Software Reliability Metrics for Military Systems

W. Edward Koss; US Air Force ESD; Bedford

Key Words: Software Development, Software Metrics, Software Reliability Metrics, Software Reliability.

1. Introduction

a. This paper addresses what is by far the most critical component in today's ground based military command and control systems but a component which has rarely been understood or managed, the area of software reliability. Software reliability has been so overlooked that there are instances as late as 1986-87 where major command and control systems were three to four years into their development cycle, and well past the software critical design reviews, and not one assessment of software with critical defense software in order to mature availability. This paper will discuss those
cases and where management and technical metrics for software reliability which must be assessed continually as the development progresses. Efforts should be taken immediately to begin these assessments on all projects with critical defense software in order to mature software reliability metrics and make these metrics an integral part of future systems development. This must be done if estimates of software reliability and system reliability can ever be taken seriously.

b. Computer software is at the core of today's defense systems. Virtually, all major military systems are essentially dependent on the correct operation of that defense systems' software. The B1-A bomber of ten years ago embodied .5 million lines of code, fewer than half of the 1.2 million lines of code in today's B-1 B. Today the F-16 has seven flight computers and 135,000 lines of code. Similar software dependencies are likewise evident in space systems and command and control systems. In fact, command and control systems are probably the most software dependent of all military systems. For example, the Joint Surveillance Target Attack Radar System (JSTARS) expects to have 1.7 million lines of code on board their mission airplanes. The World Wide Military Information System (WIS) is expected to field some 5 million lines of software for their high availability command & control system.

c. In the late 1980's it has been common for the top level system specifications for ground based military command and control systems to specify system level operational availabilities in the order .997 and .998. While these levels may be achievable, much of the rationale for these availability levels come from the dramatic advances in commercial computer hardware reliability growth curves. Since all present military command and control centers rely very heavily on this commercial computer hardware base it is logical to equate these achieved computer hardware reliabilities and availabilities with the ability to deliver total systems with high operational reliabilities and availabilities. Just as important, it is expected that these centers will have resultant much lower life cycle maintenance costs. It is important to remember that system availability and system MTBF are related. You can achieve high system availability with low MTBF's but at the expense of very high equipment sparing requirements, software maintenance costs, supply levels, and life cycle costs.

d. However, while the ability to deliver reliable computer hardware can be considered to be a given, in today's environment, it is the ability to deliver the software component of the system which will determine the extent to which the total system meets its operational availability and this has not been analyzed nor factored into the overall system operational availability equations in the development of current military systems.

ea. To illustrate the nature of the challenge between hardware and software reliability in command/control systems, in January 1986 the Reliability Plus Inc. measures show MTBF's for IBM central processors (3080 class machines) to be between 20,000 and 30,000 hours, a very substantial reliability base. In the fall of 1986 these same machines had demonstrated MTBF's of over 80,000 hours, a very substantial reliability growth. Newly developed software has an average of only 160 hours to 200 hours between faults although the vast majority (90% plus) of these software faults were not considered fatal. Nonetheless the void is there and must be actively managed to be improved.

2. Status of Software Reliability

a. The statistics of software reliability are in the formative stages right now. While this paper will discuss some software reliability metrics a software development should continually assess, the key is to also assess the stability and credibility of these reliability measures. The confidence one has that these software reliability metrics are valid is related to the insight and mastery the software developer has with:

(1) the type of software being developed (business, spaceflight, scientific, command and control, etc.),
(2) the past history the developer has with the collection of software reliability parameters,
(3) the stability of that developers software reliability data base,
(4) the precision and accuracy of the software specifications,
(5) the track record of the developer in delivering quality specifications.

If any of these basic factors is not fully developed then it is unlikely that a good evaluation of the reliability of a current software development can be made.

b. Items 4 and 5 of the above list are particularly important. In the end, the reliability of software is directly tied to the adequacy, correctness, precision and accuracy of the software specification. However this paper will deal only with software reliability metrics, as the subject of software reliability and its relationship to the software specifications has been discussed rather extensively in recent R&M Symposium papers (1986).

c. The ability to deliver quality specifications is rarely achieved within the defense industry. In the vast majority of the programs I have had direct experience with over the last five years (10 defense programs) approximately 9% of the time those programs completed only 15% to 30% of total work required of the specification during the scheduled time. In rare cases up to 85% of work required was completed, but those instances are quite rare. Other factors central

U.S. Government work not protected by U.S. Copyright.
to software reliability also vary across wide ranges on software programs. Skill levels (6 mos. to 10 years) and quality code production (50 to 250 $/LOC/mo.) are examples of the wide variances.

d. As for theoretical reliability models based upon program size, they have been constructed, however the practical utility of the resulting reliability values is difficult to assess. Error generation, upon program size, they have been constructed, however the discovery and correction of errors is still a function of human software engineering skills. A further difficulty in current software reliability modeling is the fact that errors can originate in the specification, the design, and the coding. Hardware failures are usually a function of load, strength, and time. Whether a weakness is due to the specification, design, or the production process, the physics of the failures remain the same. With software however, specification, design and coding errors are often different in nature, and the probability of their existence depends upon different factors. Coding errors may be related to the number of coding statements but the number of specification errors could easily multiply the total number of software errors regardless of the total size of the code produced. Since software reliability is currently so dependent upon human performance and non-physical factors, the use of reliability parameters will probably be credible only if the software developer has a unique mastery of the application, the development environment and a stable history of collecting and tracking software reliability metrics for their particular software development environment. It is an area which must be judged on a case by case basis with each developer being judged on their own investment and analysis of the reliability metrics and what they have contributed to the final software product.

e. The nature of this characteristic in software development can be concisely illustrated with the following example. Consider a program which reads three positive numbers and classifies them according to whether or not they can form the sides of a triangle (isosceles, scalene and equilateral). Two variants of this program which appear in the literature use essentially the same algorithm for classifying the triangles, that is counting the number of sides which are equal. However, in one, the algorithm is employed before a decision is made as to whether a triangle of any type can be formed and, in the other, the algorithm is used afterwards. The difference in the programs caused by this minor difference in strategy is dramatic; whereas one program has the 76 paths the other has 88. The optimal would have five. Clearly, the problems of testing these program variants are different because the number of significant program paths are different. Yet the problems which they solve are the same. Clearly the impact on software reliability is great given the path differences.

3. Software Errors and Software Failure Rates

a. Any discussion of software reliability metrics must start with the core of the issue, fielded (operational) software error counts and the rate at which they occur, that is the software failure rate. The extent to which the projection or assessment of failure rates in fielded software is credible, the ability to assess reliability is greatly enhanced. As has been highlighted earlier in this paper, a credible failure rate is very difficult to establish and can only be established when an investment in a systematic gathering and analysis of software reliability data is made. However, this task is far from easy, its difficulty is reflected in the fact that software reliability is traditionally assumed to be 1.0 for most military applications since supportable reliability numbers for software are so unstable. The following paragraphs provide a basic discussion on the subject.

b. The probability of failure over a specified period of time is given by the failure rate, the expected (average) number of failures per unit time, usually taken as a computer hour. The primary metric for software reliability is the failure rate and this is what software developers must ultimately assess. An associated metric is the software error count and the estimate of how often those errors will be found (fault density). Software failure rates generally depend on characteristics of:

1. the execution environment (MIPS, critical functions, type of application)
2. cyclic dependencies,
3. variability of data,
4. execution frequency.

Because these areas may not be well specified during the early stages of a software development, software error counts and detection rates are often used. As the key metrics this is true because prediction of fault density does not require explicit knowledge of the execution environment and thus can assist in reliability assessments early in a development program although heavy reliance on these type assessments is not recommended. They are good as preliminary indications but must be backed up by the software developers history of software reliability measurement, analysis, and track record.

b. A final note on software reliability, all other things being stable, a program continuously executing on a fast computer will experience a higher failure rate than the same program executing on a slower machine. The failure of a program is dependent on the number of executions and variability of data rather than any decay from passage of time. A program running on a computer twice as fast will have twice the failure rate as the same program running on a machine rated at one half the speed. Thus to assess failure rate in the operational environment the software must be tested on the actual computer it will run on.

c. Error Rates per 1000 Source lines (KSLOC)


c. Error Rates per 1000 Source lines of Code (KSLOC)


c. Error Rate: 65 to 85 errors per KSLOC during development. Approximately 75% of these errors would be removed during development/test thus some 16 to 21
errors per KSLOC will remain after fielding.


Their prediction is based on the following equation:

\[ \frac{B}{P} = \frac{A_1}{1 + a_1 P} + a_2 \ln P \]

\[ L = \text{average number of lines of executable source code} \]

\[ P = \text{number of lines of executable source code} \]

\[ A_1 = 22.5 \]

\[ a_1 = 7 \]

From this analysis they estimate that there will be 7.0 for all cases from this source.

If \( P = 1000 \), then \( B/P = 0.225 \), \( B = 22.5 \), \( K = 7.0 \).

From this analysis they estimate that there will be 22.5 faulty per KSLOC in the operational software delivered.


One of the more “optimistic” estimates of errors generated and errors found prior to delivery. They estimate that only 28 errors per KSLOC will be generated in the development cycle and that essentially only one error per KSLOC will be present in the delivered code.

(4) Source: Jones, T.C., IBM Systems Journal 17, No.1., 1982, pp 39-63

Overall error rate 30-35/KSLOC during development.

Report assumptions approximately 20% of the errors at fielding, thus an error rate of 7 per KSLOC in the delivered software system.


In 40 projects at NASA and RADC the average error rate was 1 error per 1000 lines of code. The error range was tremendous. Four projects came in at one error per 1000 lines of source code; three came in at 100 or more errors per 1000 lines and one program of the 40 came in with 150 errors per 1000 lines of code.

So from these sources, and they being far from an exhaustive list, one can see the estimates of the range of error rates a software project can carry into the operational stage. The estimated ranges go from one error per KSLOC to a high, from actual measures of errors, of 150 errors per KSLOC and rates are also documented for the ranges in between these figures. The bottom line is software error rates are quite unstable and so much depends on the people involved and a host of other difficult to repeat factors.

5. Fatal Errors. Error Detection Rates and Mean Time to Repair

a. For the assessment of software reliability, we need more than software errors per lines of code. We need assumptions and support for fatal errors, error detection rates, an estimate of Mean Time To Repair (MTTR) and Administrative Logistic Delay Time (ALDT) for an operational environment. These are also documented for the ranges in between these figures. The bottom line is software error rates are quite unstable and so much depends on the people involved and a host of other difficult to repeat factors.

b. As with the previous parameter, an approach to error detection rates is likewise fairly unstable and not conclusively supported by the research to date. For the purposes of this analysis we will assume that a range between .13 and .25 errors per KSLOC are detected and corrected per year. This source cites the error detection as for COTS software developments. Since COTS software is generally more stable, after a beta site testing program, than might be expected from straight defense development software introduced into an operational environment. Another factor here is the ability to “stress” the software. COTS software can be stressed and “field tested” quickly across hundreds and thousands of diverse sites where a military type development will have a very small, more homogeneous base and thus some errors can remain for some time without being detected.

c. The question at this point in this discussion, how do these metrics interrelate and contribute to software reliability/availability assessments? This section will address the most straightforward method for assessment of where the software reliability is on the development curve and how it is used to assess overall system reliability/availability. It is these general analyses software developers must begin early in the system development process to adequately assess what the system can be expected to do in the operational environment. With today’s reliance on software for system mission capability it is no longer adequate to assume the hardware reliability component of a system can be assumed to represent the total system reliability number. The following examples are straightforward and do not include assumptions for the critical software reliability elements of execution environment, functional cyclic dependencies, variability of data and execution frequency. To include those would be too cumbersome to present, a concise discussion that the central points of this paper would be lost. Such factors can be quite complex and are very application dependent. The examples here can illustrate the challenge of software reliability assessments without going to this level of detail.

b. In looking at the very broad range of fielded errors possible per 1000 source lines of code, presented earlier in this paper and factoring those into an assumed system availability requirement of say .997, the following sensitivity cases illustrate a balance between what can reasonably be expected from military software with the current technology. The examples address some of the evolving key parameters, make some reasonable assumptions based on some current military developments and software reliability research, and thus show how software reliability factors can be bounded to meet the system reliability requirement. In an actual program, continuing adjustments and analyses would be necessary from the point of the software development plan forward to final testing and installation. To achieve software reliability will require an early and constant investment in technical management factors.
hardware suite is composed of commercial-off-the-shelf
hardware system availability is assumed to be .9992.
To get to that availability it is assumed the computer
software reliability will be presented in the
lines of code constitute the critical software. The
source lines of code at operational fielding. These
I am assuming a software development having 227,000
systems and that there are backup processors and
computer subsystems. The .9992 is realistic and
rate, MTTR
configured to achieve. The error rate, fatal error
reflects what several complex systems have been
discussed in this paper. These calculations do not
add one to two hours downtime per year for the first
one to two years of the system.

(1) Case 1
Error Rate = 10 per KSLOC fielded
Fatal Errors Rate = 10%
MTTR & ALDT = 10 minutes (warm start)
Error Detection Rate = .13 per KSLOC
Total Software SLOC = 227,000
Total errors fielded at 10 per KSLOC = 2270
MTTR + ALDT = 10 minutes per fatal error
Errors detected first year = 295 errors (.13 x 2270)
Fatal errors (10% of errors detected) = 30

(2) Case 2
Error Rate = 25 per KSLOC fielded
Fatal Errors = 10% of total errors
MTTR + ALDT = 10 minutes
Error Detection Rate = .25 per KSLOC
Total Software SLOC = 227,000
Errors fielded at 25 per KSLOC = 227 x 25 = 5675
MTTR + ALDT = 10 minutes per fatal error
Errors detected first year at .25 per KLOC = .25 x
5675 = 1418
Fatal errors = 142

(3) Case 3
Error Rate = 25 per KSLOC fielded
Fatal Errors = 10%
MTTR + ALDT = 10 minutes
Error Detection Rate = .20 per KSLOC
Block A total developed SLOC = 227,000
Errors fielded at 25 per KSLOC = 227 x 25 = 5675
MTTR + ALDT = 10 minutes per fatal error
Errors detected first year at .20 per KLOC = .20 x
5675 = 1135
Fatal errors = 114

(4) Case 4
Error Rate = 25 per KSLOC fielded
Fatal Errors = 10% of total errors in a
year
MTTR + ALDT = 10 minutes (warm start
only)
Error Detection Rate = .13 per KSLOC
Block A total developed SLOC = 227,000
Errors fielded at 25 per KSLOC = 227 x 25 = 5675
MTTR + ALDT = 10 minutes per fatal error
Errors detected first year at .13 per KLOC = .13 x
5675 = 738
Fatal errors (10% of total errors first year) = 738 x
.1 = 74

\[
\begin{align*}
\text{Total System Availability} & = \text{SWav} \times \text{MTBF} + \text{MTTR + ALDT} \\
& = \frac{118.4}{118.4 + 0.17} = 0.9985 \\
\text{Total System Availability} & = 0.9992 \times 0.9985 = 0.9997
\end{align*}
\]

While these simplified examples illustrate how the
various software reliability parameters interact over
a fairly small range of values, one can see that the
Case 1 parameters could be an upper limit for systems
which have .997 as their system availability
requirement. The key point is that these type of
judgements and analyses are required on a continuous
basis and across all levels during software
development to ensure the software contribution to
overall system availability is constantly managed from
the earliest stages of the development.

7. Summary
a. In developing software reliability metrics for
military systems it is clear that there is a hierarchy
of technical and management activities which must be
done with diligence and attention to detail. Given
the importance of these successive levels and the need
to invest in each of these areas it is useful to
concisely highlight those levels again in this
summary.

b. At the top level there are the following
areas:
(1) Type of software being developed
(business, spaceflight, command and control, process
control etc.)
(2) Developers software reliability metric
collection history
(3) Stability of reliability data base and
practical application
(4) Developers history in developing quality
specifications and assessing reliability

c. At the next level there are the software
requirement specifications and their associated
characteristics of:
(1) Testability
(2) Traceability
(3) Correctness
(4) Clarity
(5) Completeness

d. The next level of metrics begin to address
those technical areas which must be known if
subsequent lower levels of reliability metrics are to
be meaningful. This level encompasses the following
areas:
(1) The execution environment (hardware
processing rates, critical functions, type of
application)
(2) Cyclical functional dependencies
(3) Variability of data
(4) Execution frequency

Being able to bound these technical areas means quite
a lot of detailed knowledge is necessary for adequate
software reliability assessments. Without this level of
insight; a credible reliability assessment would be
very difficult.
e. At the lowest level software reliability metrics involve assessments of:
   (1) Fielded Errors per 1000 Source Lines of Code
   (2) Fatal Error Rates
   (3) Mean Time to Repair
   (4) Error Detection Rate
   (5) Software Mean Time Between Failures

f. To make this entire process credible it is critical a software developer invest in the collection of these metrics within their own development environment. Software reliability metrics will continue to be uniquely identified with a particular developer and their associated development and management methodology since the software development process will continue to reflect the particular individuals who make up the development team. If software reliability is to be meaningful these metrics must be applied against actual military software developments. Today military systems specify operational availability, requirements and reliability requirements however there must be more substance behind meeting these requirements. To do that the military and industry must invest in practical software reliability metrics collection and application. In today’s environment and in tomorrow’s environment software reliability is system reliability since the computer hardware technology can reach such a high state of reliability with relatively small investments. That is the point to remember and software developers must invest in software reliability metrics to reflect this reality.

References
6. IBM 3090 Figures less Than Impressive, Information Week, 7 October 1986, p. 18
11. IBM/Intermetrics, Analysis of Control Center Software, System Design Review Activity, 26 September 1986, p. 10
13. Smith, J., “WIS Development Progressing”, Aviation Week, 23 Feb 1987, p. 27
16. WIS SAF PAR Briefing, ESD, Bedford, MA., July 1986

Biography
Dr. W. Edward Koss
2000 M Street, Suite 650
Washington, DC 20036

Dr. Koss is finishing his tour at Electronic Systems Division (ESD), Bedford, MA., as the System Program Director for Granite Sentry. At ESD he has also served as the Deputy Program Manager for Logistics on the WIS program. Dr. Koss has extensive space systems experience as the Director, Space Computer Resources, Air Force Space Division 1981-85 and on two other assignments at Space Division. He was involved in the Anti-Satellite Program, Milstar, and the missions associated with the first Atlantis space flight (1985). He currently serves in additional duties as technical consultant/expert witness for IRS court cases on computer leases and as expert witness for the Air Force Judge Advocate. He has past experience in military intelligence and has extensive international experience; having traveled to 73 countries, the Arctic, and the Antarctic. Lt Col Koss has a B.S. in Mathematics, MBA in Business, and a PhD in Finance with post-doctorate studies at Columbia University. He has published 63 articles and five books on the subjects of Computer Management, International Affairs, and Finance.