General Disclaimer

One or more of the Following Statements may affect this Document

- This document has been reproduced from the best copy furnished by the organizational source. It is being released in the interest of making available as much information as possible.

- This document may contain data, which exceeds the sheet parameters. It was furnished in this condition by the organizational source and is the best copy available.

- This document may contain tone-on-tone or color graphs, charts and/or pictures, which have been reproduced in black and white.

- This document is paginated as submitted by the original source.

- Portions of this document are not fully legible due to the historical nature of some of the material. However, it is the best reproduction available from the original submission.

Produced by NASA STI Compliance and Distribution Services

77-1387

CONSIDERATIONS FOR A SUCCESSFUL SOFTWARE TEST PROGRAM

Frank J. Mullin
TRW Defense and Space Systems Group
Redondo Beach, California

Abstract

This paper discusses software testing and software test activities as they should occur during the software development process. It notes that a well planned test program starts at the time software requirements are being produced and continues on through acceptance testing. It describes the activities and responsibilities of the group assigned to formally test the software and discusses their interactions with the people preparing the software requirements and the people developing the software. Three levels of testing are defined and suggested test support software is identified.

1. Introduction

Testing is often the most overlooked aspect of any software development project. This is not because anyone thinks it's unimportant — everyone agrees it must be done — but simply because it's something that can be put off for a while, something that can be started later, something that we'll do when the time is right. It's a habit that has been practiced by almost everyone at some time or another, and the consequences of this mode of development have given rise to the saying that "the test program is complete when the budget runs out." Needless to say, this is not a very satisfying situation for either the manager of the software development project or his customer, and this is being recognized more and more by both parties. Program managers are becoming willing to spend money for testing and customers are now willing to approve these expenditures. Both realize that even though the actual testing will occur late in the development cycle, test activities should be started early (hopefully during the generation of the software requirements) in order to keep overall test costs within bounds.

2. The Development Environment

Bounding of test costs is enhanced by utilizing the leverage resulting from test activities at the start of a program to derive a sensible, achievable test program which is intended to influence both software requirements and software design in order to produce high-quality, fully-testable software. This paper will expand on these thoughts and discuss just what is involved in a well-planned software test program. It will discuss the test activities that should start early as well as what should be historically recorded after the actual testing has been completed. It is based on the author's three years' experience on the Systems Technology Program (formerly the Site Defense Program) as the Assistant Program Manager for Software Integration and Test, and as the Assistant Program Manager for Software Development on several modest sized ($1M to $2M) programs.

It will be keyed to the software development phases shown in Figure 1 (Reference 1) and will be discussed from a position which is based upon the premise that testing must be a planned, integral part of the software development effort. In discussing the various aspects of testing it is assumed that the entire development process can be characterized as being "disciplined" (Reference 2). A written set of requirements exists, there are scheduled preliminary and detailed design reviews, the testers use customer-approved test plans and procedures, and the software (which has to be delivered to the customer by a particular date) goes under configuration control prior to the start of formal testing. Each agency involved in the development has been given an agreed-to budget and schedule and reports against both items, usually on a monthly schedule.

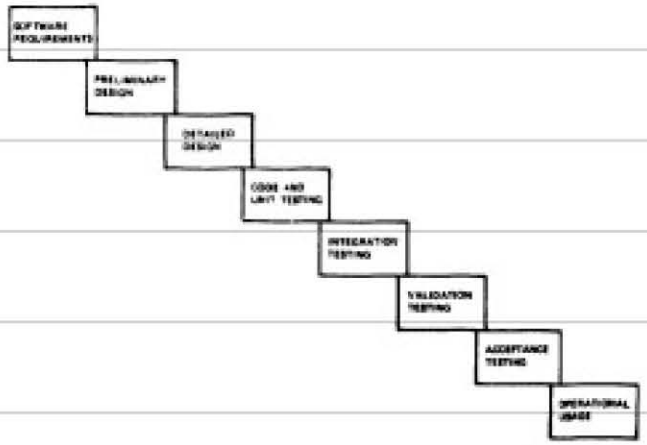


Figure 1. Software Development Phases

3. Levels of Testing

In general, there are three levels of testing in a good test program. The first is that performed by the people who have developed the software and is called unit testing. It is intended to determine that their design has been implemented correctly, as compared to their design documentation, and that limit checking, logical operations, arithmetic calculations, etc., have all been tested. This level of testing includes the usual programmer debugging, exhaustive stand-alone testing and a modest amount of string testing. It is intended to give the designers confidence that their software is ready for the next level of testing, in which the objective is to integrate individual software modules to, in some sense, determine that the software functionally does what it is supposed to do. This second or intermediate level of testing is called integration testing or functional testing and consists of taking individual software modules and integrating them to form a complete software system (Reference 1) which does accept input data, which does produce output data, and which appears to operate properly. It is intended to provide the people who are going to undertake the third level of testing with software that does function, thereby avoiding the very real problem of making the software cycle for the first time when formal testing of the software is initiated. The third level of testing will be called

Copyright © American Institute of Aeronautics and Astronautics, Inc., 1977. All rights reserved.

validation testing in this paper. It is intended to prove that the software does satisfy its performance requirements, as expressed in a requirements specification, and that the integrated software system does really meet system-level requirements when operated in the basic operational sequences. Practically speaking, this level of testing has been most effective when performed by a group which is organizationally independent from the development groups. This is because the usual programmer, if allowed to test his own software, will always determine that his software does indeed satisfy all of its requirements. (If it didn't, he wouldn't be finished with it.) To avoid this, the personal aspect of testing is removed by having a separate test group conduct validation testing and independently determine whether or not the requirements are met. Acceptance testing, which is undertaken to demonstrate to the customer that the software meets his requirements, is at the same level as validation testing and usually consists of executing a subset of the validation tests with the test data being analyzed by the customer.

4. Test Activities in the Early Development Phases

More test problems are created by vague and ambiguous requirements than any other single factor. The reason for this is straightforward. The test group has to run a test and make a "yes/no" determination of whether or not the software passes the test. To do this they have to interpret all ambiguous requirements in some definitive way and, as a consequence, they often subsequently spend a considerable amount of time justifying and explaining their interpretation of the uncertain requirements. Since not all of their interpretations will be one hundred percent correct, and what they call a failed capability may indeed be a satisfactory capability, the test activities should start during the time the software requirements are being produced to allow testers to work in parallel with the people generating the requirements, to understand the requirements and determine whether or not these requirements are testable (Reference 3).

This must be done to eliminate, or precisely define, requirements which state that the "data processing system shall degrade gracefully under overload conditions" or that the "data processing system shall continue to operate under intermittent hardware errors." This type of requirement, if it is not removed from the requirements specification, has a definite meaning to the person writing the requirements, a second meaning to the person designing the software and will inevitably be interpreted to mean a third thing by the independent tester, thereby fulfilling all aspects of Murphy's law (Reference 4). This entire hassle can be avoided by having the test people review the requirements before they are finalized and determine that they can conceive of a test to demonstrate satisfaction of each requirement. Those requirements which are deemed untestable are flagged and called to the attention of the systems engineering group preparing the requirements. The changes to the untestable requirements are derived by consultations and meetings between the test and the system engineering groups.

For the purposes of this paper let the software described above be called the applications software – the software that is deliverable to the customer – so as to distinguish it from a second category which will now be introduced. It was mentioned that, in reviewing the software requirements, the testers conceived a test that could be run to prove that an individual requirement would be satisfied. This was to be done for every requirement. In addition to reviewing requirements, this conceptualization of tests also serves another very important purpose; when all of these conceptual tests are compiled and examined, they form the basis for generating test driver requirements, i.e., the requirements on that combination of hardware and software which is necessary to test the applications software. The generation of these test driver requirements is quite often not a trivial undertaking, particularly if one is testing real-time software in a reactive, closed-loop mode with the test driver providing a response at the proper time for each stimulus generated by the applications software (Reference 5). If the environment which has to be modeled by the test driver is complex, there is the added burden of ascertaining that the test driver adequately models this environment. This is necessary in order to avoid the embarrassing situation in which the applications software performs superbly with the test driver but fails to even cycle when installed at its operational site.

5. Test Plans

To form a solid basis for later test activities and to also consolidate all of the ideas and concepts on testing the software, a test plan should be produced early in the program. The test plan, one of which should be prepared for each category of software being produced, contains the purpose, goals and an overall description of the testing that is to be performed. The test plans for both the unit and integration test phases are usually informal, working-level documents while the validation and acceptance test plans are most often contractually-deliverable documents that are formally reviewed and approved by the customer. They are reviewed and approved at the PDR and, after approval, are used as a part of the basis for generating test procedures. The validation test plan should contain the following information (Reference 6):

(1) A description of the scope, purpose and objectives of validation testing,

(2) A discussion of the resources required to conduct the tests,

(3) A description of the test management and test control procedures including:

  (a) Configuration control procedures,

  (b) Test control procedures,

  (c) Retest criteria,

  (d) Test review,

  (e) Test identification conventions

(4) Specific testing requirements including:

(a) An identification number and test title,

(b) The requirement to be used,

(c) The computer program element(s) involved in the test,

(d) A brief description of the test including the data to be collected,

(e) The test verification method,

(f) The acceptance criteria

(5) Software requirements traceability/ relationship to individual tests,

(6) Preliminary test schedule.

The acceptance test plan, which describes the test cases that must be successfully executed before the software is accepted for operational use by the customer, specifies the subset of validation test cases which are to be executed for acceptance purposes. In addition to the above information the acceptance test plan should contain:

(1) An identification of the support hardware, software and facilities to be used in acceptance testing,

(2) A description of the data base to be used and the plan for obtaining customer approval of it,

(3) A discussion of the roles and responsibilities of all participants,

(4) A summary of all customer-provided hardware, software, data, documentation and support,

(5) An acceptance test schedule.

6. Preparation for Testing

A prerequisite for a good software test program – good in the sense that it is thorough and completed on schedule – is a good design since no amount of testing can overcome basic design inadequacies. For this reason it is essential that test personnel participate in both the preliminary and critical design reviews (Reference 7 and 8) and that they have a definite input to the testing which is to be performed by the developers. The former will permit them to express their professional judgment on the adequacy of the proposed design, while the latter activity is intended to prevent the software from being turned over to the test group simple because it's the scheduled date for the software to be turned over. The latter is accomplished by requiring the developers to test their software against a Functional Capabilities List (FCL) generated by the testers to guarantee that the turned-over software has reached some degree of maturity with respect to testing. This guarantee is enforced by placing test personnel in the design group to assist them with their testing and by having the test group not accept the software until it has been completely tested by the developers against the FCL for that particular piece of software. A sample Functional Capability List is shown in Figure 2.

SPECIFICATION REQUIREMENT

THE OBJECT TRACK FUNCTION SHALL BE CAPABLE OF TRACKING OBJECTS IN ESTABLISHED ORBITS USING THE T1 WAVEFORM WITH A POSITION ERROR THAT SHALL NOT EXCEED 1000 FEET (3σ).

SAMPLE FCL'S

1. ONLY VALID T1 PULSES ARE USED IN THESE COMPURATIONS – FOR INVALID DATA THE ROUTINE IS EXITED.

2. DATA WITH ESCESSIVELY LARGE NOISE COMPONENT IS DISCARDED IN ESTIMATING THE EPHEMERIS OF ORBITING OBJECTS.

3. THE VARIANCE ON EACH COMPONENT IS COMPUTED USING THE NOISE LEVEL AS PER EQUATION X.XX.

Figure 2. A Partial Functional Capability List

There are several aspects of the FCLs which are worthwhile calling to the attention of the reader. First, as a matter of course every FCL contains the requirement to limit test that software unit to verify its performance at its operating limits, to test its operation using erroneous data to drive the software, to check for units consistency and to hand check the arithmetic computations that the software performs. These are the types of tests that any experienced programmer-analyst routinely performs in developing his software. Second, the FCLs are based on the proposed design and not on the requirements which the design is intended to satisfy. They are developed from design documentation produced by the designers for their design reviews (see Figure 3) and, consequently, the unit testing by developers is intended to verify that their code contains those capabilities they said it would contain in their design documentation. Finally, the FCL should require the software designer to execute and test every path segment during his unit testing. Software test tools (Reference 9 and 10) are used to verify that the unit testing causes every path segment to be exercised. This does not guarantee that the developer has thoroughly tested his piece of software; it simply means that he has exercised every path segment in his code. It does guarantee that the tester will think about his tests to make certain that he does exercise every path segment; whether or not his tests are really the proper ones is a judgment that can only be made from an examination of the actual test cases. From a practical view the need to test against FCLs and the need to test every path segment has resulted in the designers paying considerable attention to their test programs – much more than they might be expected to do normally.

In addition to FCLs which address unit-level tests, it is also necessary to prepare FCLs to support integration testing. The latter FCLs, as

indicated in Figure 3, are used to define the integration tests which will be run to bridge the gap between the unit-level tests of individual software modules and the system-level tests conducted during validation testing.

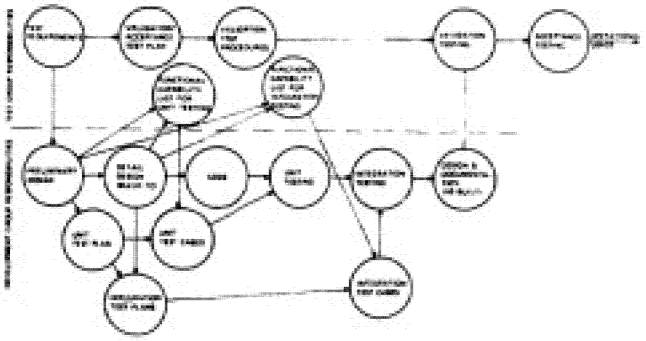
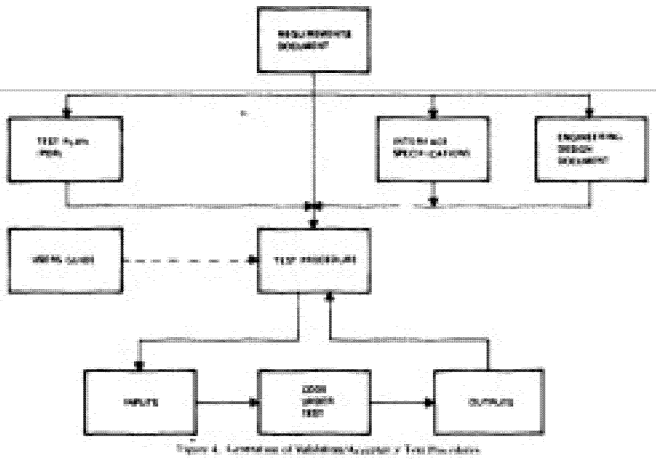


Figure 3. Test Activities

During the time the software designers are coding and testing their design, the test group is busy preparing validation and acceptance test procedures. The corresponding test plans have been reviewed at the Preliminary Design Review (PDR) and are among the support documents used in the generation of the test procedures (see Figure 4). The test procedures document provides the details on all of the activities required to test the software and subsequently analyze the test data to prove that the software does indeed satisfy a definite requirement.



Since the validation and acceptance test procedures document contains the detailed information on the individual tests, this document should address the following items for each procedure:

(1) The overall scope of each test,

(2) The objectives of the test,

(3) A description of the method of testing,

(4) The conditions necessary to execute the test,

- Personnel
- Equipment
- Input material
- Output expected
- Test preparation and setup
- Operator actions and console inputs, if required
- Prerequisite testing

(5) A step-by-step description of the procedure,

(6) The acceptance criteria,

(7) The retest requirements if the test fails.

7. Integration Testing

Integration testing is the intermediate phase of testing which bridges the gap between the so-called stand-alone testing of individual software units (performed by the developers) and the system-level validation testing (performed by the independent test group). It is the type of testing in which the individual software modules, which in total comprise the complete software system, are incrementally merged and tested to form the software system to be used in validation testing.

Just as in unit testing, Functional Capability Lists are used to generate test cases. In this case system-level Functional Capability Lists are generated by the independent test group to first test the flow of data into, out of, and within the software system under nominal or expected conditions and then to evaluate the performance of the system under off-nominal conditions. This takes the form of testing to determine that the software system can accept input data at various rates, that it can extract data from an input message, that it can produce an output message, and that the message content is indeed reasonable. It is testing to show that the software system does have the capabilities described in its design documentation.

Integration testing is also the first time that the software system is stimulated with an external driver, and it is important that this test driver be completed prior to the start of this testing. If the driver is also undergoing testing of its own while the software is being integrated, chances are that the integration test phase will take longer than the time allocated. The time to conduct integration testing is further reduced if an operational data reduction program is available. This means that test data can be analyzed and the success of a test determined quickly. This is particularly necessary when the test is one in which a proposed fix to a previously-uncovered error is being tested, and testing is held up until the error is corrected.

Even though it was mentioned before, it is worthwhile to note that there are two aspects of integration testing. The first part is directed to integrating the software modules into a cohesive, nominally-executing software system which, at the conclusion of this phase of integration testing, can reasonably be expected to execute satisfactorily. The interface problems have been worked and the data being produced by the software appears to be correct. It produces sufficiently accurate data on a proper timeline for nominal inputs. It is ready to undergo the second

integration test phase in which a limited number of off-nominal operating conditions will be invoked and the performance of the software evaluated for these conditions. This testing is designed to prepare the software for the more extensive "off-nominal conditions" testing that is to be conducted during validation testing. It makes no sense to wait for validation testing for this type of testing since it almost invariably results in some changes to the software.

The most important job of the testers in the preparation for integration testing is to make certain that they have a complete definition of the interfaces between the software modules being integrated. They must know the format and content of all data that is transferred between modules and use this as the basis for determining that data is flowing correctly between the various modules. If such a document does not exist, it can be produced by the independent test group as part of its assignment in preparing for integration testing. The same information is also needed to test the interface between the application software (the software being tested) and the external test driver program. This is usually available in an Interface control Document or Interface Specification.

At the conclusion of integration testing the software is turned over to the independent test group for validation testing. Most of the time an integration test report is prepared, primarily to record the results of individual tests. This is often a minimal report, since it is usually not used by anyone doing validation testing and serves primarily to record the results of individual integration tests for historical usage.

8. Validation Testing

Validation testing, the reader may recall, is that software testing whose purpose is to show that the software system does meet all of its system-level requirements. The test objectives are derived from the software requirements specification and not software design documentation. The software and its documentation and the validation test plans and procedures are placed under configuration control before the start of this test phase which is conducted by a project test organization which does not report to the project's development organization. It is in every sense the formal test phase of the development process and, if it is successfully completed, the acceptance test phase should present no subsequent problems.

From a test viewpoint, the question of when the software is to be given to the independent test group is a particularly important one. The developers, and sometimes the integration test team, would like to give their software to the test group as late as possible; they can always use more time to improve their designs or test the software a little more, or even to document their design. The test group would like to have the software to test as early as possible. This minimizes the risk that the software will not be thoroughly tested when it is to be delivered to the customer. Since both views have to be accommodated, a turnover date has to be negotiated by the developers and testers. They must also agree on exactly what is to be turned over to the testers. Practically speaking, however, if the turnover can be made on an incremental basis rather than as a one-time event, both the developers and the testers will benefit. It is strongly suggested that this type of arrangement should be made if at all possible.

At turnover, as a minimum the test group should receive from the developers (i.e., the software development group and the integration test team) both source and object code, full documentation on the software being turned over, a users' manual, and an identification of all known deficiencies that the test group should not bother to test in that particular release of the software. In addition, the test group should obtain several test cases recently run by the integration test team. These test cases should be re-run to gain some direct experience with the software by operating it under known, proven test conditions and to also verify that the software being turned over is the same as that used by the developers in their testing. This is a good check, since the software will be placed under formal configuration control at the time it is turned over for validation testing. At turnover, when the software and its documentation go under formal configuration control, the test procedures do also.

The fundamental premise of validation testing is that the software is being tested against an established set of requirements — a baseline which is fixed well in advance of the start of testing and which has been used in the generation of validation test plans and procedures. If the requirements baseline is not fixed and is changing before and during validation testing, the test program is in trouble since the testers will not know what they're testing against or even when they are finished. The problem is even further compounded by developers turning over updated software to satisfy the changing requirements. The entire question of how much re-testing of software is necessary as a result of the changing requirements and updated software results in a test program which is completed "by definition" on the day that the schedule requires testing to be completed. This is often summarized by test people in the following way — "Testing against a set of requirements is like walking on water — it helps when it is frozen."

Good configuration management practices (Reference 6, Section 2) are also required if the test program is to be successful. The test group has to know which version of the software it should be using in the latest tests, which previously-discovered errors have already been corrected and are contained in the updated version of the software, and the particulars of these changes as described in the software documentation so that any necessary changes to the test procedures can be submitted to the Configuration Control Board for approval. With poor configuration management in a test program, sooner or later all test groups will wind up testing a slightly different version of the software. It will also be this way at the finish of the testing with no individual version fully satisfying all of the requirements.

The reader has probably deduced by this time that validation testing is indeed something

different from testing done by the software developers. It is conducted according to an agreed to and controlled set of test procedures, and its purpose is to prove that the software does meet or exceed its requirements. It demonstrates that the software does have the required accuracy, that its response times to input stimuli are satisfactory, that the data processing system interfaces with the external environment are correct, and that the capacity to handle the specific data processing load is inherent in the operation of the software. It is a series of tests designed to thoroughly test all capabilities of the software against the requirements specification in order to determine that the software is ready to be used by the customer. It is the last activity of the company developing the software before acceptance testing.

It is important to call attention to the phrase "validation test procedures." These procedures contain a definitive set of tests which have been proposed by the test group to the customer and which, when approved by the customer, constitute the entire set of tests which must be successfully completed in order to prove that the software does meet its requirements. The negotiation and agreement gives the customer the opportunity to see that tests he would like to have run are included in the test program, and it provides the test group with a situation in which they do not "stop the test program when the program budget is expended." They stop when they have completed all of the approved tests and do not have to worry about always having to run another customer-dictated test. This not only gets the customer involved in the test program, it also forces the test group to really think about what they are doing since their proposed series of tests is going to be thoroughly reviewed by the customer and, when approved, "locked in concrete" by the Configuration Control Board. It is the complete list of tests which the testers propose to do for the budget which has been allocated to them and, consequently, represents their commitment to both the Project Manager and his customer.

Finally, at the conclusion of any test it is necessary that the test group prepare a test data package for each test. The test data package which contains the complete record of each test should include an identification of the version of the software involved, a copy of the test procedure used to conduct the test, the test results, and the analysis performed on the test data. The purpose of the test data package is not only to provide historical results of the test but also to provide the information needed to re-run a test at some future time. The actual software for re-running a test would, of course, be provided by the Configuration Management activity of the project.

9. Acceptance Testing

The acceptance test phase is initiated at the completion of validation testing. The acceptance test plan was reviewed at PDR and most likely reviewed again at the Detailed (or Critical) Design Review and then approved by the customer. The plan defines all of the tests that must be successfully completed before the customer will formally accept the software and its documentation. As a general rule the acceptance tests are a subset of the validation tests and, if all of the latter tests have been successfully completed, no trouble should be experienced during acceptance testing.

Prior to the start of acceptance testing it is necessary to make certain that all modifications to the software have been incorporated, that all failed tests (those in which errors were uncovered) have been successfully re-run, and that the design documentation has been updated (at lease in draft form) for each modification made to the software. This will allow acceptance testing to start with everything in order and correspondence and will minimize the confusion over whether the software or documentation is correct in the event some anomalous behavior is discovered.

Since all of the tests which are being run during this phase have already been successfully executed, one wonders how any errors at all will be uncovered during this phase. The response to this question is, of course, that the software is not error free and that a detailed examination of the test data by a new group of people, in this case the customer and his technical support personnel, will inevitably find things which were not obvious or were overlooked during validation testing. This is particularly true when future users of the software are members of the customer's acceptance test team. They are well aware of the real operational environment and are extremely interested and helpful in determining that the software system meets its requirements. At the completion of acceptance testing, the software is formally released to and accepted by the customer.

10. Summary and Conclusions

If all of the above material on testing were to be summarized succinctly, the summary would convey two ideas. First, it is necessary that testing be considered early in the development phase if the test program is to be successful and not cost "an arm and a leg" later on. Second, the software required to support a test program is non-trivial and must be considered from a cost and schedule viewpoint when initially planning the entire software development program. Failure to do both results in a test program which will inevitably produce inadequately-tested software or a test program which will end when the budget runs out or, more probably, a test program that is adequate but which is accomplished by schedule and attendant cost overruns. There is considerable leverage on test dollars expended early in the development phase, and a good manager will certainly take advantage of this.

11. References

1. Royce, W.W., "Managing the Development of Large Software Systems: Concepts and Techniques," 1970 WESCON Technical Papers, Los Angeles, Calif., 25-28 August 1970. (This paper has also been published in the TRW Software Series as TRW-SS-70-01, dated August 1970.)

2. Proceedings of the TRW Symposium on Reliable, Cost Effective, Secure Software, TRW Software Series, TRW-SS-74-14, March 1974, Section 2.

3. Rubey, R.J., Dana, J.A., and Biche', P.W., "Quantitative Aspects of Software Validation," IEEE Transactions of Software Engineering, Vol. SE-1, No. 2, June 1975, Pages 150-155.

4. Boehm, Brown, Kaspar, et al, "Characteristics of Software Quality," December 1973, TRW Software Series, TRW-SS-73-09.

5. Kaspar, Friedman, Katz, et al, "A Real-Time Software/Computer System Simulation Model," December 1970, TRW Software Series, TRW-SS-70-03.

6. Software Development and Configuration Management Manual, TRW Software Series, TRW-SS-73-07, December 1973, Pages 9-17.

7. Ibid, Section 8

8. Department of Defense, "Technical Reviews and Audits for Systems, Equipment and Computer Programs," MIL-STD-1521 (USAF), 1 September 1972.

9. J.R. Brown and R.H. Hoffman, "Automating Software Development: A Survey of Techniques and Automated Tools," TRW Software Series, TRW-SS-72-03, May 1972.

10. J.R. Brown and R.H. Hoffman, "Evaluating the Effectiveness of Software Verification — Practical Experience with an Automated Tool," TRW Software Series, TRW-SS-72-08, December 1972 (also published in the Proceedings of the Fall Joint Computer Conference, December 1972).