LESSONS LEARNED FROM INCORPORATION OF COMMERCIAL COMPUTER AIDED SOFTWARE ENGINEERING TOOLS IN A FLIGHT CRITICAL SOFTWARE TEST ENVIRONMENT

Jon Hagar
Lockheed Martin Astronautics
P.O. Box 179
Denver, CO 80201
jon.d.hagar@lmco.com

ABSTRACT

In flight critical, software intensive, avionics systems, a major technical and managerial component is the testing and analysis of the developed software. In many of these systems the software must be "ultra-reliable" (work the first time and every time) and produced within schedule and budget. In this paper, we will examine how an existing successful software verification and validation project incorporated commercial computer aided software engineering (CASE) tools. The paper examines the original approach, how CASE and new tool concepts have been incorporated within this approach, and some observed impacts to cost and quality. A final section identifies the challenges that were faced during development of the test system.

1.0 INTRODUCTION

Software testing is an area of software development that can occupy from twenty-five to fifty percent (or more) of software development costs. Additionally, Independent Verification and Validation (IV&V) costs can equal those of development testing. Digital avionics system do not lend themselves to the ad hoc beta testing of the "shrink wrap" industry. Therefore, comprehensive software Verification, Validation (V&V), analysis, and testing must be done. Comprehensive V&V costs a great deal of money, and so with the current interest in better, faster, cheaper, one large area to improve is software testing.

There have been numerous approaches conceived to improve software and software testing. One idea is just to eliminate or reduce the amount of software testing done. This can be disastrous, since developers of software have not found a way to produce error free software, and software testing is a risk management activity for software errors. It is not acceptable to have avionics systems with software errors which result in the failure of a system. Another approach can be seen in the Cleanroom [1] approach to software, which changes the fundamental way software is produced and tested. While clean room and other approaches like it represent interesting approaches that fundamentally change how we do software development, the definition of what really works and clear technical superiority has not been proven. Further, issues of acceptance by the software engineering staff of these fundamental changes can be insurmountable. Yet one more approach has been automation and the addition of computer aided software engineering and CASE test tools.

CASE tries to maintain similar (or better) levels of testing, but with reduced costs and time because of automation. Test automation is nothing new in software, as most software testing has always involved some degree of automation. What is new in the last decade or so is the availability and use of commercial test programs. In the past, automated tools were custom made for specific programs or companies. The applicability and longevity was thus often limited to the people using them. But, during the 1980's and 90's, vendors began offering commercial CASE tools. Vendors often advertised test tools as a way to easily save projects and software efforts. As with much of CASE technology, there are no "magic silver bullets". CASE tools do not take bad software and make it good, nor poor development processes and allow them to make good software. What we have learned is that good processes, practitioners, supporting tools, and resources (time and money) are all necessary for success.

This paper examines how an ongoing flight IV&V product area has taken existing processes, practitioners, and supporting tools, then incorporated CASE test tools to first develop and then start an avionics software test program. This paper covers the test system, CASE tools incorporated, and leassons learned. While the program is not complete, early feedback from testing analysis indicates that the introduced commercial tools are supporting program goals and customer needs while achieving reductions in cost. While this paper is about software, much of what it addresses is applicable to system and hardware levels testing.

2.0 CURRENT V&V PRACTICE

Our current IV&V approach involves different levels of testing and analysis. Our test approach spans from the software unit level to integrated hardware and software (a

system). Additionally, we have issues of configuration control, change management, documentation, and management. In all of our test activities, we use a variety of supporting software tools and metrics. The goal of our V&V is to show that flight software is ready for use and the chance of catastrophic mission loss due to software can be considered acceptable by the user. The software under test is the guidance, navigation and control software of a booster system, and so test programs that involve the fully integrated system are not possible e.g., we can not fly the software in a "beta" test on an actual rocket system to see if it will work in real use.

2.1 What We Test

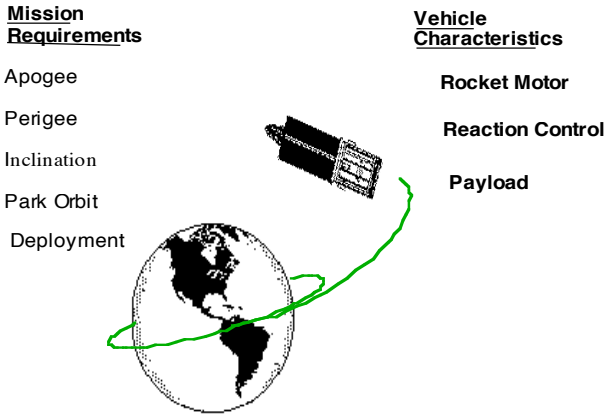
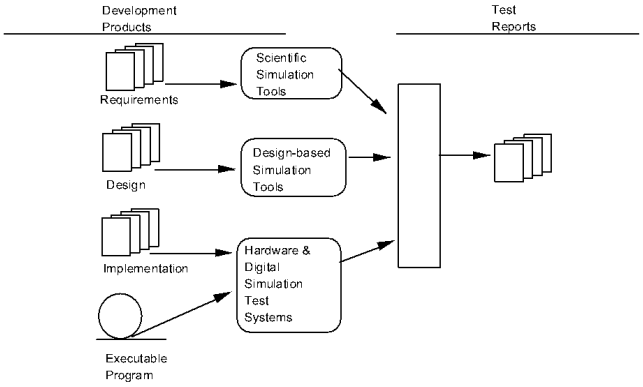


Figure 2.1-1 Complex Avionics Software Requirements

Lockheed Martin Astronautics (LMA) in Denver, Colorado has produced critical software systems for several decades. Production systems are usually one of a kind that must work the first time or hundreds of millions of dollars may be lost. These systems are typically very complex, consequently failures or errors could be introduced from many sources. These software-systems have the following characteristics: real-time; spacecraft/booster flight control; minimal human intervention possible; and numerically intensive calculations of such critical items as, trajectories, flight dynamics, vehicle body characteristics, and orbital targets. Development programs are small—under 30,000 source lines of code (with small staffs), yet these programs are critical to the control and success of the flight system. Avionics systems with software produced at LMA include the Titan family of launch vehicles, upper stage boosters, and spacecraft, as well as the associated ground systems. An example mission profile is depicted in Figure 2.1-1. The software addresses both mission related requirements as well as hardware/system related characteristics. Production of software on many of these systems followed a historic and similar development process that has been, in part, responsible for mission success at LMA.

2.2 Historic V&V Process and Tools



2.2-1 Figure -- Tested Products - Tested Products

Our product area's testing tools simulate various levels of abstraction (Figure 2.2-1). In our approach, the lowest testing level is structural verification testing conducted with a digital simulation or hardware system, such as an emulator. At this level, verification testing is done to ensure that executable programs implement such things as requirements, 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. A tool executes code in a simulator to support analysis of individual equations and simple logic structure. The comparison and review of results at this low verification level was human intensive.

The next higher tier, called integration testing within industry, uses tools that are based on code structures which 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 of software for 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 on an 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 macroscopic level.

At the system level, we test software with actual hardware in the loop. An extensive real-time, continuous digital simulation modeling and feedback system of computers is used to test the software in a realistic environment. Realistic is defined here as the software being tested as a "black box" with 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 to simulate the performance of the system and hardware interfaces. The test system runs in

actual real time, thus there is no speed-up or slow-down of the system.

Numerous tools support the tiers and many of the tools are simulations based on requirements, design information, or the computer architecture. The tools were stand-alone, custom built software programs that executed 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 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. Tools were often not well integrated, so data had to be analyzed by hand or reentered.

Overall this approach and tool set has been successful in taking input development products, doing V&V, and generating test results (reports). In spite of success, we look for improvements in our processes and tools to save time and money.

3.0 WHAT HAS BEEN ADDED FROM CASE

The current business environment most industries operate in requires such things as continuous process improvement. While our product area has been in existence for almost twenty years, we have practiced continuous change, adopting what is new and works, retiring what is antiquated, and learning what is new but maybe does not work as well. We followed with interest the introduction of CASE tools that assist in testing. We tried some CASE technology and learned from other's efforts.

During a major new system upgrade, we determined that it was worth the effort to convert many of our supporting tools to either CASE based or take direct advantage of CASE information. We outline these in this section.

3.1 Modeling Tools (Requirements Simulation)

To support determination of how requirements should behave and establish requirements based success criteria, we model the software system. These models are executable, meaning they can take input and produce some output. Input and output both may be for parts of the system or the whole system level.

Recently, we have moved from specialized FORTRAN-based modeling tools to those based on MATLAB (TM). The advantages of MATLAB are that it:

1. supports quicker development of models and comes with extensive "tool boxes",
2. is a standard that many engineers are now familiar with;
3. allows easier interface to several systems (platform independence); and
4. interfaces easily to our test data, much of which is in telemetry streams.

MATLAB is a commercial programming environment, that supports rapid development and reuse via "tool boxes" and easy engineering user interfaces. Tool boxes are callable routines that support reuse. MATLAB comes with a large library of vendor supplied routines that support sophisticated analysis with graphics. In addition to commercial libraries, many engineers on our project have developed tool box routines that are specific to our problem domain, but may be reused in several tools and/or even projects. Reuse has been advantageous both in limiting time spent on tool development and by providing analysis options that never existed before because of development expenses. We have noticed engineering "trading" tools and planning for reuse during tool development.

Additionally, since MATLAB is commercial, it runs on a variety of platforms. This allows us to develop and run on a micro computer, and then move it to a more powerful workstation when we want quick turn around. The diversification of tools on all of our computer resources allows better use of what can be a costly resource (computer equipment). Further, since turn around time is better, engineers can be more productive and even have better attitudes (engineers seem to hate waiting).

Due to the commercial nature of MATLAB, many LMA engineers now have a working knowledge of the tool. Most new college graduates have also used it. This shortens the learning curve and associated costs of training engineers. Also, the familiarity leads to new ideas for tool improvements since we have a "critical mass" of people that generate ideas from each other.

Finally, MATLAB was designed to process streams of numbers, and a stream of numbers is basically telemetry. While it was necessary to create routines to decode telemetry, this has not proven difficult. Further, because of the tool box approach, reuse of telemetry-based processing has proven possible. Once decoded, the steam of numbers serves as input to the tools. Also, we find we can do automated checking for test success. In this approach to analysis, we first compute what a function should generate then pull what the software actually computed out of telemetry. Then the tool compares the two to determine that the system is working as required. We are currently adding to our "checks" as we mature our test analysis. We expect continuing returns on this test analysis as we move to first and recurring flights.

3.2 Reverse Engineering Tools

A large part of testing is understanding the software. Another is the verification of design to code information. Both software understanding and verification are supported by the use of so called reverse engineering tools. A variety

of commercial systems take code and create design pictures. We are using Battlemap (TM) by McCabe and Associates. These design pictures are used to

1. verify developer supplied information;
2. aid engineers in their efforts to understand how the software is operating;
3. provide a variety of metrics which measure attributes and identify areas of code requiring added tests; and
4. feed design information into other commercial packages (see 3.3 and 3.4).

We are finding this category of tool advantageous in both aiding and accomplishing tests. However, the tools were found to have limitations particularly in dealing with embedded cross-compiled program environments. Tools provided incomplete support in measures such as test coverage and indication of "real" complexity. Also we found that metrics can both mislead and overwhelm. Most of the tools can produce hundreds of metrics. Care must be exercised when "understanding" what they measure. Finally, we found tools must be used in combination to provide a complete test environment.

3.3. Unit Test Procedure Generator

We have two levels of test generator tools. One assists in the generation of test harness (see 3.4) from reverse engineering tool information. The other type of tool actually helps generate the test procedure documentation needed for our project. The tools are different in function, however, both fit under the generator designation.

The first tool is really a series of tools, custom shells, and integrated products that work together to generate a unit (module or routine of code) test harness. We first take the code to be tested and reverse engineering information about it (section 3.2) then feed that into a requirement and design-based CASE tool, Cadre-Teamwork (TM), which testers enhance with test requirements and data dictionary information. Once the refined test pictures have been produced, the CASE tool with some supporting shells, generates an initial unit test harness. The harness is then compiled and executed to complete the test (see section 3.4)

The unit test process has some manual efforts which we will continue to improve. The test generator does offer an automated way to do what once was a manual process, as well as providing improved test results documentation. For example, test reviewers can now obtain on one or two pages what used to take ten to twenty. One area of disappointment has been in the level of initial coverage provided by the CASE tool. We had hoped the process would generate sufficient data sets to allow complete coverage (statement and branch) with the first harness generated. But, we have found that by both a lack of tool functionality and a lack of developer supplied data, only about ten percent of coverage is realized on the first pass. Additional human engineering is needed to reach coverage.

Another innovation we have realized is the use of tools to generate our test procedures. Test procedures are written documents that define such things as, test objects, references, special considerations, execution steps, analysis and success criteria, and ultimately, pointers to results. We became interested in ways to speed up the generation of test procedures, test execution, and documentation as well as approval of these by management and quality assurance people. We were also looking for ways to take advantage of our large scale corporate and project computer network. We felt that electronic softcopy versus hardcopy might have a variety of advantages.

This led us to the world wide web (WWW) and hypertext markup language (HTML) which are supported by commercial network tools such as NETSCAPE (TM). Using these tools, we created a system that allows the generation of executable test procedures [2].

Various HTML templates and forms have been created that allow first the generation of specific test cases, and then the execution of tests using the completed HTML-based test procedures in an on-line, interactive mode. This approach highly automates and standardizes test procedure generation and execution. Tests are designed by filling in HTML template forms which in turn generate test procedures in the form of other HTML documents. The basic process is seen in figure 3.3-1. All test inputs and outputs, shell files used, and the STEP itself, are stored in the specified location for review and analysis by engineering after the run.

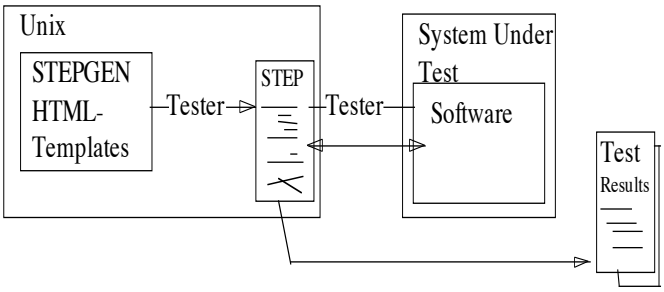


Figure 3.3-1 STEP Generation Process

As fields are entered, the system "builds" a new HTML file ("STEP" in figure 3.3-1) that is itself an executable test procedure. In order to allow the user to process smaller pieces of the test procedure, the tool offers the feature of partial submittals at the end of logical sections of the test procedure. The user can then exit the browser and begin generation of the later test procedure sections at a subsequent time. This supports test design over a number of sessions or designers. Custom and pre-designed procedures can be mixed freely with order independence.

When completed, test procedures are submitted for electronic on-line review and approval (peer, management, and quality assurance review). Once approved for use, the test procedure file can be executed via the Web. Test procedures prompt testers (test engineers) for actions interactively and spawn actual test tool execution, as well as providing both the input and output retention.

We have found that use of Web technology has numerous advantages, some of which are listed below.

1) Breaks test generation and execution into smaller more manageable pieces.
2) Utilizes hyperlinks to quickly and easily view input, output, and analysis that exists online.
3) Provides a standard test procedure form used by all groups, adding consistency to the testing.
4) Provides an intuitive and easy to use user interface
5) Browsers are device independent and allow multi-platform use with consistent results.
6) Use of an "electronic only" option helps eliminate paper and encourages a paperless office.

While this system works and is useful, it was a learning experience setting it up. Issues were raised early on by management about the security of data and whether this was more a system for "play" than work. Both of these have been addressed by standard company policies and properties of the Web tool we are using.

3.4 Unit Level Test Harness Tool

Testing units of flight code at LMA is a rigorous activity, since we must ensure the absence of certain unit level kinds of errors that might be catastrophic if they occurred. In the past we had specialized tools and a great deal of human effort to do such things as complete statement and branch coverage testing. We have incorporated a commercial package, AdaTest (TM), that supports direct unit testing with what is called a test harness system. This system allows the testing, automated checking, and reporting of unit level tests. The automation required a different unit level test process, but achieves the same and better results. Before we had to hand code a test language for a module, then debug it, generate a test, run the test, and then analyze the results. The new system is driven with an integrated process across several commercial tools. We can quickly get the basics of a harness and set of inputs from the combination of tools. This is then added to reach the complete levels and check our test procedures require. We are creating a library system of test inputs, cases, and results, which will be archived for regression tests.

Regression tests required new test scripts and test, and could only be built in a limited fashion on past results. The new system will solve these because we will reuse the harness(es) from initial testing by modification of only the "changed" code checks.

3.5 Online Documentation and Information Control Aids

A major issue for testers is that information (files, documents, tools, programs, data, drawings, etc.) used in testing must be easily and quickly accessible, correct and current [2]. Modern software systems have numerous levels of documents (requirements, design, code, data, executable, and test) all of which must be managed efficiently. Providing the configuration management information, controlling the files (e.g., write protected), and accessing the data are all necessary for success.

We have established informational pages for each product configuration under test. Within these pages are links to correct files and documentation. These links are defined and used by testers but maintained and controlled by our internal quality assurance group. This separation allows better control by quality assurance which in turn supports better review and audit by the quality assurance group. The system is online and easy to use for quick support of project test documentation needs.

Some of the activities in this area that previously involved manual transfer, review and input have been automated with the use of the Web. This information control automation using the WWW tools has resulted in:

1) timely electronic notification to engineers and correspondents;
2) improved status identification that in turn is accessible to other WWW based tools;
3) improved configuration control because of the server; and
4) improved test design and execution, since tools have direct access to Web information on the same system that they are executing.

4.0 LESSONS LEARNED & ADVANTAGES

In the discussion on the kinds of tools we have taken advantage of, we defined some of the tool specific lessons learned and advantages. This section summarizes some general level observations that our project has had during the initial set up and use of the CASE tools.

1) Training - The importance of and allowing for (time and money) training is important. Some tools require training (it is usually provided with their purchase). Training for our project been both formal and less formal—on the job.

2) Planning - CASE tools must be planned for and developed like any software effort. CASE tools are not "plug and play". To be successful, we planned for, developed, integrated, and tested our CASE tools, supporting software, and processes.

3) Thinking "outside of the box" - For existing engineers, it is easy to want new tools to be like old ones, consequently

engineers and specific activities must change- which can be difficult.

4) Determine the real requirements - Since new CASE tools did things differently when we first started, we did requirements definition followed by trade studies to determine what functions we really needed tools to perform. This resulted in selection of tools and CASE implementation efforts that were successful. Look and feel may not be real requirements.

5) Usability of a tool must be reasonable - While tools will need training and by nature have complexities, a tool that is too hard to use or is constantly in revision by vendors, leads to a frustration by users that in the extreme will lead to "shelfware". The user interface was part of our selection evaluation before purchase.

6) Engineering acceptance - Engineers can get tied to their "favorite" tool. They are slow to use another tool that they are not familiar with. This leads to some tools labeled as "Bill's" or "Ed's". And you hear things like "I am not going to use Bill's system. Get him to do it". This issue of acceptance and large scale use relates to training and management commitment. It takes time to learn anything that is complex (and most engineering tools have some complexity, otherwise they would not be engineering tools). Management has to allow for this and keep focused that "It is your job to use Bill's tool."

8) Expect some failures and learn from it - We explored several tools that we abandoned after an initial period of time. While failure is not good, it is really only total failure when one does not learn from the mistake. We have a continuity of people and processes such that when some "piece part" idea does not work, we still have overall success. This requires planning and attention to what is happening, because if you do not know, you have failed or can not say how something failed, then lessons learned will not be possible. Also, management must avoid blaming engineers for the failure of an idea since this stifles future ideas.

9) Process is important - CASE tools must fit within your process. Lack of process but just having tools will probably result in failure.

10) People are important - CASE tools by themselves do nothing. People with training and knowledge are needed to make tools work.

11) Avionics system have special problems - Despite progress, CASE tools do not totally solve all test problems in digital avionics systems. We have found problems in cross compiling, embedded applications, design, data representation, and requirements engineering, as well as other areas. These can be worked around but that means vendors have more functions to add.

4.1 Cost and Quality Impacts

When compared to custom developed tools, establishing our CASE environment has taken 50 percent less people than before. We have realized this savings while maintaining functionality though things look and feel different. Further, we expect less (50 percent reduction in budget) maintenance costs since vendors provide upgrades for a low annual fee (relative to staff costs). We have the disadvantage of not being able to add functions we want to tools, but this has proven a minor issue.

Additionally, we have reduced our test production staff by 40-to-75 percent (based on several past test cycles). We believe we will reach similar levels of quality testing and error detection rates though this remains to be seen since we are only several months into our test activity. Early engineering testing indicated our error reporting rate to be equivalent to a time when we used custom tools. Ultimate quality will be determined in long term use of the system we are testing.

5.0 SUMMARY

CASE tools can be good if one considers them as tools and not "magic bullets." People make tools work and people do the hard parts of engineering that tools cannot do. Tools should aid people—not be a replacement for them. Also, tools should fit within one's requirements and process. Modification of process procedures is unavoidable and having people that can change with the newness is important. While CASE tools are now viable, there is a lot of room for improvement in CASE environments.

6.0 REFERENCES

[1] Dyer, The Cleanroom Approach to Quality Software Development, Wiley & Sons, New York, 1994.

[2] Hagar, Burba, Wittekind, Bell, "HTML and the Web", Proceedings of 9th International Software Quality Week, 1996.