Software reliability—A method that works

by R. H. THAYER
Space and Missile Test Center
Vandenberg AFB, California

and

E. S. HINTON
Logicon, Inc.
San Pedro, California

INTRODUCTION

Software reliability is receiving increased attention from a broad spectrum of computer users as larger computer programs continue to be implemented in diverse and widespread areas. The reason is fundamental: software reliability has been poor on many large systems and poor on systems which have a high degree of human interaction.

The Department of Defense has had several systems with significant software problems. Current Air Force systems have suffered development and maintenance difficulties that were directly attributable to poor software reliability. The Strategic Air Command's command and control system (465L) and the F-111 Mark II avionics system are both examples of such systems. Many missile systems had software error problems that, at one time, gave program managers cause for concern.

It is not surprising, therefore, that many attempts have been made by the Air Force to improve the software reliability picture, especially in the space and avionics system areas. Several management and technical approaches, including those that have worked well for hardware, are being used and developed with varying degrees of success.

One of the most successful approaches to improving software reliability that the Air Force has used to date, particularly on highly critical real-time software, is the utilization of an independent software test agency, to perform analysis, testing, and overall evaluation on the system.

In fact, the improvement in reliability as a result of independent analysis and testing of software has prompted the Air Force to include the following policy statement in its new regulation, AFR 800-14, dated 10 May 1974:

"Program Management Directives require, and Program Management Plans provide for: Establishment of computer technical and managerial expertise responsive to the Program Office which is independent of the system prime or computer program development contractor and, preferably, an organic capability of the Program Office."

Independence is the key ingredient in this analysis and testing function, and in the context of the discussion, implies that the function is accomplished by an agency whose sole purpose is the detection of errors in the developed software, i.e., the agency has no development responsibility.

BACKGROUND

The beginning

The independent analysis and testing concept has been evolutionary in development, and it will continue to evolve as new and varied analysis and test techniques and tools are designed. Initial conception of the "independent" approach is believed to have been the result of critical software failures on early missile and space launches. The first application of the approach is believed to have been on the software associated with the Titan II and III during the 1962-1964 time period.

In the critical area of missile flight safety, instances have been reported where range safety system software has failed causing costly delays of launch operations. Furthermore, software problems have been detected which could have produced a situation where a good launch would appear bad to the missile flight control officer. Such situations would require the destruction of an expensive vehicle and payload.

The costs associated with such failures prompted the use of, what in 1971 were then considered to be, drastic actions. The Air Force, considering the expense of failed launch missions, proceeded to hire an independent contractor to analyze the software in a "validation" process, prior to releasing the software for operational use. It was felt that an outside or fresh look at the software should reveal faults, whereas the development contractors and programmers were "too close to the problem" to come up with an objective analysis. This approach proved to be highly successful and this method of software validation has been used at Vandenberg AFB from 1971 until the present.
INDEPENDENT ANALYSIS AND TESTING

Figure 1—Software development process with independent analysis and testing

Present usage

The confidence gained from successful missile launches and other developments that utilized the dual contractor approach prompted continuing it as a standard procedure on critical, high-risk software packages. Examples of DoD systems which use and have used an independent agency for software testing and analysis are Safeguard, Minuteman II and III, Titan, and B-1.

In general, most of the independent analyses and testing performed to date have been done on real-time control programs. There are however, extremely critical programs, such as the real-time software that controls nuclear weapon systems, which have become prime candidates for testing and evaluation by an independent agency. This procedure is now standard Air Force practice on such systems and is prescribed in Air Force Regulation 122-9, dated 19 July 1974. The fact is, that on the most critical software with which the nation is involved, that which controls nuclear weapon systems, analysis and evaluation by an agency independent from the software developer is mandatory. This strongly implies that it is one of the most effective known ways to achieve software reliability.

DESCRIPTION OF THE INDEPENDENT TESTING AND ANALYSIS METHODS

A software development life cycle is illustrated in the block diagram of Figure 1 and can be considered a chain of events beginning with a customer's requirements and ending with the operation and maintenance of the software through the program's useful life. Independent Testing and Analysis (IT&A), is a nonspecific set of analyses, studies, tests, and evaluations that is conducted by an independent agency on a computer program that are intended to verify compliance with requirements and show correctness of programming. IT&A does not absolutely guarantee verification or correctness of software, but it improves confidence that the program can be depended upon to perform reliably in an operational environment. As illustrated in Figure 1, independent analysis and testing is most effective when applied across all phases of this development. From the developer's point of view, software development is a serial process with frequent iterative loops between adjacent phases. The independent tester, on the other hand, is not in the direct chain of development, and he analyzes the output of each phase as to its ability to satisfy input requirements, e.g., specification to design, or design to coding. He performs this function for all phases as soon as the output becomes available, again with the sole purpose of detecting errors.

INDEPENDENT TEST AND ANALYSIS WORKS

Some proof

Independent Test and Analysis can find errors that are not found through ordinary software testing. A software reliability study recently completed for the Advanced Ballistic Missile Defense Agency has pointed out the importance of early error detection. In the study report, software errors found through independent analysis on eleven separate projects were categorized as being catastrophic, serious, moderate or trivial. These categories were defined essentially as follows: catastrophic errors which would terminate the program execution; serious errors which could severely degrade the program performance but would not be fatal; moderate errors which would not have major impact on program performance; and trivial errors which would have no effect on program performance, e.g., flow chart did not match code, though code was correct. A summary of the errors discovered, by severity, is shown in Table 1.

Significantly, many errors which might initially have been considered trivial, if not discovered early in program development, would have created error conditions of much greater severity. Independent analysis discovered errors in released specifications that, at that time, were considered trivial because of the relative ease of correction, hence the 553 trivial errors in the top row of Table 1. The effect of implementing those "trivial" errors into program design and code would have been to increase catastrophic, serious and moderate errors to the numbers shown in the bottom row of Table 1.

<table>
<thead>
<tr>
<th>ERROR SEVERITY</th>
<th>CATASTROPHIC</th>
<th>SERIOUS</th>
<th>MODERATE</th>
<th>TRIVIAL</th>
<th>TOTAL ERRORS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERRORS DISCOVERED</td>
<td>21</td>
<td>149</td>
<td>479</td>
<td>553</td>
<td>1202</td>
</tr>
<tr>
<td>EARLY IN SPECIFICATION</td>
<td>36</td>
<td>200</td>
<td>558</td>
<td>408</td>
<td>1202</td>
</tr>
<tr>
<td>ERRORS ANTICIPATED</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AFTER IMPLEMENTATION</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

From the collection of the Computer History Museum (www.computerhistory.org)
row. Trivial errors would have decreased to 408 as they were reflected in more serious categories.

These trivial errors were concerned primarily with requirements and specifications. In fact, 31 percent of all non-trivial errors would have been attributable to errors in specifications. The significance of the independent analysis actions is pointed out by the fact that the specifications had been reviewed by the developer and had been released, and discovery of the errors came as a result of the independent review.

Obviously, if the software specification contains an error with respect to the system software requirements, the correction will cost much less before design, coding and testing are accomplished than after. For this reason, early establishment of the independent agency is likely to provide greater cost savings.

*Some more proof*

One data point that may provide some insight to quantitative benefits of IT&A concerns the results of development, independent testing, and operational use of a highly critical, real-time control system. Several hundred errors were found by the developer in two versions of three separate programs and corrected during development and initial testing. Such a situation could be considered normal on any large software development.

However, sixty additional errors were found during IT&A, several of which would have been catastrophic. Since release to the operational agency, one error has been detected in one year’s use. This error was non-critical. As another single example, this instance would tend to support the effectiveness of the use of an independent agency for testing critical software.

*And more proof*

A 25,000 word program, which was integral to the range safety system for Vandenberg AFB missile launches in the 1960’s, was completed and turned over to the Air Force. This program was operational for approximately eight years with many modifications being made in response to changing requirements. At that time, an independent agency was charged to perform detailed IT&A of the program and 20 errors were detected, seven of which were critical. In another case a new range safety system was delivered to Vandenberg AFB. This system, after undergoing IT&A, was determined to be unready for operational use and subsequently had to undergo considerable modification.

*More proof needed*

Although Independent Test and Analysis has proven to be an excellent method of finding software errors, there is little data available on the completeness of this approach. There are several reasons for the lack of data: one being the added cost for performance of testing beyond that accomplished by the independent test group. Since such testing and resultant data are not available, a comparison of the software reliability (error-freeness) before and after independent testing is not quantitatively possible in this paper.

Another reason comparative data is not available is that IT&A has only been accomplished on highly critical programs. Therefore, emphasis has been on performance of testing and achievement of reliability on a program by program basis, not on the gathering of data for analysis of how well the testing was done.

Research into the overall problem of software reliability has begun to examine the various parameters associated with software development and testing which seem likely to affect reliability. One such study effort is being undertaken by the Rome Air Development Center for the Air Force. It is hoped that the results of this and other analyses will provide enough data so that quantitative assessment of the software reliability associated with the various testing approaches will be possible.

**INDEPENDENCE TEST AND ANALYSIS TECHNIQUES**

Analysis and testing techniques have also been evolutionary in their development. Early independent analysis consisted largely of “manual” code review, comparison of logic flows with listings and execution of modules whenever possible. These same techniques are still basic to the overall process, though many tools have been developed which automate many of its aspects. Examples of these tools are automatic flow charters, compare programs, logic/equation generators, structure analyzers and timing analyzers. Table II lists several of these with their associated characteristics.

Some of the more interesting work being accomplished in tool design and development is in the area of automatic verification systems, including test sequence generators, code segmenters, and path execution counters which gather statistics concerning program structure and path execution. Automated tools are now being developed and tested which analyze program structure and produce normalized flow diagrams which represent the actual program as if it were coded in accordance with the rules of structured programming. In addition, the test program then produces logic and arithmetic statements which represent the effect of the code execution.

Analysis and testing processes have become more sophisticated as experience has been gained and new tools designed. This is reflected in the increased use of simulation as a testing tool. Modern test configurations may use a closed-loop system involving six-degree-of-freedom vehicle models, environmental simulations, simulated system interfaces and, when actual flight computers are not available, interpretive computer simulations which run on general purpose host computers. These simulations may be modified forms of those used by the software
A third advantage of IT&A results from the natural effects of specialized experience gained from “doing the job.” Effective analysis of software documentation and code requires insight into requirements/specification relationships, methods of functional allocation, code analysis techniques, and simulation capabilities. These are “learned” aspects of the analysis and test process. When an independent agency is available that specializes in this type of work, learning time is reduced and effective effort begins immediately upon assumption of the task.

REDUNDANT PROGRAMMING

Another, and somewhat related, approach to reliable software is redundant programming (also called parallel programming and dual programming). This concept gets its name from the redundant systems used by the hardware engineers to increase hardware reliability. Here, two separate and independent programs, which can be executed independently to perform the same function, are produced by separate development teams; both programs are run at the same time on separate hardware systems or sequentially on a single system. In case of software failure of one system, the other system would take over.

A less drastic approach is to again have two programs, functionally identical, developed by independent programming teams. Here only one program would be used at a time and this could be parallel programming rather than redundant programming. One can be a back-up in case the other fails. Comparison of outputs can detect errors in one or both. Competition in development adds incentive for the production of a good program. Two knowledgeable groups are available for further effort in the area, eliminating the “sole source” situations. Two groups are more likely to establish a reasonable delivery time for the product than one working alone, again because of competitive pressure.

This approach also has the key ingredient of independence. Completeness of testing through the use of automated tools is not implicit in this method, though it certainly could be incorporated by both programming teams. The parallel approach is the result of philosophy that programs are too complex to be tested “adequately,” and that

developer or they may be designed and produced by the IT&A agency for its sole use.

WHY INDEPENDENCE

The greatest single advantage of IT&A over the more conventional approach of having the developer perform all software testing is that a new and different point of view is established with regard to the reliability of the software. Development personnel must hold to cost and schedule constraints with the fundamental goal of “making it work, within cost and on schedule.” Their philosophy is, and has to be, oriented toward showing that the software is correct, i.e., it performs all intended functions and performs no unintended ones. This is not the best frame of reference from which to detect software errors. The independent agency is brought aboard explicitly to analyze and test the developed software. His sole goal is to find errors and bring them to the attention of the developer so that they may be corrected. He assumes errors exist in the software and his whole reputation for thoroughness and effectiveness rests on his being able to find them. With this point of view, the independent analysis and test agency has much stronger incentives to find all errors than the unaided developer.

A beneficial side effect contributed by use of an independent group is the establishment of a competitive atmosphere in which the developer, knowing his work is being independently checked, strives to make fewer errors, while the independent agency tries even harder to find them. The professional pride of each group contributes to the building of the competitive environment and software quality benefits as a result of the interaction.

Another advantage associated with independence in analysis and test is that the tendency for management (always success oriented) to curb test efforts in the hope that the software is “good enough” is greatly reduced. When development and testing are under common management control, pressures usually exist which dilute the effectiveness of the analysis and testing function. An independent agency with a stated goal of detecting errors in the software being developed is in a stronger position with regard to management influence.

### TABLE II—Software Testing Tools with Associated Characteristics

<table>
<thead>
<tr>
<th>TOOL</th>
<th>PHASE USED</th>
<th>INPUT</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flowchart</td>
<td>Pre Execution</td>
<td>Source Code</td>
<td>Logic flow diagrams</td>
</tr>
<tr>
<td>Comparator</td>
<td>Pre Execution</td>
<td>Two versions of Source code &amp; software probes</td>
<td>Locations of all differences</td>
</tr>
<tr>
<td>Structure Analyzer</td>
<td>Pre Execution</td>
<td>Source code &amp; pattern</td>
<td>Locations of all occurrences of specified pattern</td>
</tr>
<tr>
<td>Timing Analyzer</td>
<td>Execution</td>
<td>Source code</td>
<td>Module execution times</td>
</tr>
<tr>
<td>Logic/equation generator</td>
<td>Pre Execution</td>
<td>Source code</td>
<td>Logic/Equations</td>
</tr>
<tr>
<td>Correctness proof</td>
<td>Pre Execution</td>
<td>Source code</td>
<td>Statement of equivalence or non-equality</td>
</tr>
<tr>
<td>Syntax Analyzer</td>
<td>Pre Execution</td>
<td>Source code</td>
<td>Syntax Error listing</td>
</tr>
<tr>
<td>Test Sequence generators</td>
<td>Execution</td>
<td>Source code</td>
<td>Test sequences</td>
</tr>
<tr>
<td>Path Execution counters</td>
<td>Execution</td>
<td>Source code</td>
<td>Execution data</td>
</tr>
<tr>
<td>Code segmenters</td>
<td>Pre Execution</td>
<td>Source code</td>
<td>Code segment designations</td>
</tr>
</tbody>
</table>

From the collection of the Computer History Museum (www.computerhistory.org)
matching comparisons of results from the efforts of two separate programs designed to perform the same function is a valid and sufficient test.

Overall, the advantages of parallel programming may outweigh the disadvantages, though the real question of how much this approach improves software reliability, has not been answered. Whether more independent testing on one program would produce higher reliability than normal development testing on two programs that are functionally the same is not clear. As software becomes more expensive and reliability more critical, both approaches may be used on the same development effort.

DISTRIBUTION OF SOFTWARE DOLLARS

In time of reduced budgets and spiraling costs, the concept of dual analysis and testing, independent or not, is likely to raise the budgeteering eyebrows, and when dollar cuts are inevitable, such activity may get immediate nomination for the proverbial “ax.” This is particularly true when management has not developed “software awareness.” Expenditures for software have been increasing dramatically, not only in an absolute sense, but relative to the associated hardware. According to the frequently-referenced Air Force Study, published in 1972, software costs were three times that of hardware and the ratio was increasing continually. Based on the labor-intensive character of software and the decreasing cost of machine processing capability, current inflationary trends will undoubtedly increase the software-to-hardware cost ratio even more rapidly than predicted.

The referenced study also reported that, as an average, the effort expended on software was divided into three phases on a percentage basis as follows: design and analysis, 40 percent; coding, 20 percent; checkout and test, 40 percent. These percentages have now become almost classic throughout the software development community. However, they refer in general to the development costs incurred by the software development contractor. The concept of analysis and test of software by an independent agency implies an additional cost of some magnitude, though it may have a wide variance with respect to

<p>| Table III—Relative Costs of Development and Independent Analysis and Test for Ten Programs |
|---------------------------------|---------------------------------|---------------------------------|</p>
<table>
<thead>
<tr>
<th>NAME</th>
<th>SIZE</th>
<th>DEVELOPMENT COST (MM)</th>
<th>INDEPENDENT ANALYSIS &amp; TEST COST</th>
<th>PERCENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>LARGE</td>
<td>750</td>
<td>252</td>
<td>33</td>
</tr>
<tr>
<td>B</td>
<td>LARGE</td>
<td>750</td>
<td>300</td>
<td>40</td>
</tr>
<tr>
<td>C</td>
<td>MEDIUM</td>
<td>337</td>
<td>142</td>
<td>42</td>
</tr>
<tr>
<td>D</td>
<td>MEDIUM</td>
<td>310</td>
<td>150</td>
<td>48</td>
</tr>
<tr>
<td>E</td>
<td>MEDIUM</td>
<td>112</td>
<td>37.5</td>
<td>33</td>
</tr>
<tr>
<td>F</td>
<td>SMALL</td>
<td>12.5</td>
<td>6.25</td>
<td>50</td>
</tr>
<tr>
<td>G</td>
<td>SMALL</td>
<td>12.5</td>
<td>6.25</td>
<td>50</td>
</tr>
<tr>
<td>H</td>
<td>MODIFICATION</td>
<td>125</td>
<td>50</td>
<td>40</td>
</tr>
<tr>
<td>I</td>
<td>MODIFICATION</td>
<td>650</td>
<td>450</td>
<td>69</td>
</tr>
<tr>
<td>J</td>
<td>MODIFICATION</td>
<td>300</td>
<td>112</td>
<td>37</td>
</tr>
</tbody>
</table>

The total dollars available for software development can be allocated in various ways. Assuming that reliability is a prime requirement in any software procurement, one should be able to add some amount to the developer’s contract and achieve an incremental increase in error-freeness. An alternative approach would be for one to take that amount and apply it totally to an independent contract. All intermediate variations are, of course, other alternatives. Which alternative is the most cost-effective, depends on several factors, such as status of documentation, uniqueness of system, and availability of test tools. Generalized relationships are depicted in Figure 2 which are believed to hold. In all cases, some level of reliability and dollar costs are basic as a result of the initial contract, i.e., a specific expenditure is indicated as a horizontal constant-dollar line in Figure 2. The curves indicate a level of specific reliability or error-freeness, and they show that by splitting the dollars spent to obtain significant software reliability improvement, one is more likely to achieve
higher reliability than by spending it on either the developer or the independent agency alone.

The curves are likely to be asymmetrical or skewed with respect to the 50/50 percent abscissa since several factors affected the specific cost relationships for a particular project. As an example, if the developer has not produced and maintained adequate documentation during the development process, an independent agency may have a difficult task when assigned to do analysis and testing. Funds for preparation of the documentation would likely be most effectively spent with the developer. The result might be the skewing of the curve for a specific level of reliability to the left. If adequate documentation is available and initial integration testing has been completed, the curve will probably be skewed right.

CRITICAL SOFTWARE IN THE BUSINESS WORLD

Computer software made its original debut in the field of accounting, and the greatest use of software still remains within the business world of commerce and industry. Military application of computers and their associated software began shortly thereafter and has steadily increased, not only in the business-oriented payroll and accounting areas, but more significantly, in the area of real-time control of complex equipment such as missile, satellite and aircraft avionics. The software in such systems frequently affects the safety of personnel and expensive equipment, and all reasonable steps are taken to insure its reliability.

Though personnel safety is not normally involved in the operation of commercial software, the dollar costs associated with software failure can be significant. This is particularly true in view of the increased size of business systems, the increased use of remote time-sharing terminals over widely distributed geographic locations and the trend toward control of fund disbursement directly by computer. Interactive systems with distributed time-sharing terminals require complex system software and failure modes can take many forms which may not be obvious to the system user or management. As the complexity increases, the risk of failure also rises, and with fund control being handled increasingly by machine, risk of fiscal liability will undoubtedly increase.

For clarification, the definition of failure may require some rethinking as it applies to business-oriented software. However, the concept that software fails if it does not perform its intended function or it performs unintended functions, should serve equally well in both the military and commercial world. With that definition in mind, software that writes checks to fictitious companies for services not rendered could be said to fail, at least as far as management is concerned. It may be that the software has been deliberately designed to produce such unearned checks. In this case, though it might be highly reliable in the eyes of the programmer, management undoubtedly would have a different view. Of the various approaches tried by the military software managers to insure software reliability, the one most suited to preventing the insertion of an "intentional" error is the utilization of an independent agency to perform an in-depth analysis and test of the program. In view of the increasing number of news accounts regarding computer associated fraud or embezzlement, and the difficulties that appear to be arising in the employment of computers in vote-counting, it is anticipated that independent analysis and testing of software and associated systems will soon become commonplace in many areas in the business world.

SUMMARY

Although independent test and analyses does not guarantee absolutely error free software, it has become an important approach to achieving highly reliable software in the Air Force. It has been used successfully on many critical programs particularly those involved with missile and space systems. The independent aspect of the function is the key characteristic and it offers the advantages of a new and objective point of view, freedom from unrealistic management constraints and the opportunity for the software manager to use personnel experienced in software testing.

Commercial software and computer systems have many characteristics which would tend to make independent analysis and testing attractive to the business world. Extensive application of the principle to critical commercial systems is anticipated for the near future.

ACKNOWLEDGMENTS

The authors wish to acknowledge the many informative discussions with James C. Alexander with regard to the history of operational software at Vandenberg Air Force Base. Acknowledgment is also given Ms. Barbara Avillanoza for her excellent typing and redaction.

REFERENCES


From the collection of the Computer History Museum (www.computerhistory.org)


PART III

INTERACTION WITH SOCIETY