Software Reliability

Measuring Software Reliability © D L BIRD 2003

Abstract

This paper sets out a technique for measuring software reliability. It introduces a new style of metric that establishes software 'reliability-confidence', a simple measure that can be applied to any software system to give both Customer and Supplier an easy to understand means of qualitatively expressing software reliability. The technique is especially pertinent to the use of Commercial Off the Shelf (COTS) products whose build quality and configuration management status cannot be easily ascertained, a growing requirement!

Introduction

1. Software is not traditionally associated with conventional reliability metrics, therefore choosing an approach that enables meaningful measures of software reliability can be a useful technique, especially when suppliers will not contract for specific software reliability levels.

Aim

2. The primary aim of this software metric is to prevent defective software intensive systems passing into customer service whilst obtaining confidence in the delivered reliability levels. To achieve this, both customers need assurance from their suppliers that software meets the system requirement for reliability.

3. An additional aim is to change the way in-which software reliability is presented to the Customer whilst also enabling the Supplier to balance design margins against a confidence of achieving acceptable reliability.

Background

4. Experience of conventional software metrics is that they do-not demonstrate or convey to the Customer an understanding of delivered software reliability levels and the Customer does not necessarily appreciate or realise that price or time constraints can be reflected in the software quality levels. When the software enters customer service, these constraints are likely to be of no concern to software users, who in-turn can develop often-negative views of system reliability levels which may or may not be justified.

5. There is a degree of consensus that software is 100% reliable if programme execution follows the design intent; the opposite is also generally true, in that any failure to translate or incorporate a requirement is interpreted as an error. It is this latter interpretation which gives the perception of low reliability rather than a design shortfall or lack of requirement definition passed to the supplier.

6. Where the operation of software is subject to random variations or failures, confidence is needed at delivery that reliability has been achieved. Testing every program state and every input condition is time consuming and not cost effective, so qualitative approaches that establish confidence in the software provide an acceptable alternative to traditional demonstration or verification methods.

7. There is also general consensus that reliable software will emerge from the use of assurance methods. For example, the use of quality 'procedural' methods such as

ISO9001 and TickIT certification and the use of mature development 'processes' which can be assured through the use of Capability Maturity Models.

8. Finally the use of a metric for reliability provides the overall product assurance. There is agreement that procedural and process controls will be covered by a Suppliers quality and software management plans. The metric for reliability performance generally is either deterministic[1] or probabilistic[2], and the following proposal sets out a method that moves towards qualitative measures of software reliability - i.e. to have confidence that deterministic errors have been fixed and that probabilistic errors are within agreed limits.

Requirement

9. To identify an appropriate reliability statement that can be used at all levels of the supply chain with an aim of engendering a qualitative understanding of software reliability. Specifically customer needs are for:

- Software dependability - confidence that during and after the Contract, the reliability of software has been built-in and not just proven at acceptance.
- Software with an agreed or acceptable reliability for a known distribution of input and system states.

10. Our principle requirement is to achieve a strong customer focus i.e. 'Software that performs to meet the Customer requirements'.

Baseline Understanding and Definitions

12. Software errors may exhibit themselves as either deterministic (repeatable) or probabilistic (random) in nature. A customer should expect the supplier to remove all or the majority of deterministic errors during Factory Acceptance Test (FAT) or pre-delivery inspections.

13. Probabilistic errors resulting from uncertain or varying input states, or outputs that are subject to error, are most likely to arise from:

- Random or ignorant inputs
- Unknown or untested inputs
- Concurrent and uncoordinated processing

14. Software reliability should be addressed at all stages of the product life-cycle. In general there are three basic aspects; Processes, Procedures and Metrics, covering the requirement for quality, software development activities, the code itself, test plans and the final testing. These aspects are covered thus:

a) *Quality assurance processes provide evidence that 'the process is correct*'
   - ISO9001 accredited and certified suppliers (including sub-systems). The PCO Software Management Plan recommends Sub-System Suppliers who are TickIT Certified
   - Say what your going to do, do it and show you have done it

b) *Procedural methodologies are process driven*

- Use of business capability models such as the Capability Maturity Model (CMM) to provide a reasonably objective assessment of process application and maturity.
- Software questionnaire to measure potential supplier conformance to nature processes and to **audit** the declared processes

c) *Metrics that measure the software reliability during the performance of the Contract and In-service.*

15. Software 'Errors' are an omission or coding mistake that results in a 'Fault'

16. Software 'Faults' are defined as a defect in the code that may cause a 'Failure'.

17. Software 'Failures' are defined as some behaviour of the system that does not meet the Customers requirement. A failure is also considered to be the observable failure of an available function; it is not only a system crash or freeze.

18. Other definitions comprise 'Any deviation from specified program behaviour is a failure' or 'any deviation from required, specified or expected behaviour'.

19. Failure-intensity is defined as the number of failures per unit of time. This is a Customer orientated measurement.

20. Failure-density is defined as the number of failures per software modules or lines-of-code. This is a traditional developer measurement.

21. Repeated failure reports will be ignored with this metric to prevent skewing of the results. The assumption and philosophy is that Users will be made aware of all known software errors at delivery which should help to reduce repeat errors. The process assumes that the Customer/User will not attempt to skew the results with spurious or malevolent reports.

22. Software can be measured in terms of 'degrees of excellence' (how well it meets the customer expectations) or 'degrees of conformance' (how well it meets a requirement and standard), in this context 'degrees of excellence' is the preferred viewpoint.

23. Software reliability is considered to be ‘the probability that software will perform a required function under known conditions for a stated time period’. The time period will be measured in hours of operational use.

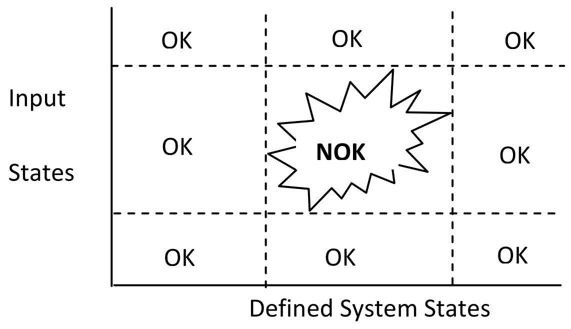
Software Reliability Proposal

23. It is proposed that Customers and Suppliers should move from a quantitative to qualitative approach to defining system reliability using a metric that to specify a confidence of achieving system reliability. It is based on a functional view of systems to which the measure is applied, initially at system levels then at major or ‘super-system’ levels.

24. The precursor to metric application is the determination of functional areas that have the greatest effect on system reliability. This provides a qualitative view of where development and test resources should be directed to achieve maximum growth of confidence. An example of the metric is to 'achieve a 90% confidence of achieving 95% reliability', but note it is not an absolute reliability outcome.

25. The following diagram illustrates that software with known or deterministic errors can be acceptable providing the distribution of equipment states and inputs is known and declared to the Customer. The functional elements of the metric, articulates the

delivered reliability-confidence by describing system attributes and then ranking them in importance. Errors not seen at acceptance are treated as probabilistic errors that may or may not emerge during a normal mission. However, in both cases we would expect these errors to be fixed during the warranty period and to be within the next software up-issue. When probabilistic errors occur, they are likely to change status to deterministic if they can be repeated.



OK = Error free System operation

NOK = Not OK, system exhibits errors

When states and inputs are aligned

26. The assumption associated with this metric is that Customers should be aware of known issues with delivered software. In the example state diagram shown above, the Customer should be made aware of the conditions that will create a system error. A declaration of known system problems should increase Customer awareness of system limitations, so doing so should also alleviate negative criticism and reduce repeat error reports. For this premise to have credibility, the Customer must be assured that any known errors will be fixed within a realistic time period.

The 'Software Confidence in Reliability' Metric

27. This software reliability metric has been selected specifically to provide the end User and Customer with a simple method of determining software reliability whilst appreciating the confidence of that measure. Here reliability is considered to be ‘the probability that software will perform a required function under known conditions for a stated time period’. The time period will be measured in hours of operational running time.

28. The following metric is based on the quality assurance process known as ‘The Run Time Success Theorem’. This metric has been selected as the required initial levels of reliability are highly likely to be achieved by a software supplier employing a sound software quality assurance process, and as the software faults are identified and repaired, the run time metric can quickly indicate the ‘improved’ reliability.

29. The run time success theorem is normally employed when the shape of the failure distribution is not known or cannot be assumed. It should be noted that several statistical models for estimating software reliability have been developed. Whilst many of these assume faults are distributed at random, this assumption may be invalid because software failure will only occur as a result of executing a specific path in the code with specific inputs.

30. The metric will measure failure intensity of a supplied software system. Failure intensity is the number of failures experienced per-unit time period. A failure is considered to be the observable failure of an available function. (It is not only a system crash or freeze and repeat errors are ignored to reduce skewing of the results).

31. For the run time success theorem to apply, it is essential to employ a unit of time that is meaningful to the user. In this case, 1000 hours has been chosen as it equates quite closely to a month of continuous usage (45days x 24 hrs = 1080 hours).

32. The metric is based on the following equation:

$$Rc = (1-C)^{1/(n\times 459/200)}$$

Where:

Rc	= Reliability confidence (0-1)
C	= Confidence (0-1)
n	= Number of Hours between failures.

(A time-unit is currently 459/200 hours to allow a per-unit baseline referenced to 5 failures per 1000Hrs to be established - see later examples)

Or, rearranged as Hours

$$n = \frac{\log(1-C)}{\log Rc}\times\frac{200}{459}$$

33. When the run time formula is employed for single shot events, *n* equals the number of single shot successes before failure to give the required confidence. In this case *n* will represent the number of Hours before failure. **The required confidence for the user will be 90% or better and the target reliability will be 99.5% at Customer Delivery (CD) improving to 99.9% one year after CD.**

34. The failure intensity expected will therefore be 5 failures maximum per 1000 hours at CD and 1 failure per 1000 hours one year after CD. Table-1 illustrates the minimum run time required to indicate a reliability of 90 to 99% with a confidence of 85 to 95%. This table demonstrates a key feature of the metric in that confidence and reliability values can be calculated after just a few hours without failure.

35. However it should be noted that after 10 hours the User could assume he has 90% confidence that the software is 90% reliable, the User is unlikely to be impressed, as this indicates a potential failure once every 10 hours (i.e. 90% reliable using this metric). Only when the software reaches 99.5 to 99.9% reliable (at 90% confidence) or better will the users consider the software to be reliable. This would require the software to operate for 500 to 2500 hours with just one failure; which does not seem an unreasonable target.

-	-	**Running Time in Hours before failure**			
	Reliability →	***0.90***	**0.99**	**0.995**	**0.999**
Confidence	**0.85**	8	82	165	826
Levels	***0.90***	*10*	*100*	***200***	***1002***
	0.95	12	130	260	1304

Table 1: Shows minimum run time in hours without failure to achieve a given reliability at a given confidence.

-	-	Running Hours before failure					
	Time before Failure →	10	100	500	1000	1500	3000
Confidence Levels	50%	0.9702	0.9970	0.9994	0.9997	0.9998	0.9999
	90%	*0.9045*	*0.9900*	*0.9980*	***0.9990***	***0.9993***	***0.9997***
	99%	0.8182	0.9801	0.9960	0.9980	0.9987	0.9993

Table 2: Shows the reliability assumed, for confidences of 50, 90 and 99% after operating for a range of 10 to 3000 hours with 1 failure being recorded between each time period.

Expected Reliability Growth and Targets at CD and Beyond

35. Customers should expect reliability growth to be demonstrated throughout the test and acceptance phase of the product life-cycle. Leading to CD.

36. The customer should expect to see 5-failures per 1000 hours or 90% confidence of achieving 99.5% reliability and at CD + 1year he should expect to see 1-failure per 1000 hours or 90% confidence of 99.9% reliability.

37. At this stage the customer should not expect to add confidence bounds to the metrics or to embrace hard and stretch targets.

Illustrative Trials and CD Examples

38. The customer has set a requirement for 90% confidence that software will achieve 99.5% confidence at CD.

39. During test and acceptance trials the supplier recorded a failure after 8 hours operation. The metric gives the following indicator:

- $Rc = (1-0.9)^{1/(8\times 459/200)}$

- This yields a Reliability-confidence (Rc) of **88.2%** relative to the expected 90% confidence of 99.5% reliability and a time unit of 459/200 hours. The result is outside the requirement although expected at this stage of the test and integration activity.

40. At follow-on test and acceptance trials, the system operated for a total of 40 hours before the first failure occurred. The metric gives the following indicator:

- $Rc = (1-0.9)^{1/(40\times 459/200)}$

- This yields a Reliability-confidence (Rc) of **97.5%**, which is just outside the requirement although perhaps still to be expected at this stage of the test and integration activity.

41. Leading up to CD the system was operated for a total of 1000 hours before the first failure. The metric gives the following indicator:

- $Rc = (1-0.9)^{1/(1000\times 459/200)}$

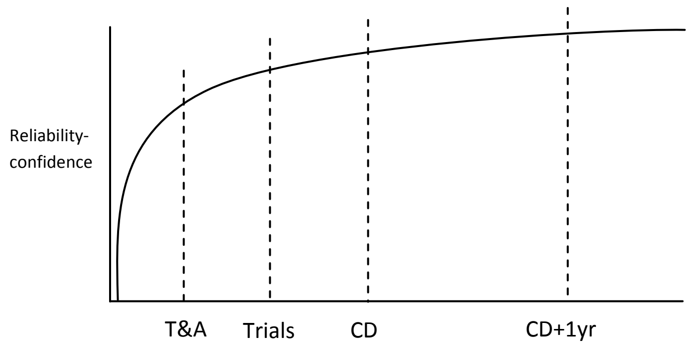
- This yields a Reliability-confidence (Rc) of **99.8%**, which is inside the requirement and the system meets the CD requirement.

42. At CD + 1year the system has been operated for a total of 3000 hours before a failure. The result for this trial using the Reliability-confidence was:

- $Rc = (1-0.9)^{1/(3000 \times 459/200)}$

- This yields a Reliability-confidence (Rc) of **99.9%**, which is inside the requirement and the system continues to provide User confidence and reliability growth.

42. In these examples the growth in reliability-confidence has continued from test and acceptance through to CD and beyond, which should be the customers expectation. The distribution of failures and reliability growth is not likely to be as depicted in these examples, since software changes made through life are expected to change the spread of errors, however reliability growth should continue as runtime increases. Graphically, reliability growth is expected to resemble:



Supply Chain Reliability Measurements

43. As outlined, the customer objective in using this metric should be to measure software reliability from a User viewpoint and ideally the metric would be applied to the supply-chain to form a back-to-back and consistent measurement of software reliability expectations.

44. This proposal is based on black-box testing and takes no account of system complexity. In the latter case complex systems should be tested many times through and conversely low complexity systems tested fewer times to take into account what is expected to be a typical Customer viewpoint.

Illustrative Process Improvements Methods

45. To achieve reliability growth and product confidence levels customers should expect the Suppliers design and development effort is directed towards meeting a customer orientated failure-intensity (failures per hour) rather than the 'traditional' developer orientated fault-density (faults per modules or lines-of-code) measurement.

46. The method concentrates quality or design improvement effort on a Customer view of the expected operational profiles, and ranks these with respect to the important and frequently used functional system elements.

47. The customers expected usage profile is an important consideration during system development and test, since system usage will determine the Customers expected reliability levels.

Conclusion

48. The reliability-confidence method offers an opportunity to use simple formulae to produce a metric that provides a broad range of customer skills, knowledge and experience to gain confidence in the delivered software products through the metric.

49. Using the metric also allows the supply-chain to manage design-margins and deliver only that reliability-confidence needed to meet the requirement. In addition, a qualitative rather than quantitative target for reliability in the supply chain is easier to deliver rather than meet hard contractual commitments.

50. The technique also enables some reduction in system and regression tests, factory facility usage during test and acceptance trials and can reduce the duration of traditional reliability demonstration phases of a project.

51. Delivery of software without verification infers a heavier reliance on modelling or probabilistic acceptance methods, however the approach proposed provides a coherent method of determining the confidence of software reliability and customer in-service quality levels thereby negating that need.

52. The use of reliability-confidence embraces the concept of 'smart' principles by allowing the supplier freedom to balance resources and design margins whilst delivering systems that meet customer expectations.

---

[1] Deterministic errors have outcomes determined by system inputs.

[2] Probabilistic errors can produce several outcomes, with associated probabilities for a fixed input.