Adding Formal Specifications to a Proven V&V Process for System-Critical Flight Software

Jon Hagar
Martin Marietta Astronautics Company
P.O. Box 179, M/S H0512
Denver, CO 80201
Voice: (303) 977-1625
Fax: (303) 977-1472
INTERNET: hagar@den.mmc.com
and
Dr. James M. Bieman
Colorado State University
Computer Science Department
Fort Collins, Colorado 80523
(303) 491-5792
INTERNET: bieman@cs.colostate.edu

Abstract

The process used to validate, verify, and test flight avionics control systems has produced software that is highly reliable. However, ever greater demands for reliability require new automated tools to improve existing processes. We use the Anna formal specification language and supporting tool set to develop a Test Range Oracle Tool (TROT) to automate the testing of equation execution. Our approach fits within the existing testing process and can increase the level of test coverage without increasing testing costs. The TROT approach introduces the use of formal specification languages and supporting tools to an existing industry program. This approach is being evaluated for expansion to other test support areas.

Keywords: Formal Specifications, Verification & Validation, Testing Process, Industrial Software, Test Oracle, Process Improvement

1.0 Introduction

Software application domains such as flight avionics control systems require extremely reliable software because failures of these systems can have severe human and financial costs. The industry has been successful in producing such reliable software. However, because of the critical nature of avionics software and business competition, we need to continually seek to improve the validation, verification, and testing process. Systems are becoming much more complex and insuring high reliability becomes an even greater challenge. One improvement is to increase the automation of the testing process. We have developed a technique using the Anna Formal Specification language and tool set that supports the creation of simple test oracles to check the correctness of equation execution [3]. These simple oracles are called Test Range Oracle Tools (TROT).

A major challenge in our development effort is to design a TROT approach so that it fits into an existing, very successful approach used in the validation and verification (V&V) of software used for flight avionics control systems [4]. Similar V&V approaches are used in numerous programs at various operating elements of Martin Marietta and other Aerospace companies. The V&V approach has been successful in producing a zero number of catastrophic software failures during actual use. **(**A zero failure rate does not mean software errors do not remain within the system or have not been encountered during software use, only that mission loss due to a software failure have not been observed.) The current approach includes functional, structural, requirements-based simulations, and software usage test techniques . These complementary techniques are applied to software in a maintenance life cycle mode. Using requirements expressed in an executable form has been a normal part of the process associated with both the functional and structural testing; for example, FORTRAN programs model parts of the required behavior of a guidance system. The use of requirements expressed in executable form and the actual software under test, essentially operates very much like an N-version programming environment that supports V&V [5].

We have recently begun trying to incorporate executable specifications that are expressed in a language other than the standard programming languages. We started with the Anna specification language because it supports Ada (a common language for many of our projects) and a supporting Computer Aided Software

Engineering (CASE) tool set is available. We have used Anna and its CASE set to produce a test support tool, TROT, which is actually a simple test oracle [1] that can judge correctness of software results.

The current version of TROT supports design level specifications that accomplish simple testing at an equation level. The TROT approach is being evaluated for incorporation within an existing and proven environment.

In this paper, we focus on the impact of the introduction of TROT to the existing processes. We first describe our current V&V process; outline the basic concepts of TROT; and then we describe how TROT is being incorporated with and improving our current V&V process. We also examine the TROT results that have been gathered during our evaluation of this approach. The paper examines some of the problems encountered during the effort to incorporate formal specifications within an existing software development process. And, since this is work in progress, potential uses and future efforts are also reported.

2.0 Current V&V Practice Methodology

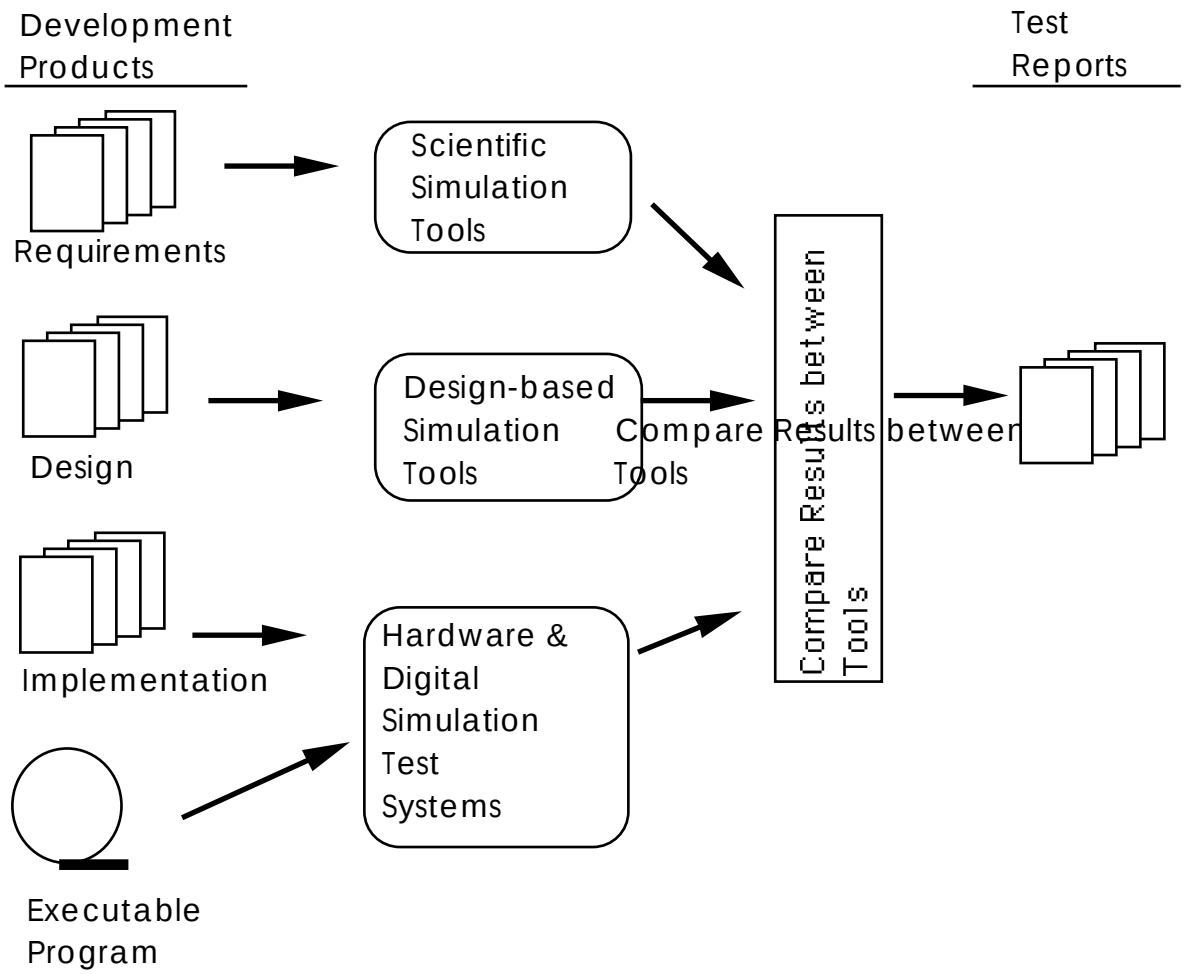
Our current V&V approach involves different levels of testing analysis. At all levels of testing we use a variety of supporting software tools and metrics.

The most basic testing level is structural-based verification testing conducted on a digital simulator or hardware system, such as an emulator. At this level, verification testing is done to assure that our code implements such things as design information and software standards. This testing is usually done at a module-level with small segments of the code being executed somewhat in isolation from the rest of the system. Tools executing FORTRAN-like code are available to support analysis of individual equations and simple logic structure. The comparison and review of results at this low verification level is human intensive.

The next higher level (called integration testing within industry) makes use of tools that are based on code structures that have been integrated across module boundaries. These are design-based tools and, at this level, they simulate aspects of the system but lack some functionality of the total system. These tools allow the assessment software for these particular aspects individually.

The next level is requirements-based simulation or what we call scientific simulation tools. These simulations are done in both a holistic fashion and individual functional basis. For example, a simulation may model the entire boost profile of a launch rocket with full 6-degrees of freedom simulation, while another simulation may model the specifics of how a rocket thrust vector control is required to work. This allows system evaluation starting from a microscopic level up to a macro-scopic level.

At the system level, we test software with actual hardware-in-the-loop. An extensive real-time, continuous simulation modeling and feedback system of computers is used to test the software in a realistic environment. Realistic here is defined as the software being tested as a "black box" and has the same interfaces, inputs, and outputs as an actual flight system. To test our real-time software system, we surround the computer with a first level of electrically equivalent hardware interfaces. We input signals into this test bed that simulate the performance of the system and hardware interfaces. The inputs stimulate the system under test which responds with the computed outputs. These outputs are read from the hardware of the test bed back into a workstation. The workstation software computes appropriate new inputs which are then fed back into the test bed. This arrangement forms a closed loop simulation environment that allows the software under test to be exercised in a realistic fashion. This is needed because the actual usage of the system is in space or is a flight environment such as zero gravity, that cannot be duplicated on the ground. In addition, unusual situations and system/hardware error conditions can be input into the software under test. The test system runs in actual real time, thus there is no speed-up or slow-down of the system.



2.0-1 Figure -- showing test tool levels.

A large number of the tools are simulations that are based on requirements or design information. The tools are stand-alone software programs (created by engineers) that can execute on separate platforms from the software under test. These tools take data that could be the input to the system under test, and produce expected outputs. These outputs can then be compared to results generated by the actual software being tested. Some of the tools simulate individual equations or logic sequences, while other tools simulate aspects of the entire system. Scientific simulation-based tools provide success criteria or analysis capability that allow engineers to judge the success of the software under test without relying entirely on human judgment. These are software testing tools that simulate various levels of abstraction (see Figure 2.0-1).

This system of simulation software tools and executing software under test form the equivalent of an N-version programming system. In one of our project areas, we have a minimum N = 3, and a maximum N = 8. Thus we compare and check points of at least N different programs and/or levels.

No in-use catastrophic software failures have occurred when we have fully applied our basic V&V approach. (Note: Some software errors have been seen but software redundancy, fault tolerance, and "safing" which are required as part of the software, protected the system). In spite of our success we are still looking for improvement. The current requirements and design-based simulation tools in use require human translation of English requirements into an executable program with the associated risks and costs of doing this.

Although successful, our approach has disadvantages including: humans must translate requirements into a standard executable programming language; no function exists within the programming language for automated compares; the current translation and compare processes are prone to error resulting in either missed problems or wasted time. Thus, we began to express requirements in formal specification languages. The introduction of these formal specifications in an in-use production environment and process is being done slowly and in an incremental fashion. Risk of unproven tools, methods, and languages can be offset by a slow evolution and process improvement.

3.0 Introduction of Test Oracles

Our current V&V environment provides some level of automation in testing. Tools to automatically determine the correctness of program output during testing can dramatically increase the level of automation [1 and 3]. Such tools are generally called "automated test oracles." An oracle is a tool or technique that will determine the correctness of execution results for input-process-output operations. Through TROT, we are investigating the use of formal specifications to produce test oracles that increase the test coverage levels, while reducing or keeping constant the "costs" of doing testing.

Using TROT, the correct execution of software at the equation level and statement level is automatically determined from a set of executable formal specifications.

Formal specifications define "what to do" requirements in a clear and unambiguous fashion. Specifications express information about a program that is not normally part of the program [6 and others]. TROT code, which is written in Anna, takes the form of a special test script that, once run through the Anna tool set and an Ada compiler, forms a test oracle. TROT specifications themselves are derived from an English requirements document.

Anna is a language extension to Ada that allows the formal specification of the intended behavior of programs [3, 4]. A supporting tool set that was developed at Stanford University [7] is available for Anna and has sufficient capabilities to support TROT. The Anna tool provides a fairly robust environment and is capable of transforming Anna constructs into executable programs. The basis of Anna is in-line code annotations. Code annotations (Anna code) are essentially comment fields within Ada programs. An example of an Anna code fragment is shown in Figure 3.0-1. Annotations are converted by the supporting CASE tool set into Ada code that can be compiled by a normal Ada compiler. Code with the Anna-Ada code inserted into it, is called a "self-checking" program, because each time an annotated code sequence is encountered, it is checked for correctness to the associated code.

```
  .....
    Ada Software associated with the Specification......
-- Names of months do not contain the letter x
--|   for all X : MONTH range JAN .. DEC  =>
--|      for all index_month : 1..MONTH'WIDTH =>
--|                                  MONTH'IMAGE(X)(index_month) /= 'X';
  ....More Ada Software.....
-- Initialization of a Variable
--|   B5_Guid_Update = 3.0
```

Figure 3.0-1 Anna Code Listing [7]

3.1 THE TROT SUPPORTING SYSTEM

The TROT supporting system that we developed has two major components: the test environment and the analysis environment (see figure 3.1-1).

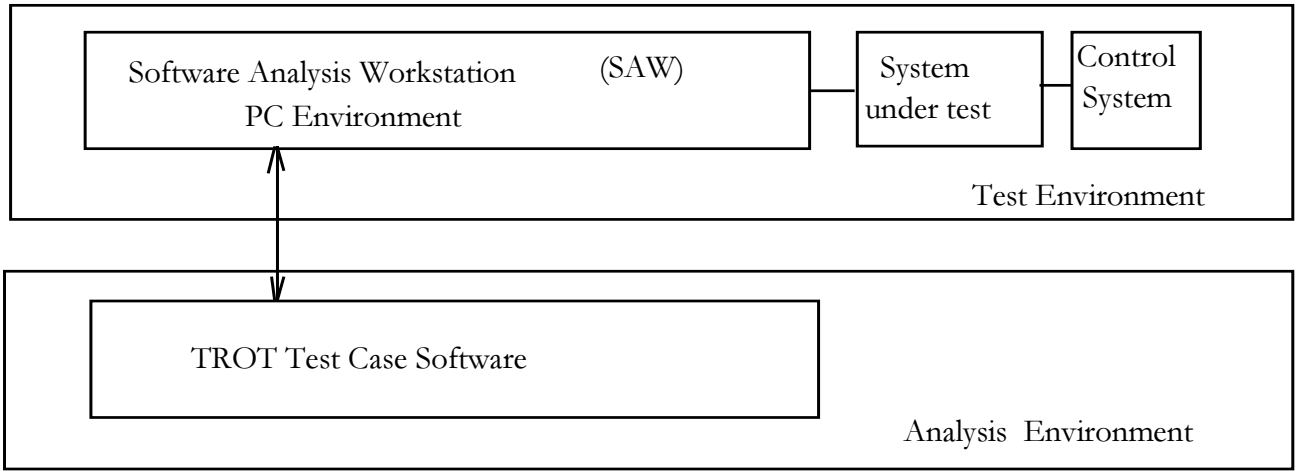


Figure 3.1-1 Major Components and Sub-components of the TROT System

In the TROT approach, the software under test is executed on a hardware-based test system that has low-level CPU monitoring capability. In our current prototype, a Software Analysis Workstation (SAW) system [2] is used to monitor the CPU

while a second computer is used to control the CPU of the software under test. This test environment is made up of hardware and software components that are detailed in [3]. Assembly language trace and result files are generated in the test environment on the SAW and transferred to a separate workstation analysis environment for post-processing by the TROT test case software. A trace file contains the following types of information:

- executed assembly language statements via a disassemble and non-intrusive hardware logic probe on the CPU bus;
- timing numbers (each statement is time tagged because the SAW's clock is faster than the system under test);
- memory addresses accessed and changed (memory inputs and output results); and
- computer status information.

This information is then available as an ASCII file for post processing by the TROT test oracle. A series of runs and associated traces can be made and retained in this fashion.

The TROT test cases are created from English requirements by a tester using the Anna specification language and some supporting Ada code. TROT test cases are run through the supporting Anna tool set to produce a program capable of automatically analyzing the SAW trace files for correctness. Correctness is defined in terms of requirement expectations when associated with inputs that were initialized in the software under test.

An executable TROT test case is a program produced from an Anna specification and supporting Ada libraries. The TROT test case starts from a standard template that:

- initializes files used in processing;
- reads in needed files;
- provides access to the variable being tested;
- provides numeric conversion utilities;
- provides the base Anna requirement that is being tested;
- provides a reporting mechanism; and
- handles exceptions raised by the Anna logic.

The tester also identifies the code variables that are under test as part of the Anna specification. TROT logic automatically determines if the specified conditions of a variable are met. The compare capabilities are a normal part of what the Anna tool set provides. For variables that pass the annotation's specifications, a nominal report is issued at the end of TROT test case processing. However, if an annotation is violated, an Anna exception is raised. These are handled by Ada exception handlers which can trap the violation and issue a failed test report.

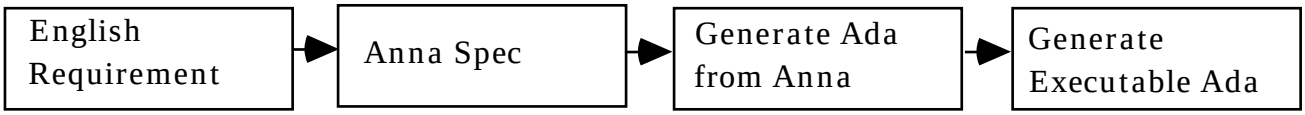


Figure 3.1-2 Production of a Test using Anna

Conceptually, the TROT process can be viewed as a form of N-version programming, whereby two or more versions of the software are created. However, in the TROT system, one version is "how-to-do" code and the other is "what-to-do" specifications. This is slightly different from the normal meaning of N-version program, where the two or more version of software are created by different people or groups, possibly using different system and languages. A crititism of normal N-version program is that typically all programs use the same requirements, and little or nothing is done (it terms of formality) with the requirements. And if there is an error in the requirement, all "Ns" suffer the same fault. In TROT, the "what-to-do" Anna specifications with supporting software are transformed by the Anna system into an Ada program that can check the functionality of the "how-to-do" code that is under test. To do this the TROT test program must be expressed with the required functions under test, and required constraints on these. This is an improvement over our current system since we express much of the requirements in a formal specification language rather than using a human process of deduction and interpretation which are needed during the normal programming process. Expression in formal specifications allow some formal reasoning about the specification, which the Anna tool set support, and we used in our evaluation. In the future, additional formal reasoning about the specification and verification by proof will be possible.

TROT code does not impact the timing and memory of the system under test. It is thus a not interfering method of test analysis. **The Anna specifications are feasible in a test environment because they do not have to function with the same timing and memory constraints as the software under test, and thus can be inefficient in these areas (i.e. it would not be suitable for actual flight use).**

4.0 PRELIMINARY TROT IMPACTS AND RESULTS

We have been in the process of evaluating and testing the current implementation of TROT on actual flight software. Because of the high level of confidence in the existing and proven methodology, new tools and techniques must be evaluated extensively before they can be incorporated and trusted. New techniques cannot be included in the proven methodology and applied to test new systems, unless we can show that the new techniques will generate correct results and detect errors Our initial evaluation involves re-execution of already tested logic, testing of previously found errors, error seeding, and step-wise incorporation of TROT-Anna prototype constructs. To date, we have evaluated TROT in testing initialization routines, several guidance equations, simple logic structures (e.g. “if”) and a math utilities. We have not intended Anna-TROT to test more complex things at this stage of development and incorporation of formal specifications into our process.

4.1 Results

This section summarizes our initial efforts and results. There were basically two phases of prototype evaluation. First was a period of learning Anna and some simple Anna programs. Second was a period were we implemented a integreate Anna with the SAW/PCTS system and performed a series of demonstrations. These two phases have been followed just resently with the production of a Anna-TROT system that was used to support an production test on actual flight software. This effort is only briefly touched on in this paper.

The first phase of Anna-TROT development was to understand and demonstrate the basic capabilities of the Stanford Anna system. After the Anna system had been installed and ported to a local computer system, a series of standalone programs

written in Anna and Ada code were generated/executed. Programs were written and results generated without interfaces to the SAW/PCTS system. This was done to help us understand the basic capabilities of Anna and its tool set, as well as demonstrating that the system was functional on our system. We did this with examples and regression sets provided with Anna as well as three test programs we generated. When this testing proved satisfactory and showed that Anna's tool set was working on our computers, the next phase was started in which we create interfaces between the Anna system and the SAW/PCTS.

It should be noted that Anna itself is a prototype tool set, and not a fully functional system, however the limitations of Anna did not significantly encumber our work.

Table 4.1-1 First phase of Anna-TROT Prototypes

Function Tested	Test Runs	Tool Success	Notes
Anna Regression test set	~ 300 sets	see notes	This set of tests demonstrated that the function needed by us were working, but showed some Anna functional areas were either not yet fully developed in Anna or functioning on our system. This was determined to be not a hindrance.
Variable Initialization	8	passed	Constraints placed on variables and tables being initialized to a constant value.
Numeric equation constraint	4	passed	Constraints placed on variables computed by a simple mathematical function with several inputs
Square root	8	passed	Constrains on a floating point math function, tests included 0 and min./max. numeric values
Sort function	4	passed	Numeric sort function tested with several different data sets, no erroneous results were tested (sort function/Anna Spec provide with Anna tool set)
Membership function	1	passed	Member of set function tested with no erroneous result (function/Anna Spec provide with Anna tool set)

Note: In each data set, at least two runs were made with Anna. Nominal as well as error conditions were executed. Success is defined as Anna Spec detecting a difference between code and the spec. This conditions raises an Anna-Ada exception, which can be propogated externally to the program or can be handled within the Anna-TROT program.

The second phase of Anna-TROT prototypes used Ada code, logic, and processes to access SAW trace data. SAW traces were generated by executing a segment of code of the software under test using the PCTS to initialize and control the target computer. The SAW captured the results and save this data to a disk file. This data could then be compared to the Anna specifications once it was transferred.

We first started with simple initialization routines that were captured from the SAW. This involved very simple Anna spec, support Ada program libraries, and short SAW traces. The Anna/Ada test programs did “pass” a series of five seperate SAW traces (corresponding to five different initialization routines. This was not particular interesting, but did demostrate the logic needed to read and process SAW trace files.

add example of the code here?

When the testing of initialization routines succeeded, we proceeded to test a square root function and a simple math equation that were present in the software under test (via SAW traces). This again was done by capturing SAW trace files of the software under test which had been initialized to know states. The SAW and state information was provide to the Anna-TROT test case for evaulation. For a series of

different data sets, TROT indicated the code met specifications, which was as expected since this code had be tested extensively and been shown to be very robust.

Once we had run a set test cases through the SAW, we became interested in testing the capabilities of Anna-TROT to detect erroneous results, which is what we are really interested in. But since the software under test did not have any errors of the type were interested in (as previous testing and analysis had revealed years ago), we “seeded” SAW results with errors, as if the software under test had generated incorrect results. The square root was tested by seeding SAW traces with resultant values that were just above, below, and within the required resolutions. During this testing, all seeded errors were detected and acceptable values were passed. This approach was repeated with a series of runs with the simple mathematical equation with similar results.

Once the error seeding approach had shown the feasibility of Anna spec to detect numeric computation errors, we the sought actual past code problems that had been found previously by V&V effort in our software under test. Since the flight code that we test is in the "ultra" reliable range, very few significant problems have ever been found by independent V&V which is conducted after development test efforts are completed. Documentation problems or problems that did not impact requirements or software code products were eliminated as candidates for Anna-TROT, since our approach is not intended to detect these kinds of errors. However, we did identify a set of problems that had been found by tools that TROT was being built to replace. We used these past faults to generated trace and results information for the SAW. These were then submitted to Anna-TROT programs for analysis.

As seen in the table, TROT detected most of these code based errors. One error concerned program-compilation error where an “IF” logic had been code incorrectly. The expected computation was not produced in the results for a given set of inputs, and this was detect by the Anna generated logic (an exception was raised). Another error concerned a variable that was accessed by a wrong index. This caused the code to access the wrong memory location. Again, Anna-TROT was able to detect this memory access problem. A third problem concerned a condition where the requirement and code did not match each other. This is a typical verification kind

of problem and was exactly the type of problem Anna-TROT was design to find. We are able to produce a run in which the results failed to meet the spec.

In should be noted, that in these past-history "error" based runs, we knew the problem and the "offending" conditions ahead of time (we had test cases and data the exposed them). While this kind of pre-ordained knowledge is not typical of the test process, we used inputs and conditions that were based on the earlier testing that had detected these problems when we reran with Anna-TROT. So, we do not believe there is any major bias. This also explains why we only ran 1 or 2 test case. Once one test case detects a problem, we did see the benefit to running additional cases (e.g. a tester experiencing an anaomaly detected by a TROT test case would have stopped testing and analyzed the situation to determin the source of the problem. It was sufficient to have one test case that demonstrated the detection of an error).

During the past errror based testing, we also ran a test where we used Anna to establish input Pre-condition specifications. And while this had not been part of our original intent with Anna-TROT, it did prove useful in flagging an error with input states. This use of Anna will be considered as an expansion in the future, based on this.

Table 4.1-2 Second phase of Anna-TROT Prototypes

Function Tested	Test Runs	Tool Success	Notes
Initialization function	5 modules	Passed	Exact equality required
Square root function	10 data sets from SAW traces	Passed	Values as calculated by software under test
	10 data sets with values exactly equal to high resolution requirements	Passed	Done with a seeded square root results
	10 data sets with values exceeding high resolution requirements	Passed (Error detected by Anna-TROT)	Done with a seeded square root results
	10 data sets with values exactly equal to low resolution requirements	passed	Done with a seeded square root results
	10 data sets with values exceeding low resolution requirements	Passed (Error detected by Anna-TROT)	Done with a seeded square root results
Simple Floating Point Numeric Equation	10 different data sets	Passed (SAW and Anna-TROT matched)	Input conditions/values were picked by engineering analysis at high/low and significant data points
Incorrect IF logic	1 data set picked and ran with SAW trace that contained the error	Passed (TROT name exception raised)	Old Code error reproduced
Incorrect memory index (memory leak)	1 data set ran and demonstrated the error in SAW	Passed (Anna exception raised	Old Code error with a specification of variable result range and resolution requirements
Guidance equation accuracy	2 data set ran and demonstrated the error in SAW	Passed (Anna exception raised	Old requirment and code problem replicated in Anna-TROT system
Data Input Precondition problem	4 data set (past test) was run with an Anna-TROT test that had a Pre-condition spec	Passed (Anna exception raised	Input range values were constrained based on other inputs
Divide-by-zero equation problem	10 data sets (data from original tests) were executed on the SAW	Failed (see text)	Anna-TROT failed over several data sets to find problem

A final test was done using a previously found error that involved a divide by zero condition. The original testing of this code by the existing V&V tools, had failed to find this problem. A time related function had been assumed to have a hardware-based sampling rate that prevented the function from ever returning a zero value. A divide by zero problem was discovered by "accident" during a complete hardware-software system test because it turned out under certain hardware conditions a zero was possible. Since the prototype TROT approach does not employ a random based testing approach and was being done with the inputs as the original test used, the same inability to find this condition exists. However, TROT could be used to support a random based testing scheme, employing a much large set of input conditions, and this should stand the same probability of detecting this type of error

as did the system-level testing that uncovered it by chance (note: since our process allows more analysis due to automation, more data sets could be processed in the same time as what we did by “hand” in the past).

4.1.2 Problems during Anna-TROT evaluation

One problem we have had is that our current software is "ultra" reliable. Even going back to the earliest versions of this software, the actual numbers of errors are relatively low. This makes the evaluation of TROT difficult against real problems and led us to error seeding. This situation also mean the collection of "real" test data and problems will be a very slow and time consuming process. This in turn slows the introduction of formal specifications into our existing V&V process.

Introducing formal specification techniques to the engineering staff has taken much patience. Formal specifications are a very new ideology to them. Staff using Anna required very detailed training to help overcome this problem. In addition, management and the customer needed convincing that this new approach would save time and money while achieving equal or better levels of software testing. Our efforts to solve management and the customer's perception problem have been aided by the prototype and evaluation approaches.

Also, the Anna tool set, while remarkably robust for "academic-ware", has had problems. These problems include some features (not used by current TROT versions) that are not fully implemented; some supporting tools which we could never fully make operational; and some functions that were misleading. The tool set is a possible candidate for re-engineering and upgrade to support a full industry environment, however, it has been sufficient to generate interest and support our evaluations, to date.

Although the introduction of TROT has been a success so far, several risks remain. The risks include: "scaleablity" of the TROT concept to higher levels; continuing engineering staff acceptance; reliability of the Anna tool set and TROT test cases; the ability to automate additional aspects of the current TROT approach; and the application of the TROT concept to other programs and software areas. These remain topics for additional work.

4.1.4 Potential Benefits and Generalizations

Our initial testing and evaluation suggests that the TROT system can replace the existing design to code, low level verification tool. TROT should be able to:

- achieve the same or better coverage levels in testing;
- offer more automation;
- run tests on the actual hardware versus digital simulations;
- base tests on formal specifications that are derived for requirements/design information (which is at a higher level of abstraction); and
- avoid using any software instrumentation or intrusion.
- **support formal reasoning about and verification of the requirements themselves. (more?)**

The errors we detected extend to both code and simple logic errors as well as problems in requirement range/resolutions. Also the increased rigor in test development proves useful in better testing. For example, during the square root testing, reviews of the tests done using existing tools and processes exposed deficiencies in the old test cases. These were simply added and accomplished using the TROT version without any extra time in TROT test development. Expanded future versions of TROT seem to offer more coverage of requirements and design verification features.

Generalizes? or does this repeat the conclusions?
A stand alone executable formal spec system where the software under test and the oracle are executed sperately, even on different computer system, should be reliazable in other test environments. This should prove useful in testing other embedded and real time computer systems where the “overhead” of self checking code and embedded formal spec are too great. We used a generalized spec language (Anna) and commerical systems (the SAW). And while some customization of software was needed, these were done in a library structures which could be easly replace, updated, and improved. Substitues for hardware, software, and languages are quite possible. We have complete use of this system in a very simple production test, and are now working to expand this approach to another new test environment which is currently under development.

This is another demo that formal Specs can be used to generate test oracles. Test oracles will be of great benifit to the testing community, who have been looking for ways to improve the test process.
Formal Spec and oracle can be introduced into existing industry software process. Our experience is that this will have to be slow and done in small steps. Not introducing all the functions and features that normal spec might offer right away.

TROT has also proven useful in finding problems in requirements. Many of the current English specifications are incomplete or have "derived" requirements information. During the design of several equation-based TROT test cases, these requirements were identified and clarified. Clarification involved understanding the resolution provided by the actual code and then "backing" this into a requirement. This was a simple out-growth of the detail needed to write an Anna Specification. In our case, the additional information was noted with the development or test information and did not result in a code changes, but does contribute to improved longer term maintenance of the software.

4.2 TROT's Future

Our initial evaluation of TROT is based on a small set of test cases with very simple Anna constructs. This evaluation is limited, in part, due to limitations in the current TROT prototype [3] and our concern about quickly changing our existing V&V process. Thus, we plan a slow implementation, evaluation, and then implementation of an expanded TROT approach.

We have completed a TROT module and used to test a minor revision to flight software. The minor revision concerned a set of Boolean operation. Anna-TROT was able to analyze the production code and compare to a specification. The test and code while simple was the first production use of Anna-TROT in our testing. Future expansion of this work is planned.

We are developing plans to use TROT to evaluate a future software system. Future systems will have unknown errors and TROT will be used to verify design requirements to code implementation. This verification will be done at the equation level as well as expanded levels. Integration of commercial products, for example Cadre, T, AdaTest or AdaCAST, with the Anna tool set is being actively pursued.

We also plan to assess the impact of TROT to our existing V&V system. The quantitative analysis will be based on:

- time to prepare a test case, and
- time to analyze test results.

Once we have the above measures, we can make comparisons to historic numbers from the verification tool that we are replacing. We will begin collecting data when we have actual new production code. Until then, evaluation and certification of TROT will continue to take place in preparation for its production use. We also expect that the use of formal specifications will increase the level of coverage in testing while reducing or keeping constant the "costs" of doing testing.

Once TROT is in use, we will look for improvements in our V&V process resulting from the introduction of formal specifications. Additionally, we anticipate improvements in our test process due to the use of formal specifications because they allow more expressive power than our current programming languages. This expressive power will allow our test and system experts to use TROT to formulate specifications that our current V&V techniques may have assumed or left as "derived" requirements. Such "unwritten requirements" can be formalized in TROT-Anna so that the test process can include them. Thus, TROT-Anna will support complete testing of the system and support the improvement of requirements. This should increase system reliability.

While the current TROT prototype is limited to range & resolution checking and limited path checking, we have shown that the basic concept is workable and that the Anna language and TROT system has potential for expansion. Our current approach relies on automation, human experts, and classical functional/structural testing. This approach has demonstrated the ability to detect errors before a system is used in actual production, producing error free (catastrophic) software.

Fully working Anna or other systems?
We are considering the application of additional advanced error seeding techniques, such as mutation analysis, before more advanced production versions of TROT are completed.

5.0 CONCLUSIONS AND SUMMARY

The TROT system is a feasible, non-interfering, formal specification-based test system, which can be used for automated verification that an implementation meets requirements. Our approach is unique in that we are applying formal specifications to an ongoing test process used to test software that is in ongoing maintenance mode and that must be highly reliable. TROT prototypes have shown a way to incorporate formal specification techniques in an industry setting which had not previously used formal methods as part of a test program. The tool supports faster and more complete testing based on requirements/design information. Initial results support the conclusion that TROT can support an existing verification and validation process.

The initial TROT prototype is designed only to test variable range and resolution requirements of equations and math functions. TROT and the supporting tools are currently completing development to support actual product testing. Additionally, we are planning on TROT upgrades to add functionality to accommodate other requirement structures. The expansion of TROT and the long term evaluation of its impact on our ongoing V&V process is a natural outcome of this work. Because of the high reliability nature of our test environment and software under test, we can include new techniques like TROT slowly and only as part of an orderly transition.

The TROT system is part of ongoing efforts to identify improvements to our existing V&V system. These improvements include:

- General hardware and software system upgrades;
- Test tool processing automation (of analysis results like TROT does); and
- Test process improvements including formal methods and test data selection techniques as in [8].

We plan to upgrade and continue to evaluate TROT. This evaluation will include quantitative and qualitative measures of TROT's impact on the existing test process. We expect to see improvements in test capabilities due to the expressiveness of Anna. We also expect to see improvements in time-to-test costs because of the increase in automation (or self analyzing) capability of the TROT system. Thus, we expect more reliability in the software, because more testing will be possible without additional time or effort.

REFERENCES

1 Bieman and Yin, "Designing for Software Testability Using Automated Oracles", Proceedings International Test Conference, pp. 900-907, September 1992

2 CADRE, Workstation Environment Software, Manual Package - 02584-04033, CADRE Technologies Inc. 1988.

3 J. Hagar, J. Bieman "A Technique for Testing Highly Reliable Real-Time Software", Proceedings of Software Technology Conference, April 1994.

4 J. Hagar " A Systems Approach to Software Testing and Reliability", Proceedings of the 10th Annual Software Reliability Symposium, June 1992

5 J. Knight, P. Ammann, Testing Software using Multiple Versions, SPC publications, June 89.

6 David C. Luckham, Friedrich W. von Henke "An Overview of Anna, a Specification Language for Ada", IEEE Software, IEEE, 1985, pp. 19-22.

7 David C. Luckham, An Introduction to Anna, A Language for Specifying Ada Programs, Programming with Specification, Springer-Verlag, 1990 .

8 Tim Raske, "More Efficient Software Testing through the Application of Design of Experiments (DOE)", submitted for publication in 1994 ISSRE.