

Chapter 18



Chapter 18 Software Reliability

Learning Objective

... Define the basic principles underlying *software reliability engineering* (metrics, measurement, and prediction)



Frederick T Sheldon
Assistant Professor of Computer Science
Washington State University

Software Reliability

Categorizing and specifying the reliability of software systems

Objectives

- To discuss the problems of reliability specification and measurement
- To introduce reliability metrics and to discuss their use in reliability specification
- To describe the statistical testing process
- To show how reliability predications may be made from statistical test results

Topics covered

- Definition of reliability
- Reliability and efficiency
- Reliability metrics
- Reliability specification
- Statistical testing and operational profiles
- Reliability growth modeling
- Reliability prediction

What is reliability?

Probability of failure-free operation for a specified time in a specified environment for a given purpose

This means quite different things depending on the system and the users of that system

Informally, reliability is a measure of how well system users think it provides the services they require

Software reliability

Cannot be defined objectively

- Reliability measurements which are quoted out of context are not meaningful

Requires operational profile for its definition

- The operational profile defines the expected pattern of software usage

Must consider fault consequences

- Not all faults are equally serious. System is perceived as more unreliable if there are more serious faults

Failures and faults

A failure corresponds to unexpected run-time behavior observed by a user of the software

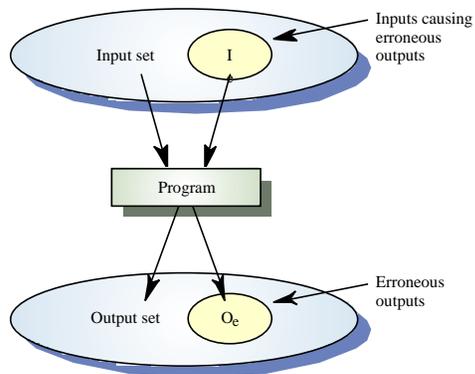
A fault is a static software characteristic which causes a failure to occur

Faults need not necessarily cause failures. They only do so if the faulty part of the software is used

If a user does not notice a failure, is it a failure?

Remember most users don't know the software specification

Input/output mapping



Reliability improvement

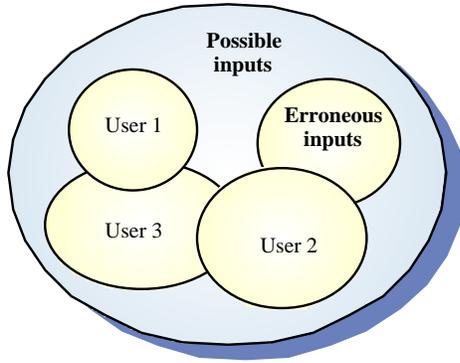
Reliability is improved when software faults which occur in the most frequently used parts of the software are removed

Removing x% of software faults will not necessarily lead to an x% reliability improvement

In a study, removing 60% of software defects actually led to a 3% reliability improvement

Removing faults with serious consequences is the most important objective

Reliability perception



Reliability and formal methods

The use of formal methods of development may lead to more reliable systems as it can be proved that the system conforms to its specification

The development of a formal specification forces a detailed analysis of the system which discovers anomalies and omissions in the specification

However, formal methods may not actually improve reliability

Reliability and formal methods

The specification may not reflect the real requirements of system users

A formal specification may hide problems because users don't understand it

Program proofs usually contain errors

The proof may make assumptions about the system's environment and use which are incorrect

Reliability and efficiency

As reliability increases system efficiency tends to decrease

To make a system more reliable, redundant code must be included to carry out run-time checks, etc. This tends to slow it down

Reliability and efficiency

Reliability is usually more important than efficiency

No need to utilize hardware to fullest extent as computers are cheap and fast

Unreliable software isn't used

Hard to improve unreliable systems

Software failure costs often far exceed system costs

Costs of data loss are very high

Reliability metrics

Hardware metrics not really suitable for software as they are based on component failures and the need to repair or replace a component once it has failed. The design is assumed to be correct

Software failures are always design failures. Often the system continues to be available in spite of the fact that a failure has occurred.

Reliability metrics

Probability of failure on demand

This is a measure of the likelihood that the system will fail when a service request is made

POFOD = 0.001 means 1 out of 1000 service requests result in failure

Relevant for safety-critical or non-stop systems

Rate of fault occurrence (ROCOF)

Frequency of occurrence of unexpected behavior

ROCOF of 0.02 means 2 failures are likely in each 100 operational time units

Relevant for operating systems, transaction processing systems

Reliability metrics

Mean time to failure

Measure of the time between observed failures

MTTF of 500 means that the time between failures is 500 time units

Relevant for systems with long transactions e.g. CAD systems

Availability

Measure of how likely the system is available for use. Takes repair/restart time into account

Availability of 0.998 means software is available for 998 out of 1000 time units

Relevant for continuously running systems e.g. telephone switching systems

Reliability measurement

Measure the number of system failures for a given number of system inputs

Used to compute POFOD

Measure the time (or number of transactions) between system failures

Used to compute ROCOF and MTTF

Measure the time to restart after failure

Used to compute AVAIL

Time units

Time units in reliability measurement must be carefully selected. Not the same for all systems

Raw execution time (for non-stop systems)

Calendar time (for systems which have a regular usage pattern e.g. systems which are always run once per day)

Number of transactions (for systems which are used on demand)

Failure consequences

Reliability measurements do NOT take the consequences of failure into account

Transient faults may have no real consequences but other faults may cause data loss or corruption and loss of system service

May be necessary to identify different failure classes and use different measurements for each of these

Reliability specification

Reliability requirements are only rarely expressed in a quantitative, verifiable way.

To verify reliability metrics, an operational profile must be specified as part of the test plan.

Reliability is dynamic - reliability specifications related to the source code are meaningless.

No more than N faults/1000 lines.

This is only useful for a post-delivery process analysis.

Failure classification

| Failure class | Description |
|----------------|--|
| Transient | Occurs only with certain inputs |
| Permanent | Occurs with all inputs |
| Recoverable | System can recover without operator intervention |
| Unrecoverable | Operator intervention needed to recover from failure |
| Non-corrupting | Failure does not corrupt system state or data |
| Corrupting | Failure corrupts system state or data |

Steps to a reliability specification

For each sub-system, analyze the consequences of possible system failures.

From the system failure analysis, partition failures into appropriate classes.

For each failure class identified, set out the reliability using an appropriate metric. Different metrics may be used for different reliability requirements.

Bank auto-teller system

Each machine in a network is used 300 times a day

Bank has 1000 machines

Lifetime of software release is 2 years

Each machine handles about 200,000 transactions

About 300,000 database transactions in total per day

Examples of a reliability spec.

| Failure class | Example | Reliability metric |
|----------------------------|--|---|
| Permanent, non-corrupting. | The system fails to operate with any card which is input. Software must be restarted to correct failure. | ROCOF 1 occurrence/1000 days |
| Transient, non-corrupting | The magnetic stripe data cannot be read on an undamaged card which is input. | POFOD 1 in 1000 transactions |
| Transient, corrupting | A pattern of transactions across the network causes database corruption. | Unquantifiable! Should never happen in the lifetime of the system |

Specification validation

It is impossible to empirically validate very high reliability specifications

No database corruption means POFOD of less than 1 in 200 million

If a transaction takes 1 second, then simulating one day's transactions takes 3.5 days

It would take longer than the system's lifetime to test it for reliability

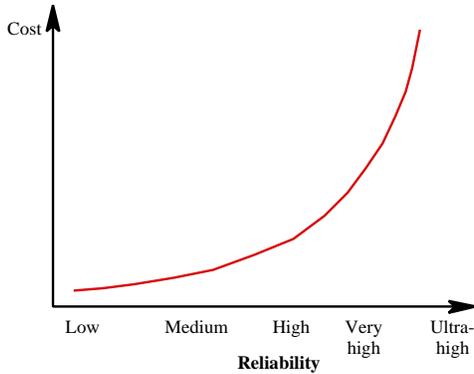
Reliability economics

Because of very high costs of reliability achievement, it may be more cost effective to accept unreliability and pay for failure costs

However, this depends on social and political factors. A reputation for unreliable products may lose future business

Depends on system type - for business systems in particular, modest reliability may be adequate

Costs of increasing reliability



Statistical testing

Testing software for reliability rather than fault detection

Test data selection should follow the predicted usage profile for the software

Measuring the number of errors allows the reliability of the software to be predicted

An acceptable level of reliability should be specified and the software tested and amended until that level of reliability is reached

Statistical testing procedure

Determine operational profile of the software

Generate a set of test data corresponding to this profile

Apply tests, measuring amount of execution time between each failure

After a statistically valid number of tests have been executed, reliability can be measured

Statistical testing difficulties

Uncertainty in the operational profile

This is a particular problem for new systems with no operational history. Less of a problem for replacement systems

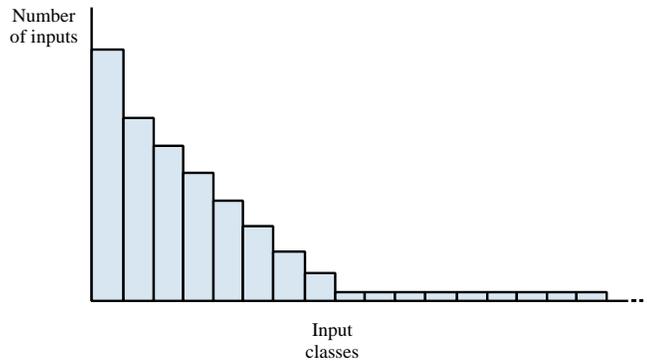
High costs of generating the operational profile

Costs are very dependent on what usage information is collected by the organization which requires the profile

Statistical uncertainty when high reliability is specified

Difficult to estimate level of confidence in operational profile
Usage pattern of software may change with time

An operational profile



Operational profile generation

Should be generated automatically whenever possible

Automatic profile generation is difficult for interactive systems

May be straightforward for 'normal' inputs but it is difficult to predict 'unlikely' inputs and to create test data for them

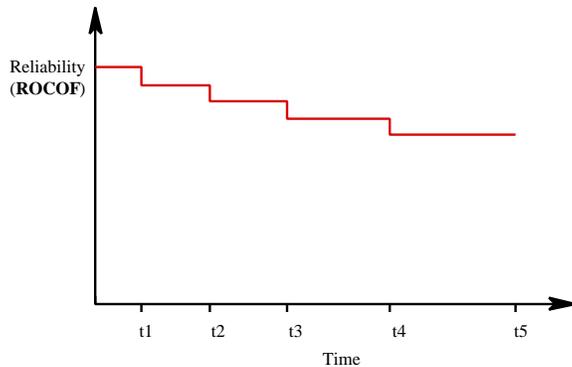
Reliability growth modeling

Growth model is a mathematical model of the system reliability change as it is tested and faults are removed

Used as a means of reliability prediction by extrapolating from current data

Depends on the use of statistical testing to measure the reliability of a system version

Equal-step reliability growth



Observed reliability growth

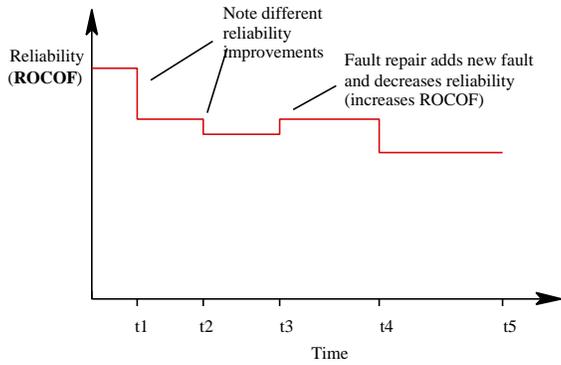
Simple equal-step model but does not reflect reality

Reliability does not necessarily increase with change as the change can introduce new faults

The rate of reliability growth tends to slow down with time as frequently occurring faults are discovered and removed from the software

A random-growth model may be more accurate

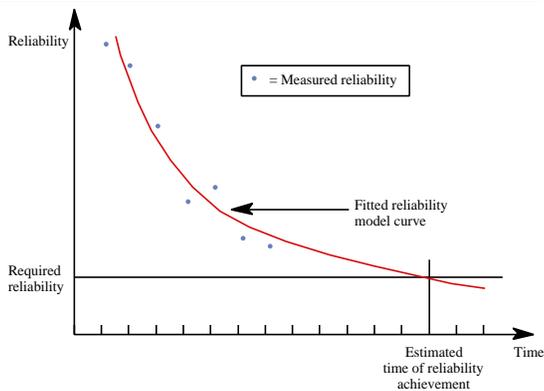
Random-step reliability growth



Growth models choice

- Many different reliability growth models have been proposed
- No universally applicable growth model
- Reliability should be measured and observed data should be fitted to several models
- Best-fit model should be used for reliability prediction

Reliability prediction



Key points

Reliability is usually the most important dynamic software characteristic

Professionals should aim to produce reliable software

Reliability depends on the pattern of usage of the software. Faulty software can be reliable

Reliability requirements should be defined quantitatively whenever possible

Key points

There are many different reliability metrics. The metric chosen should reflect the type of system and the application domain

Statistical testing is used for reliability assessment. Depends on using a test data set which reflects the use of the software

Reliability growth models may be used to predict when a required level of reliability will be achieved
