A Recommended Practice for Software Reliability

Dr. Norman F. Schneidewind
Naval Postgraduate School

This article reports on the revisions to the American Institute of Aeronautics and Astronautics' (AIAA) publication "AIAA Recommended Practice for Software Reliability (R-013-1992)" [1]. Sponsored by the AIAA and the Institute of Electrical and Electronics Engineers, the revision addresses reliability prediction through all phases of the software life cycle, since identifying errors early reduces the cost of error correction. Furthermore, there have been advances in modeling and predicting the reliability of networks and distributed systems that are included in the revision.

Software reliability engineering (SRE) is a discipline that can help organizations improve the reliability of products and processes. The American Institute of Aeronautics and Astronautics (AIAA) defines SRE as,

> The application of statistical techniques to data collected during system development and operation to specify, predict, estimate, and assess the reliability of software-based systems. [1]

This recommended practice [1] is a composite of models, tools, and databases, and describes the *what and how* details of SRE, predicting the reliability of software. It provides information necessary for the application of software reliability measurement to a project, lays a foundation for building consistent methods, and establishes the basic principles for collecting the performance data needed to assess software reliability. The document describes how any user may participate in ongoing, software reliability assessments or conduct site- or package-specific studies.

It is important for an organization to have a disciplined process if it is to produce highly reliable software. This article describes the AIAA's recommended practice and how it is enhanced to include the risk to reliability due to requirements changes. A requirements change may induce ambiguity and uncertainty in the development process that cause errors in implementing the changes. Subsequently, these errors propagate through later phases of development and maintenance, possibly resulting in significant risks associated with implementing the requirements. For example, reliability risk (i.e., risk of faults and failures induced by changes in requirements) may be incurred by deficiencies in the process (e.g., lack of precision in requirements).

A revision of the "AIAA Recommended Practice for Software Reliability (R-013-1992)," sponsored by AIAA and the Institute of Electrical and Electronics Engineers, will address reliability prediction through all phases of the software life cycle since identifying errors early reduces the cost of error correction. It will also examine recent advances in modeling and predicting the reliability of networks and distributed systems. At this time, it is not known when this revision will be released. The following sections taken from [1] provide an overview of the planned revisions.

Purpose

The "AIAA Recommended Practice for Software Reliability (R-013-1992)" is used from the start of the requirements phase through the operational-use phase of the software life cycle. It also provides input to the planning process for reliability management.

The practice describes activities and qualities of a software reliability estimation and prediction program. It details a framework that permits risk assessment and predicting software failure rates, recommends a set of models for software reliability estimation and prediction, and specifies mandatory as well as recommended data collection requirements.

The AIAA practice provides a foundation for practitioners and researchers. It supports the need of software practitioners who are confronted with inconsistent methods and varying terminology for reliability estimation and prediction, as well as a plethora of models and data collection methods. It supports researchers by defining common terms, by identifying criteria for model comparison, and by identifying open research problems in the field.

Intended Audience and Benefits

Practitioners (e.g., software developers, software acquisition personnel, technical managers, and quality and reliability personnel) and researchers can use the AIAA practice. Its purpose is to provide a common baseline for discussion and to define a procedure for assessing software reliability. It is assumed that users of this recommended practice have a basic understanding of the software life cycle and statistical concepts.

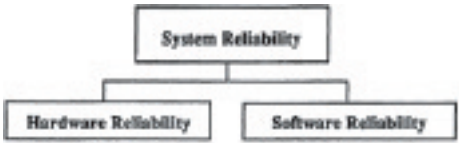
This recommended practice is intended to support designing, developing, and testing software. This includes software quality and software reliability activities. It also serves as a reference for research on the subject of software reliability. It is applicable to in-house, commercial, and third-party software projects and has been developed to support a systems reliability approach. As illustrated in Figure 1, the AIAA practice considers hardware and, ultimately, systems characteristics.

SRE Applications

Industry practitioners have successfully applied SRE to software projects to do the following [2, 3, 4, 5, 6]:

- Indicate whether a specific, previously applied software process is likely to produce code that satisfies a given software reliability requirement.
- Determine the size and complexity of a software maintenance effort by predicting the software failure rate during the operational phase.
- Provide metrics for process improvement evaluation.
- Assist software safety certification.
- Determine when to release a software system or to stop testing it.
- Predict the occurrence of the next failure for a software system.
- Identify elements in software systems that are leading candidates for redesign to improve reliability.
- Estimate the reliability of a software system in operation using this information to control change to the system.

Figure 1: *System Reliability Characteristics*



Terminology [1]

Software Quality: (1) The totality of features and characteristics of a software product that bear on its ability to satisfy given needs; for example, to conform to specifications. (2) The degree to which software possesses a desired combination of attributes. (3) The degree to which a customer or user perceives that software meets his or her composite expectations. (4) The composite characteristics of software that determine the degree to which the software in use will meet the customer's expectations.

Software Reliability: (1) The probability that software will not cause the failure of a system for a specified time under specified conditions. The probability is a function of the inputs to and use of the system, as well as a function of the existence of faults in the software. The inputs to the system determine whether existing faults, if any, are encountered. (2) The ability of a program to perform a required function under stated conditions for a stated period of time.

Software Reliability Engineering: The application of statistical techniques to data collected during system development and operation to specify, predict, estimate, and assess the reliability of software-based systems.

Software Reliability Estimation: The application of statistical techniques to observed failure data collected during system testing and operation to assess the reliability of the software.

Software Reliability Model: A mathematical expression that specifies the general form of the software failure process as a function of factors such as fault introduction, fault removal, and the operational environment.

Software Reliability Prediction: A forecast of the reliability of the software based on parameters associated with the software product and its development environment.

The AIAA practice enables software practitioners to make similar determinations for their particular software systems as needed. Special attention should be given in applying this practice to avoid violating the assumptions inherent in modeling techniques. Data acquisition procedures and model selection criteria are provided and discussed to assist in these efforts.

Relationship to Hardware and System Reliability

Hardware Reliability

There are at least two significant differences between software reliability and hardware reliability. First, software does not fatigue, wear out, or burn out. Second, due to the accessibility of software instructions within computer memories, any line of code can contain a fault that, upon execution, is capable of producing a failure. A software reliability model specifies the general form of the dependence of the failure process on the principal factors that affect it: fault introduction, fault removal, and the operational environment.

The failure rate (failures per unit time) of a software system is generally decreasing due to fault identification and removal. At a particular time, it is possible to observe a history of the failure rate of the software. Software reliability modeling is done to estimate the form of the curve of the failure rate by statistically estimating the parameters associated with the selected model. The purpose of this measure is twofold: (1) to estimate the extra execution time required to meet a specified reliability objective, and (2) to identify the expected reliability of the software when the product is released. This procedure is important for cost estimation, resource planning, schedule validation, and quality prediction for software maintenance management.

The creation of software and hardware products is the same in many ways and can be similarly managed throughout design and development. However, while the management techniques may be similar, there are genuine differences between hardware and software. The following are examples:

- Changes to hardware require a series of important and time-consuming steps: capital equipment acquisition, component procurement, fabrication, assembly, inspection, test, and documentation. Changing software is frequently more feasible (although effects of the changes are not always clear) and oftentimes requires only code, testing, and documentation.
- Software has no physical existence. It includes data as well as logic. Any item in a file can be a source of failure.
- Software does not wear out. Furthermore, failures attributable to software faults come without advance warning and often provide no indication they have occurred. Hardware, on the other hand, often provides a period of graceful degradation.
- Software may be more complex than hardware, although exact software copies can be produced, whereas manufacturing limitations affect hardware.
- Repair generally restores hardware to its previous state. Correction of a software fault always changes the software to a new state.
- Redundancy and fault tolerance for hardware are common practice. These concepts are only beginning to be practiced in software.
- Software developments have traditionally made little use of existing components. Hardware is manufactured with standard parts.
- Hardware reliability is expressed in wall clock time. Software reliability is expressed in execution time.
- A high rate of software change can be detrimental to software reliability.

Despite the above differences, hardware and software reliability must be managed as an *integrated* system attribute. However, these differences must be acknowledged and accommodated by the techniques applied to each of these two types of subsystems in reliability analyses.

System Reliability

When integrating software reliability with the system it supports, the characterization of the operational environment is important. The operational environment has three aspects: (1) system configuration, (2) system evolution, and (3) system operational profile.

System configuration is the arrangement of the system's components. Software-based systems are just that; they cannot be pure but must include hardware as well as software components. Distributed systems are a type of system configuration. The purpose of determining the system configuration is twofold:

- To determine how to allocate system reliability to component reliabilities.
- To determine how to combine component reliabilities to establish system reliability.

In modeling software reliability, it is necessary to recognize that systems frequently evolve as they are tested. That is, new code or even new components are added. Special techniques for dealing with evolution are provided in [7].

The system's operational profile characterizes in quantitative fashion how the software will be used. It lists all operations

realized by the software and the probability of occurrence and criticality of each operation.

A system may have multiple operational profiles or operating modes, which usually represent difference in function associated with significant environmental variables. For example, a space vehicle may have ascent, on-orbit, and descent operating modes. Operating modes may be related to time, installation location, customer, or market segment. Reliability can be tracked separately for different modes if they are significant. The only limitation is the extra data collection and cost involved.

Software Reliability Modeling

Software is a complex intellectual product. Inevitably, some errors are made during requirements formulation as well as during designing, coding, and testing the product. The development process for high-quality software includes measures that are intended to discover and correct faults resulting from these errors, including reviews, audits, screening by language-dependent tools, and several levels of test. Managing these errors involves describing, classifying, and modeling the effects of the remaining faults in the delivered product and thereby helping to reduce their number and criticality.

Dealing with faults costs money and impacts development schedules and system performance (through increased use of computer resources such as memory, CPU time, and peripherals requirements). There can be too much as well as too little effort spent dealing with faults. Thus the system engineer (along with management) can use reliability estimation and prediction to understand the current system status and make trade-off decisions.

Prediction Model Validity

In prediction models, validity depends on the availability of operational or test failure data [4]. The premise of most estimation models is that the failure rate is a direct function of the number of faults in the program, and that the failure rate will be reduced (reliability will be increased) as faults are detected and eliminated during test or operations. This premise is reasonable for the typical test environment, and it has been shown to give credible results when correctly applied [3, 5, 6]. However, the results of prediction models will be adversely affected by the following:

- Change in failure criteria.
- Significant changes in the code under test.
- Significant changes in the computing environment.

All of these factors will require a new set of reliability model parameters to be computed. Until these can be established, the effectiveness of the model will be impaired. Estimation of new parameters depends on the measurement of several execution time intervals between failures.

Major changes can occur with respect to several of the above factors when software becomes operational. In the operational environment, the failure rate is a function of the fault content of the program, of the variability of input and computer states, and of software maintenance policies. The latter two factors are under management control and are frequently utilized to achieve an expected or desired range of values for the failure rate or the downtime due to software causes. Examples of management action that decrease the failure rate include avoidance of data combinations that have caused previous failures, and avoidance of high workloads.

Software in the operational environment may not exhibit the reduction in failure rate with execution time that is an implicit assumption in most estimation models. Knowledge of the management policies is therefore essential in selecting a software reliability estimation procedure for the operational environment. Thus, the estimation of operational reliability from data obtained during test may not hold true during operations.

Life-Cycle Approach

A key part of the revision will be the life-cycle approach to SRE. The following example illustrates the life-cycle approach to reliability risk management of the revised recommended practice: This approach has been demonstrated on the space shuttle avionics software [2, 3].

AIAA Practice Applied to the Space Shuttle

The space shuttle avionics software represents a successful integration of many of the computer industry's most advanced software engineering practices and approaches. Since its beginning in the late 1970s, this software development and maintenance project has evolved one of the world's most mature software processes applying the principles of the highest levels of the Software Engineering Institute's Capability Maturity Model®, trusted software methodology, ISO 9001 standards, and [1].

This software process, considered a *best practice* by many software industry organizations, includes state-of-the-practice software reliability engineering methodologies. Life-critical shuttle avionics software produced by this process is recognized to be among the highest quality and highest reliability software in operation in the world. This case study explores the successful use of extremely detailed fault and failure history, throughout the software life cycle, in the application of SRE techniques to gain insight into the flight worthiness of the software and to suggest *where to look* for remaining defects. The role of software reliability models and failure prediction techniques is examined and explained to apply these approaches on other software projects. One of the most important aspects of such an approach is addressed: *how to use and interpret the results* of the application of such techniques.

Interpretation of Software Reliability Predictions

Successful use of statistical modeling in predicting the reliability of a software system requires a thorough understanding of precisely how the resulting predictions are to be interpreted and applied [5]. The primary avionics software subsystem (PASS) (430,000 lines of code) is frequently modified, at the request of NASA, to add or change capabilities using a constantly improving process. Each of these successive PASS versions constitutes an upgrade to the preceding software version. Each new version of the PASS (designated as an operational increment) contains software code that has been carried forward from each of the previous versions (*previous-version subset*) as well as new code generated for that new version (*new-version subset*). By applying a reliability model independently to the code subsets according to the following rules, you can obtain satisfactory composite predictions for the total version:

1. All new code developed for a particular version does use a nearly constant process.
2. All code introduced for the first time for a particular version does, as an aggregate, build up the same *shelf life* and operational execution history.
3. Unless subsequently changed for a newer capability, thereby becoming new code for a later version, all *new code* is only changed thereafter to correct faults.

It is essential to recognize that this approach requires a very accurate code change-history so that every failure can be uniquely attributed to the version in which the defective line(s) of code was first introduced. In this way, it is possible to build a separate failure history for the new code in each release. To apply SRE to your soft-

ware system, you should consider breaking your systems and processes down into smaller elements to which a reliability model can be more accurately applied. Using this approach, the Naval Postgraduate School has been successful in applying SRE to predict the reliability of the PASS for NASA.

Estimating Execution Time

At the Naval Postgraduate School, we estimate execution time of segments of the PASS software by analyzing records of test cases in digital simulations of operational flight scenarios as well as records of actual use in *shuttle* operations. Test case executions are only counted as *operational execution time* for previous-version subsets of the version being tested if the simulation fidelity very closely matches actual operational conditions.

Prerelease test execution time for the new code actually being tested in a version is never counted as operational execution time. We use the failure history and operational execution time history for the new code subset of each version to generate an individual reliability prediction for that new code in each version by separate applications of the reliability model.

This approach places every line of code in the total PASS into one of the subsets of *newly* developed code, whether it is new for the original version or any subsequent version. We then represent the total reliability of the entire software system as that of a composite system of separate components (*new code subsets*), each having an individual execution history and reliability, connected in series. Lockheed Martin is currently using this approach to apply the Schneidewind [8, 9] model as a means of predicting a *conservative lower bound* for the PASS reliability.

Verification and Validation

Software reliability measurement and prediction are useful approaches to verify and validate software. Measurement refers to collecting and analyzing data about the observed reliability of software, for example the occurrence of failures during test. Prediction refers to using a model to forecast future software reliability, for example failure rate during operation. Measurement also provides the failure data that is used to estimate the parameters of reliability models (i.e., make the best fit of the model to the observed failure data).

Once the parameters have been estimated, the model is used to predict the software's future reliability. Verification ensures that the software product, as it exists in a given project phase, satisfies the conditions imposed in the preceding phase (e.g., reliability measurements of safety-critical software components obtained during test conform to reliability specifications made during design) [5]. Validation ensures that the software product, as it exists in a given project phase, which could be the end of the project, satisfies requirements (e.g., software reliability predictions obtained during test correspond to the reliability specified in the requirements) [5].

Reliability Measurements and Predictions

There are a number of reliability measurements and predictions that can be made to verify and validate the software. Among these are *remaining failures*, *maximum failures*, *total test time required to attain a given fraction of remaining failures*, and *time to next failure*. These have been shown to be useful measurements and predictions for: (1) providing confidence that the software has achieved reliability goals, (2) rationalizing how long to test a software component (e.g., testing sufficiently to verify that the measured reliability conforms to design specifications), and (3) analyzing the risk of not achieving *remaining failure* and *time to next failure* goals [6].

Having predictions of the extent to which the software is not fault-free (remaining failures) and whether a failure is likely to occur during a mission (time to next failure) provides criteria for assessing the risk of deploying the software. Furthermore, the fraction of remaining failures can be used as both an *operational quality* goal in predicting total test time requirements and, conversely, as an indicator of operational quality as a function of total test time expended [6].

Risk Assessment

Safety risk pertains to executing the software of a safety-critical system where there is the chance of injury (e.g., astronaut injury or fatality), damage (e.g., destruction of the shuttle), or loss (e.g., loss of the mission) if a serious software failure occurs during a mission. In the case of the shuttle PASS, where the occurrence of even trivial failures is extremely rare, the fraction of those failures that pose any impact to safety or mission success is too small to be statistically significant.

As a result, for this approach to be feasible, all failures (of any severity) over the entire 20-year life of the project have been included in the failure history database for this analysis. Therefore, the risk criterion metrics to be discussed for the shuttle quantify the degree of risk associated with the occurrence of *any* software failure, no matter how insignificant it may be. The approach used can be applied to safety risk where sufficient data exist.

Two criteria for software reliability levels will be defined, then these criteria will be applied to the risk analysis of safety-critical software using the PASS as an example. In the case of the shuttle example, the risk represents the degree to which the occurrence of failures does not meet required reliability levels, regardless of how insignificant the failures may be. Next, a variety of prediction equations that are used in reliability prediction and risk analysis have been defined and included in the document; included is the relationship between *time to next failure* and *reduction in remaining failures*. Then it is shown how the prediction equations can be used to integrate testing with reliability and quality. An example is shown of how the risk analysis and reliability predictions can be used to make decisions about whether the software is ready to deploy; this approach could be used to determine whether a software system is *safe* to deploy.

Criteria for Reliability

If the reliability goal is the reduction of failures of a specified severity to an acceptable level of risk [10], then for software to be ready to deploy, after having been tested for total time (t_t), it must satisfy the following criteria:

Predicted remaining failures

$$r(t_t) < rc \qquad (1)$$

where,

rc is a specified critical value, and

Predicted time to next failure

$$TF(t_t) > t_m \qquad (2)$$

where,

t_m is mission duration

The total time (t_t) could represent a safe/unsafe criterion, or the time to remove all faults regardless of severity (as used in the shuttle example).

For systems that are tested and operated continuously like the shuttle, t_t, TF (t_t), and t_m are measured in execution time. Note that, as with any methodology for assuring software reliability, there is no guarantee that the expected level will be achieved. Rather, with these criteria, the objective is to reduce the risk of deploying

the software to a *desired* level.

Summary

The existing AIAA practice and planned revisions have been described. The principles of SRE, as applied to the revision have been reviewed. A life-cycle approach to SRE in the revision has been emphasized. The revision is expected to be an important life-cycle software reliability process document to achieve the following objectives:

- Provide high reliability in Department of Defense (DoD) and aerospace safety and mission-critical systems.
- Provide a rational basis for specifying software reliability requirements in DoD acquisitions.
- Improve the management of reliability risk.◆

References

1. American Institute of Aeronautics and Astronautics. AIAA Recommended Practice for Software Reliability (R-013-1992). ISBN: 1563470241. Reston, VA: AIAA, 1992.
2. Billings, C., et al. "Journey to a Mature Software Process." IBM Systems Journal 33.1 (1994): 46-61.
3. Keller, Ted, and N.F. Schneidewind. "Successful Application of Software Reliability Engineering for the NASA Space Shuttle." Software Reliability Engineering Case Studies. International Symposium on Software Reliability Engineering, Albuquerque, N.M., Nov. 1997: 71-82.
4. Musa, J., et al. Software Reliability: Measurement, Prediction, Application. New York: McGraw-Hill, 1987.
5. Schneidewind, N.F., and T. Keller. "Application of Reliability Models to the Space Shuttle." IEEE Software 9.4 (July 1992): 28-33.
6. Schneidewind, N.F. "Reliability Modeling for Safety-Critical Software." IEEE Transactions on Reliability 46.1 (Mar. 1997): 88-98.
7. Musa, J., et al. Software Reliability: Measurement, Prediction, Application. New York: McGraw-Hill, 1987: 166-176.
8. Schneidewind, N.F. "Report on Results of Discriminant Analysis Experiment." 27th Annual NASA/IEEE Software Engineering Workshop, Greenbelt, MD., 5 Dec. 2002.
9. Keller, Ted, N.F. Schneidewind, and P.A. Thornton. Predictions for Increasing Confidence in the Reliability of the Space Shuttle Flight Software. Proc. of the AIAA Computing in Aerospace 10, San Antonio, TX, 28 Mar. 1995: 1-8.
10. Schneidewind, N.F. Reliability and Maintainability of Requirements Changes. Proc. of the International Conference on Software Maintenance, Florence, Italy, 7-9 Nov. 2001: 127-136.

Additional Reading

1. Schneidewind, N.F. "Software Reliability Model With Optimal Selection of Failure Data." IEEE Transactions on Software Engineering 19.11 (Nov. 1993): 1095-1104.
2. Farr, W., and O. Smith. Statistical Modeling and Estimation of Reliability Functions for Software (SMERFS) Users Guide. NAVSWC TR-84-373, Revision 3. Naval Surface Weapons Center, Revised Sept. 1993.
3. Lyu, M. Handbook of Software Reliability Engineering. New York: McGraw-Hill, 1995.
4. Musa, John D. Software Reliability Engineering: More Reliable Software, Faster Development and Testing. New York: McGraw-Hill, 1999.
5. Schneidewind, N.F., and T. Keller. "Application of Reliability Models to the Space Shuttle." IEEE Software 9.4 (Jul. 1992): 28-33.
6. Voas, J., and K. Miller. "Software Testability: The New Verification." IEEE Software 12.3 (May 1995): 17-28.

About the Author

Norman F. Schneidewind, Ph.D., is professor of Information Sciences in the Department of Information Sciences and the Software Engineering Group at the Naval Postgraduate School. Schneidewind is a Fellow of the Institute of Electrical and Electronics Engineers (IEEE), elected in 1992 for "contributions to software measurement models in reliability and metrics, and for leadership in advancing the field of software maintenance." In 2001, he received the IEEE "Reliability Engineer of the Year" award from the IEEE Reliability Society.

Naval Postgraduate School
2822 Raccoon TRL
Pebble Beach, CA 93953
Phone: (831) 656-2719
(831) 372-2144
(831) 375-5450
Fax: (831) 372-0445
E-mail: nschneid@nps.navy.mil