components that initiate the operations in the system).

Determine and group "Operations" so that the list includes only operations that are significantly different from each other (and therefore may present different faults).

Determine occurrence rates for the different operations.

Construct the operational profile based on the individual operation probabilities of occurrence.

**Test Preparation and Plan**:-

Test preparation is a crucial step in the implementation of an effective software reliability program. A test plan that is based on the operational profile on the one hand, and subject to the reliability allocation constraints on the other, will be effective at bringing the program to its reliability goals in the least amount of time and cost.

Software Reliability Engineering is concerned not only with feature and regression test, but also with load test and performance test. All these should be planned based on the activities outlined above.

The reliability program will inform and often determine the following test preparation activities:

Assessing the number of new test cases required for the current release.

New test case allocation among the systems (if multi-system).

New test case allocation for each system among its new operations.

Specifying new test cases

Adding the new test cases to the test cases from previous releases.

**Software Reliability Models**:-

Software reliability engineering is often identified with reliability models, in particular reliability growth models. These, when applied correctly, are successful at providing guidance to management decisions such as:

Test schedule

Test resource allocation

Time to market

Maintenance resource allocation

**Software Security**

Any compromise to integrity, authentication and availability makes a software unsecure. Software systems can be attacked to steal information, monitor content, introduce vulnerabilities and damage the behavior of software. Malware can cause DoS (denial of service) or crash the system itself.  
  
Buffer overflow, stack overflow, command injection and SQL injections are the most common attacks on the software.  
  
Buffer and stack overflow attacks overwrite the contents of the heap or stack respectively by writing extra bytes.  
  
Command injection can be achieved on the software code when system commands are used predominantly. New system commands are appended to existing commands by the malicious attack. Sometimes system command may stop services and cause DoS.  
  
SQL injections use malicious SQL code to retrieve or modify important information from database servers. SQL injections can be used to bypass login credentials. Sometimes SQL injections fetch important information from a database or delete all important data from a database.  
  
The only way to avoid such attacks is to practice good programming techniques. System-level security can be provided using better firewalls. Using intrusion detection and prevention can also aid in stopping attackers from easy access to the system.

**Software Safety**

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.  
  
However, the main problem with using the traditional system safety method on software is that the probability of software failure is not measurable or even easily estimated. Traditional system safety uses a combination of probability and severity to rate the risk of each hazard. Software does not "fail" after it is completed. What happens is that latent defects in the original product assert themselves later in the life of the product, potentially causing safety problems. The methods described in the handbooks above use an alternative to probability - "software control authority" that is, how much control the software has over the system and how much time there is for a human operator to intervene if the software does something unexpected.  
  
An alternative approach is to use the techniques of Software Reliability Engineering to develop estimates of the reliability of a piece of software as it is going through the development process. This is done by tracking the "bug reports" and matching the rate of bug removal to an exponential curve. The curve is then extrapolated to estimate future reliability

**Software Quality**

Software quality is an abstract concept. Its presence can be difficult to define, but its absence can be easy to see instantly. Thus, in the quest for improving software quality, we must first understand the software quality definition. [Wikipedia](https://en.wikipedia.org/wiki/Software_quality)describes software quality as follows:

*“In the context of software engineering, software quality measures how well software is designed (quality of design), and how well the software conforms to that design (quality of conformance). It is often described as the ‘fitness for purpose’ of a piece of software.”*

There are many variations to the definition of software quality, but if you examine the definition above, “fitness for purpose” questions whether or not the software fulfills its purpose, or in layman’s terms, “Does it do what it’s supposed to do?” Those are the characteristics that we see as end users. Quality of design and quality of conformance to that design are related to internal aspects of the software, some of which we may see, like the user interface’s navigation, placement of controls and so on. Others, we would not normally see, like code architecture, code quality and code security.