SOFTWARE RELIABILITY & SAFETY

ALD offers a range of services targeted at improving the reliability, dependability and safety of your software. Whether your software is safety-critical, mission-critical, or expected to satisfy strict reliability and availability requirements to be certified as market-ready, we can lead and support your effort in attaining these goals.

SOFTWARE FAILURES

Before we list the tasks undertaken to analyze software reliability and safety it is important to understand the meaning of a failure due to software. Software does not exhibit the random or wear out related failure behavior we see in hardware. Software will always function in the same way as long as the same input and computer states are present. Software can cause system failures either because of design errors or implementation errors. Design errors are often caused by wrong assumptions about system operation, e. g., that input A is always followed by input B. Typical implementation errors are caused by confusing symbols, such as 'g' instead of 'G'. Software faults can only cause failures if the fault is encountered during usage. Therefore, faults existing in frequently used code will cause failures more often than faults residing in rarely used code but the latter can be equally serious. In mission or safety critical applications it is particularly important to review and test rarely used code.

SOFTWARE RELIABILITY

As is the case for hardware, software reliability engineering involves much more than analyzing test results, estimating remaining faults, and modeling future failure probabilities.
Although in most organizations software test is no longer an afterthought, management is almost always surprised by the cost and schedule requirements of the test program, and it is often downgraded in favor of design activities. Often adding a new feature will seem more beneficial than performing a complete test on existing features. 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.
• Offering management a quantitative assessment of the dependence of reliability metrics (software/system availability; software/system outages per day, etc.) on the effort (time and cost) allotted to testing.
• Providing the most efficient test plan targeted to bringing the product to market in the shortest time subject to the reliability requirements imposed by the customer or market expectations.

• Performing continuous quantitative assessment of software/system reliability and the effort/cost required to improve them by a specified amount.

ALD software reliability engineers are experienced in all the stages and tasks required in a comprehensive software reliability program.

We can support or lead such reliability program tasks as:

• Reliability Allocation
• Defining and Analyzing Operational Profiles
• Test Preparation and Plan
• Software Reliability Models

RELIABILITY ALLOCATION

Reliability allocation is the task of defining the necessary reliability of a software item. The item may be a part of an integrated hardware/software system, may be a relatively independent software application, or, more and more rarely, a standalone software program. In any 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.

ALD 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

Ideally, reliability allocation should be performed early in the lifecycle and may be modified and refined as both software and other system components are developed. At these early stages, ALD can assist in the above tasks with limited design and requirements inputs. As the system is developing and getting more mature, software reliability allocation becomes more accurate. The dependence of reliability allocation on cost and schedule also solidifies when the software goes into testing. Although it is ideal to begin these tasks early on and follow during system evolution, often organizations do not implement a software reliability program until very late in the software development cycle. The delay may be up to the time of test preparation and plan, or even later when testing has started yielding results that need to be interpreted to verify or ascertain reliability.

DEFINING AND ANALYZING OPERATIONAL PROFILES

The reliability of software is strongly tied to the operational usage of an application - much stronger than the reliability of hardware. A software fault may lead to a 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 a frequently 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.

ALD 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 in achieving the program's 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 existing test cases from previous releases

SOFTWARE RELIABILITY MODELS

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

- **Test schedule**
- **Test resource allocation**
- **Time to market**
- **Maintenance resource allocation**

The application of reliability models to software testing results allows us to infer the rate at which failures are encountered (depending on usage profile) and, more importantly, the changes in this rate (reliability growth). The ability to make these inferences depends critically on the quality of test results. It is essential that testing be performed in such a way that each failure incident is accurately reported.

ALD's software reliability engineers will work with developers, testers and program management to apply an appropriate model to your failure data. In order for the model prediction to be useful we must ensure that the assumptions and structure of the model coincide with the underlying coding and testing process. It is not sufficient to find a mathematical function that best fits the data. In order to infer future failure behavior it is crucial that the underlying assumptions of the model be understood in terms of program management and progress towards release. This requires experience working with the software reliability models as well as understanding of latent issues in the development and testing process that may impact the test data.

Software Safety

As systems and products become more and more dependent on software components it is no longer realistic to develop a system safety program that does not include the software elements.

Does software fail?

We tend to believe that well written and well tested safety critical software would never fail. Experience proves otherwise with software making headlines when it actually does fail, sometimes critically. Software does not fail the same way as hardware does, and the various failure behaviors we are accustomed to from the world of hardware are often not applicable to software. However, software does fail, and when it does, it can be just as catastrophic as hardware failures.

Safety-critical software

Safety-critical software is very different from both non-critical software and safety-critical hardware. The difference lies in the massive testing program that such software undergoes.

What are "software failure modes"?

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 often cause the "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 and somewhat subjective.

Anomalous inputs can be due to:

• failed hardware
• timing problems
• harsh/unexpected environmental conditions
• multiple changes in conditions and inputs that are beyond what the hardware is able to deal with
• unanticipated conditions during software mode changes
• bad or unexpected user input

Often the conditions most difficult to predict are multiple, coinciding, irregular inputs and conditions.

Safety-critical software is usually tested to the point that no new critical failures are observed. This of course does not mean that the software is fault-free at this point, only that failures are no longer observed in test. Why are the faults leading to these types of failures overseen in test? These are faults that are not tested for any of the following reasons:

• Faults in code that is not frequently used and therefore not well represented in the operational profiles used for testing
• Faults caused by multiple anomalous conditions that are difficult to test
• Faults related to interfaces and controls of failed hardware
• Faults due to missing requirements

It is clear why these types of faults may remain outside of a normal, reliability focused, test plan.

How to protect against such failures once software is released?

Current guides and standards are not nearly as specific and clear as the hardware equivalent. Most notably RTCA/DO-178B, the Radio Technical Commission for Aeronautics Software

Considerations in Airborne Systems and Equipment Certification document, whose latest version was issued in 1992, and is considered to be the main guideline for safety-critical software development does not deal in particular with types of failures and how to avoid them. The document deals mainly with process control that will hopefully ensure good software. It does not dictate how to verify that the process works "well enough" for the requirements of any particular system. The much awaited update to this document, DO-178C, was released in 2011 and most certainly offered more prescriptive guidelines to the certification of safety-critical software. These are based not only on more current design environments such as Object Oriented Design but in general model-based software and more current formal methods for verification.

Since safety-critical software is most often a part of a larger system that includes hardware, then the safety assessment process should follow the process applied to hardware:

SOFTWARE PRELIMINARY/FUNCTIONAL HAZARD ANALYSIS (PHA/FHA)

If the top level architecture of the system details software components it is necessary that they be included in this qualitative analysis. The PHA is also needed to formulate a software specification that (a) prevents software from interfering with established hazard handling provisions, and (b) directs software design to reinforce hazard handling where established provisions are lacking. The analysis is necessary for completing the later stages of the safety review. ALD will work with your requirements and specification documents as well as any early design documents and artifacts available. Model based artifacts such as use-case scenarios are very helpful at this level of analysis.

WHAT IS A FAILURE IN SOFTWARE TESTING?

If under certain environment and situation defects in the application or product get executed then the system will produce the wrong results causing a failure.

Defects and failures basically arise from:

- Errors in the specification, design and implementation of the software and system
- Errors in use of the system
- Environmental conditions
- Intentional damage

- Potential consequences of earlier errors

Errors in the specification and design of the software:

Specification is basically a written document which describes the functional and non – functional aspects of the software by using prose and pictures. For testing specifications there is no need of having code. Without having code we can test the specifications. About 55% of all the bugs present in the product are because of the mistakes present in the specification. Hence testing the specifications can save lots of time and the cost in future or in later stages of the product.

Errors in use of the system:

Errors in use of the system or product or application may arise because of the following reasons:

– Inadequate knowledge of the product or the software to the tester. The tester may not be aware of the functionalities of the product and hence while testing the product there might be some defects or failures.

– Lack of the understanding of the functionalities by the developer. It may also happen that the developers may not have understood the functionalities of the product or application properly. Based on their understanding the feature they will develop may not match with the specifications. Hence this may result into the defect or failure.

Environmental conditions:

Because of the wrong setup of the testing environment testers may report the defects or failures. As per the recent surveys it has been observed that about 40% of the tester's time is consumed because of the environment issues and this has a great impact on quality and productivity. Hence proper test environments are required for quality and on time delivery of the product to the customers.

Intentional damage:

The defects and failures reported by the testers while testing the product or the application may arise because of the intentional damage.

Consider an example where an application is not secure and does not check for SQL Injections. During security testing, testers can inject SQL commands that may result in the application data or database being corrupted. In this case the intentional damage would have been caused and reported by the testers.

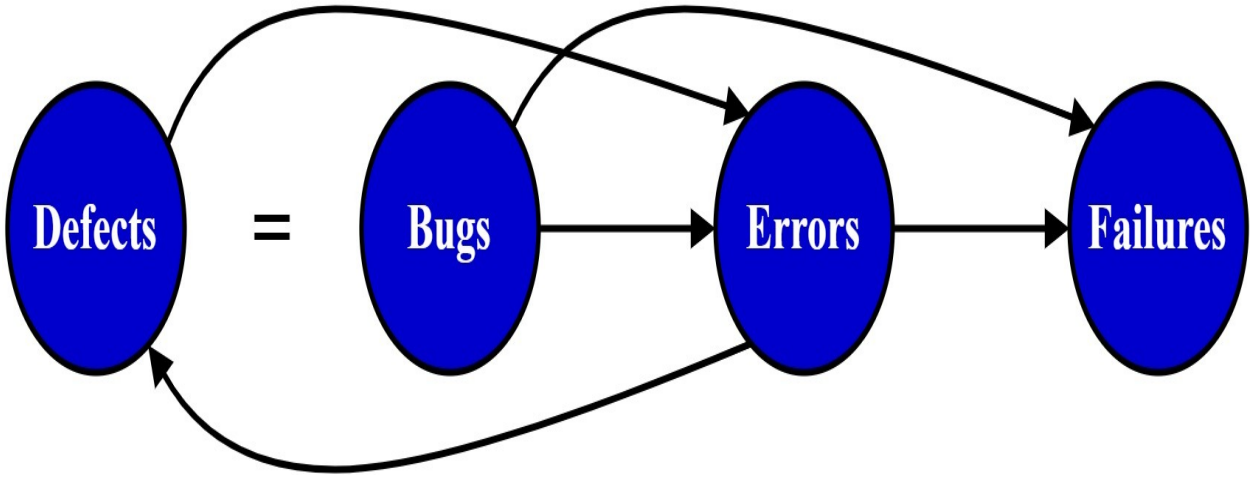
If this issue is not caught, it could be exploited by hackers who could also inflict intentional damage.

Potential consequences of earlier errors:

Errors found in the earlier stages of the development reduce our cost of production. Hence it's very important to find the error at the earlier stage. This could be done by reviewing the specification documents or by walkthrough. The downward flow of the defect will increase the cost of production.

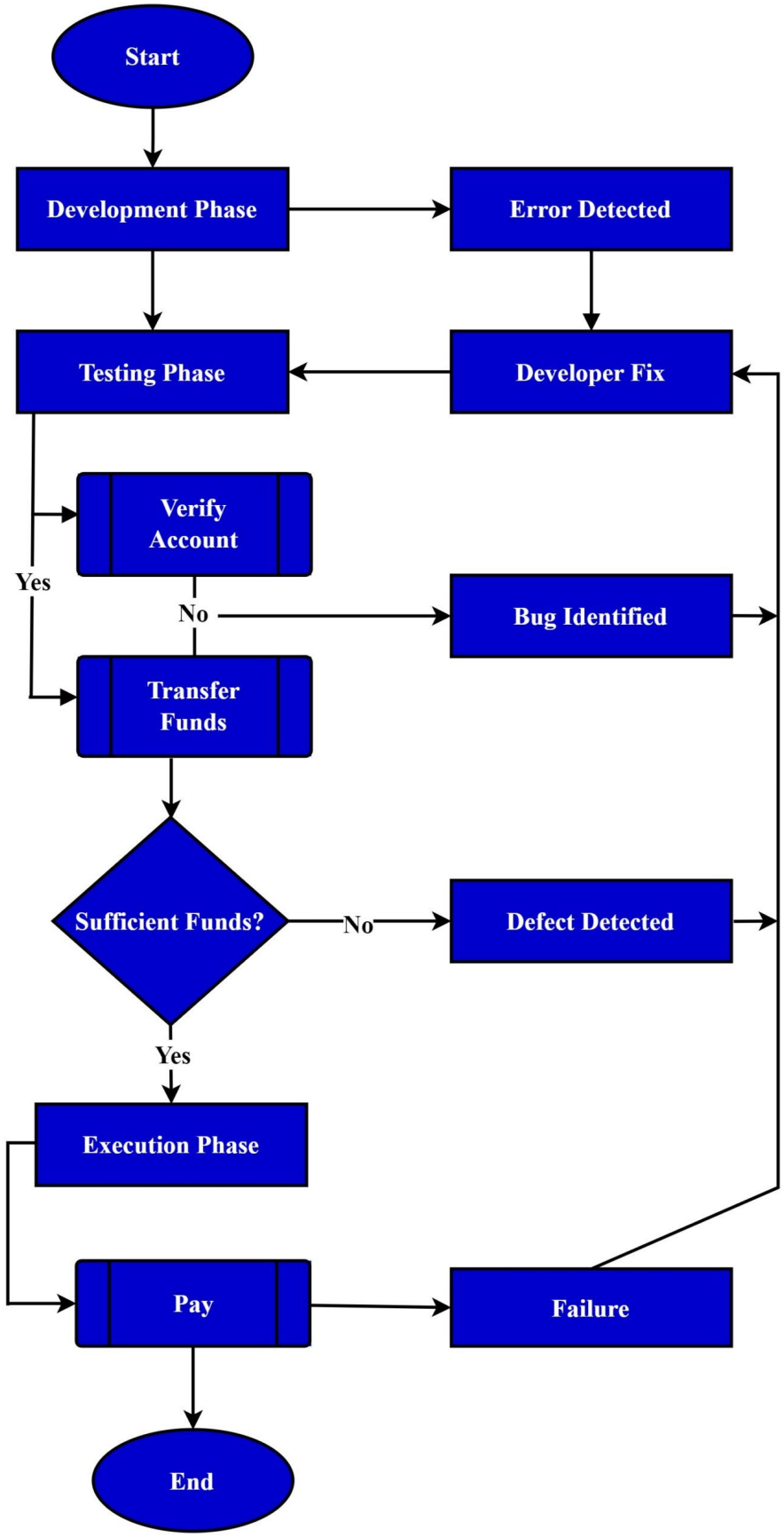
ERROR AND FAULTS(DEFECTS)

While we often use defects and bugs interchangeably, it is essential to distinguish between them in the context of software testing. The distinction lies in the stages at which they occur:



Example: Online Banking Application

Let’s consider an online banking application designed for various banking transactions, such as checking account balances, transferring funds, and paying bills. During various phases of software development, the team examines the application’s functionalities to ensure it meets the desired requirements:


Start
Development Phase
Error Detected
Testing Phase
Developer Fix
Verify Account
Yes
No
Bug Identified
Transfer Funds
Sufficient Funds?
No
Defect Detected
Yes
Execution Phase
Pay
Failure
End

In one scenario, a tester attempts to transfer funds from one account to another using an online banking application. However, the application allows the transfer to proceed even if the sender's account has insufficient funds. This means that there is a defect in the system.

We find a bug in the application that allows transfers without verifying the sender's account balance. After identifying the issue, we report it and log it as a bug. Our analysis indicates that the root cause of this bug is the absence of confirmation for the sender's account balance.

Then, a developer analyzes the code responsible for fund transfers during the code review. The developer identifies an error where an incorrect variable is used for account balance validation. This error could potentially lead to further issues if left unresolved.