Presentation
Paper
Bio
Return to Main Menu

P R E S E N T A T I O N

W14

Wednesday, May 3, 2000
3:15PM

GRAYBOX SW TESTING IN THE REAL WORLD IN REAL-TIME

André Coulter
Lockheed Martin

International Conference On
Software Testing Analysis & Review
May 1-5, 2000
Orlando, FL, USA

Graybox Testing

Graybox Software Testing in the Real World in Real-Time

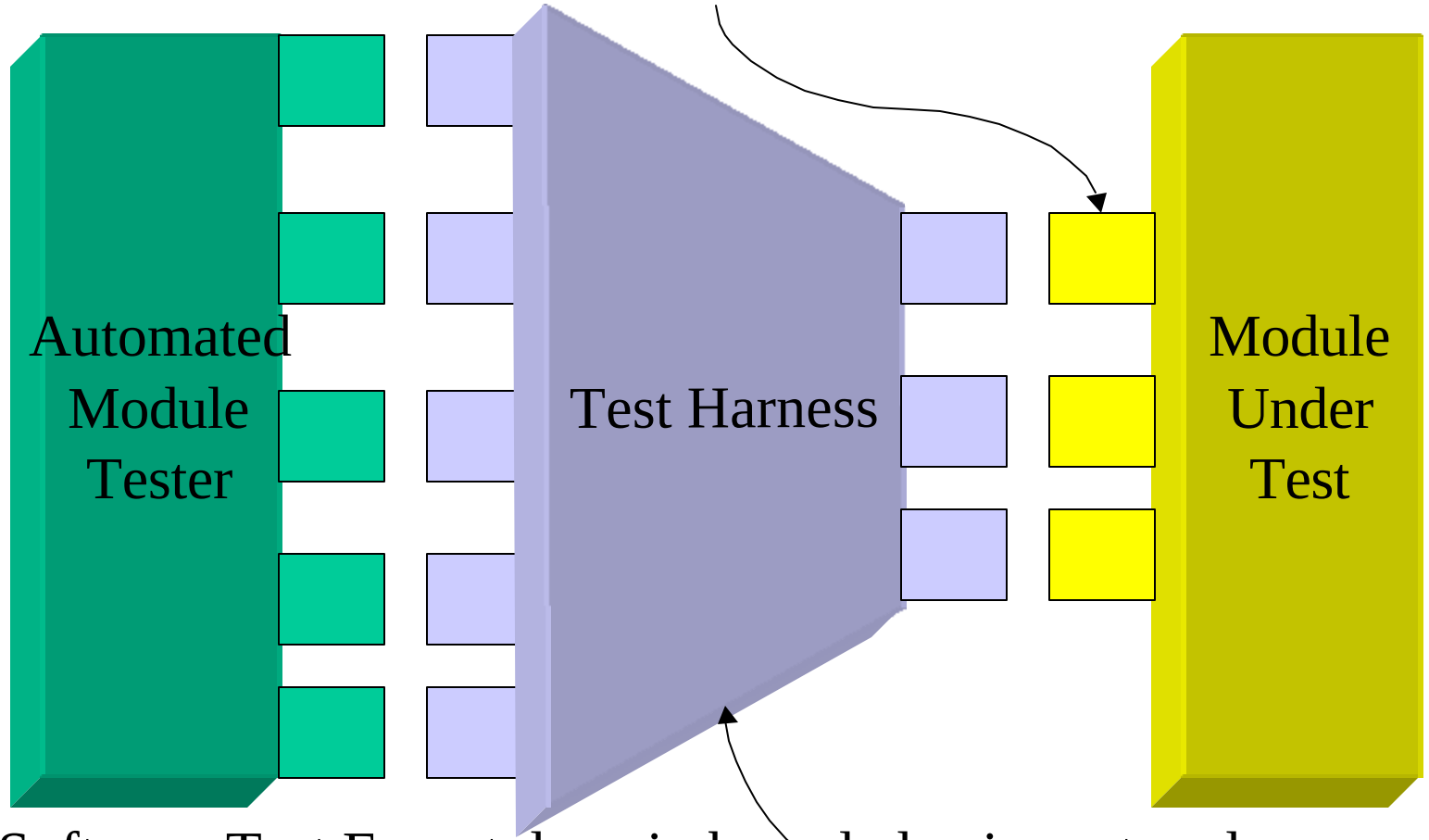
André Coulter

Graybox – Data Capture

- The Graybox methodology involves software/hardware testing using a-priori Systems behavioral modeling data to determine the expected results of the system.
- Given a system with a known state and input data, it should be possible to predict the next system state and its output data.
- A system with 5 modes (off, standby, norm-op, test, calibrate) can be tested to verify proper state transitions.

Graybox Testing – AutoTester

Test harness created from the MUT's specification

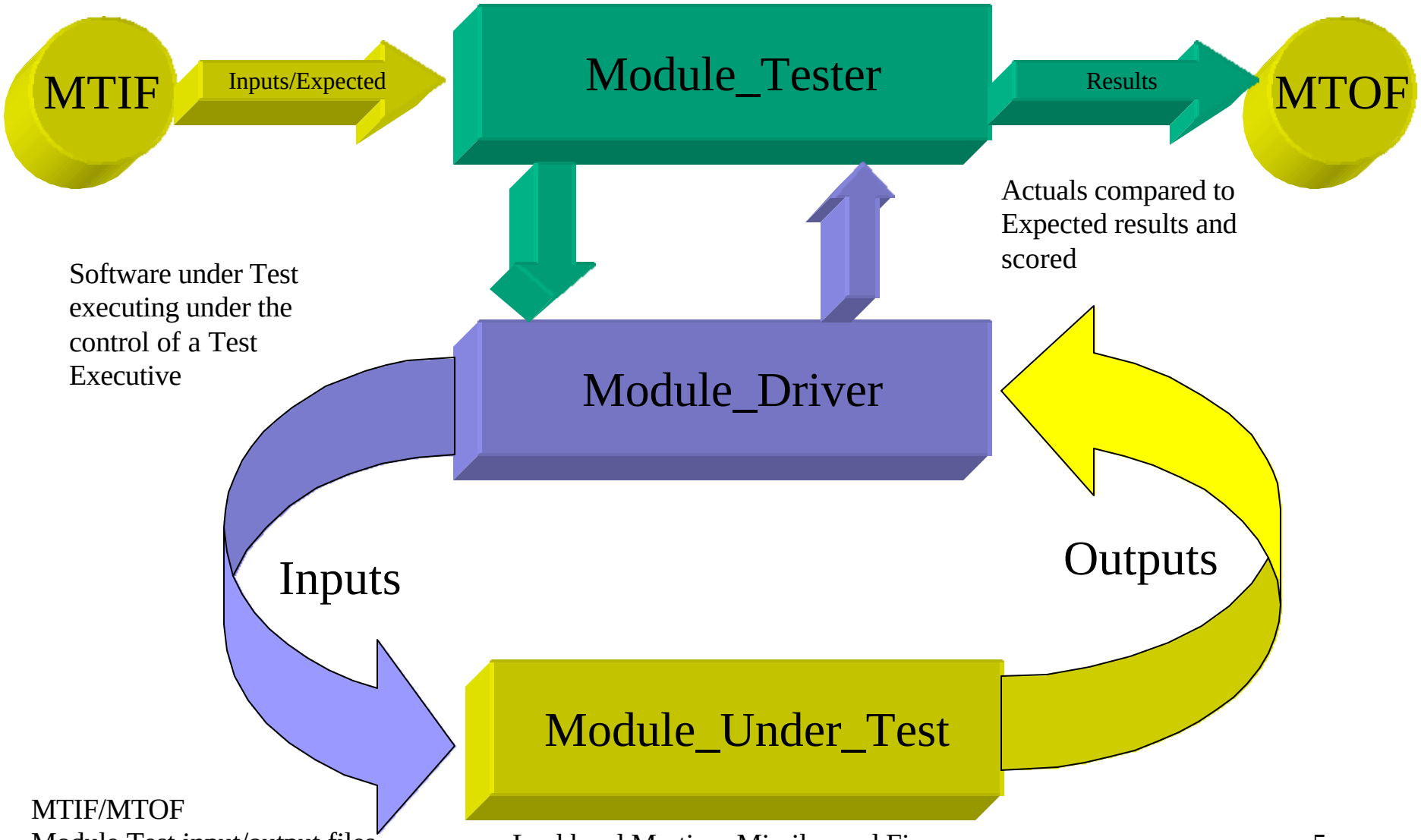


Software Test Expert domain knowledge is captured

Graybox Testing - Modal

- The valid state transitions are:
  - Off => Standby
  - Standby => Norm-Op, Test, Calibrate, Off
  - Norm-Op, Test, Calibrate => Standby
- The System in the Off mode can only transition to the Standby mode.
- The System in any mode must first transition to the standby mode before going to a new mode.

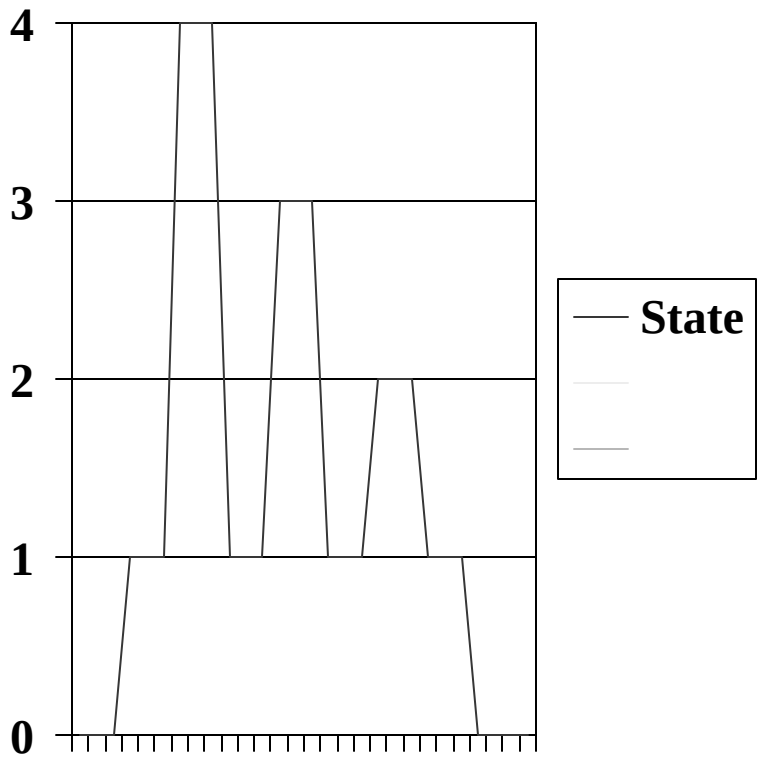
Graybox Testing – Module_Tester



MTIF/MTOF
Module Test input/output files

Graybox – System Moding

- Discrete state data
  - Off Mode (0)
  - Standby Mode (1)
  - Normal Op Mode (2)
  - Test Mode (3)
  - Calibrate Mode (4)
- System transitions to a commanded mode and remains in that mode until a new mode command is received



Graybox Testing - Modal

- A Graybox test consists of setting up an input condition and verifying the system correctly responded.
- During a modal test, the system will normally remain in the commanded mode, therefore time is not a factor in testing.

Graybox – Modal Test

- Modal Test
  - Verify state = Off
  - Transmit state = Standby
  - Verify state = Standby
  - Transmit state = Test
  - Verify state = Test
  - Transmit state = Off
  - Verify state = Test

Graybox Testing – Non Real Time Vs Real Time

- During Modal testing, time is not important.
- During Operational Data testing, time is important. Data is constantly changing over time.
- Operational data captured during normal system operation requires a definite sense of time. A radar system tracking targets must calculate the a/c position in real time.

Graybox Testing – Non Real Time

- The original concept of Graybox testing provided a method to verify embedded software during the Code and Unit Test phases of a project.
- The methodology utilized no concept of time during testing since the software was executing in a debugger.
- To expand the Graybox methodology into the formal test arena a definite concept of time was needed.

Graybox Testing – Time Domain

- Time Domain
  - Non Real-Time (NT), Passage of time is not important, event driven system.
  - Near Real Time (NRT), Passage of time is important but simulated by frame clock. Used when performing Real Time simulations.
  - Real Time (RT), Passage of time is important. Time is real. Execution of the actual system.
  - Soft Real Time (SRT), Hard Real Time (HRT)

Graybox Testing – Real Time

- Real Time
  - Local Time
    - Internal CPU clock time
    - Local Frame Time (1 tic = 100ms)
    - Object Level Time (OLT) starts at the object's birth!
  - Master System clock time
    - ZULU time
    - IRIG time.

Graybox Testing – Operational Data

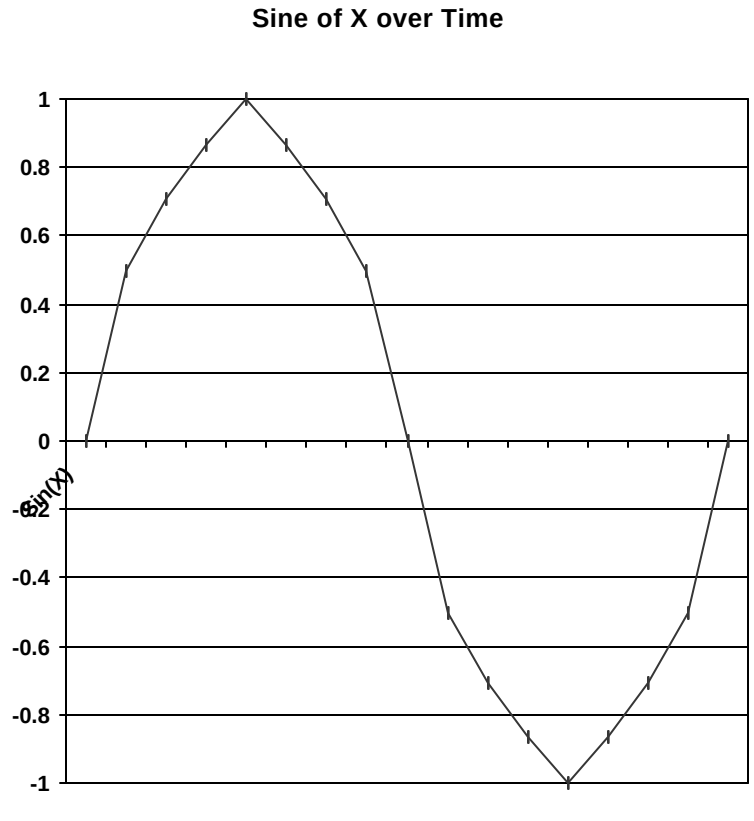
- Operational Data is collected and time stamped at the object level.
- Object level time stamping involves dating an object based on the amount of time the object has been alive.
- Data extraction and verification can now be based on the life time of the object vs actual wall clock time.
- Object Level Time stamping allows tests to be repeated in real time.

Graybox Testing – Object Level Time

- Object Level Time in the Graybox sense is fuzzy time.
- To verify an Object has the correct value at the correct time, an object is first requested to return its lifetime value, then the object request value, followed by the lifetime value.
- The object is then tested to verify a value was returned that is within the starting OLT and the ending OLT.

Graybox Testing in Real Time

- Capturing data in real time requires a knowledge of time
- Each data point must be time stamped.
- Each data point must be compared to the expected data point in the time dominion.

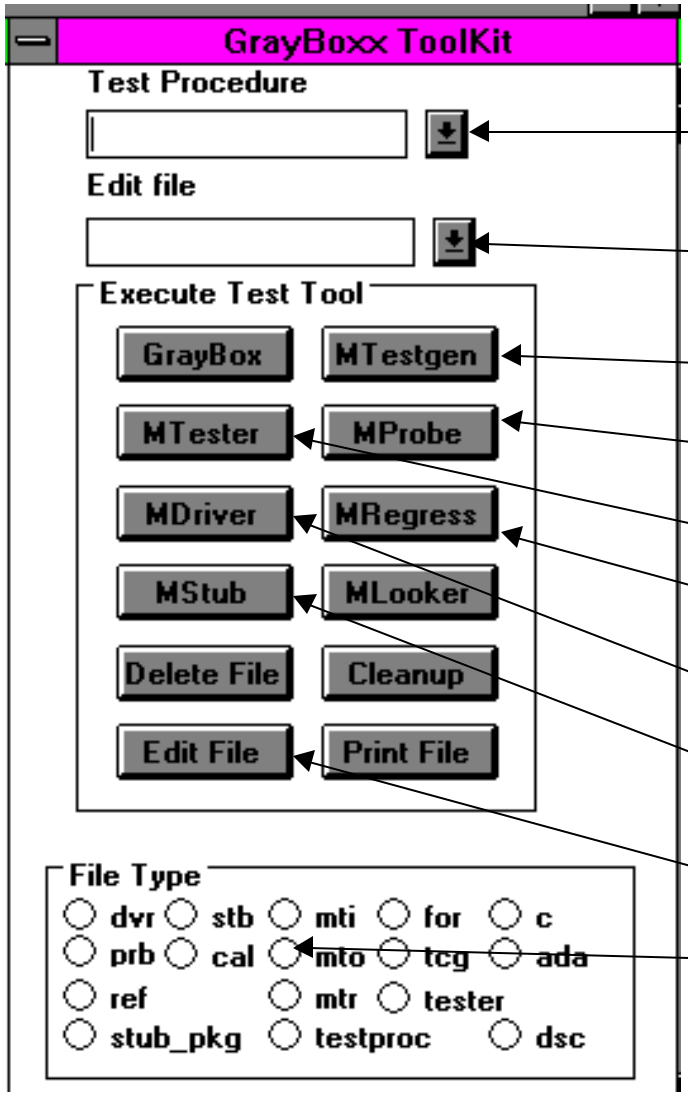


Graybox Testing in Real Time

- Systems A-priori Data in Table
- Captured Data
  - Object Start_Lifetime = 3
  - Object Value = **0.866**
  - Object Ending_Lifetime = 5
- Data is validated if Object Value falls within Beginning and Ending OLT times.
- All Three values are acceptable
- (0.7071, 0.866, 1.0)

FrmTm	OLT	Value
110	1	0
120	2	0.5
130	**3**	0.707
140	4	**0.866**
150	**5**	1.0
160	6	0.866
170	7	0.707

Graybox Testing - ToolKit



Testing as easy as point and click

Identify a module to be tested

Identify filename of the module

Generate a test case file

Instrument module for coverage

Execute module in the tester

Generate a regression test file

Generate a test driver

Stub incomplete modules

View results after execution

Graybox Software Testing in the Real World in Real-Time

André C. Coulter
Lockheed Martin Missiles and Fire Control - Orlando

Abstract

The Graybox Testing Methodology is a software testing method used to test embedded systems. The methodology is platform and language independent. The current implementation of the Graybox methodology is heavily dependent on the use of a host platform debugger to execute and validate the software under test. Recent studies have confirmed that the Graybox method can be applied in real-time using software executing on the target platform. This now expands the capabilities of the Graybox method to include not only path coverage verification but also worst-case / best-case path timing. The Graybox toolset can now be called upon to perform and verify performance requirements. It is now possible to verify/validate CSCI/CSU/Module/Path timing, functional and structural requirements in a single test case with the same format used to verify/validate functional requirements. The Graybox methodology is a full life-cycle testing methodology that enables software developers to test embedded systems in non real-time or in real-time. This paper will present the Graybox methodology and how this method has been applied in a real-time environment to validate mission critical embedded software systems.

1.0 Introduction

The computer industry is changing at a very rapid pace. In just a few short years we have seen an industry where access to a computer was limited to a select few individuals in white lab coats to one where almost every employee in an office complex has a computer on their desktop. This explosive trend has been fueled by both hardware and software advances. Computers are becoming an indispensable item for every product being developed.

In the aerospace industry, computers and their corresponding software control every aspect of a product from its concept and design, to its development, manufacture, and test. "Software as a product item refers to any computer program, whether it is an operating system, a single subroutine, or a microprogram". [Spencer85] In order to keep pace with a rapidly changing computer industry, software testing must develop methods to verify and validate software for all aspects of a product lifecycle. Software engineers must have tools that will enable the rapid and accurate testing of both functional and performance requirements.

2.0 Scope

Graybox Software testing in Real-Time seeks to provide a method of testing software that will be both easy to implement and easy to understand using Commercial Off The Shelf (COTS) software and hardware. The Graybox methodology is defined as "Blackbox testing + Whitebox testing + Regression testing + Mutation testing". [Coulter99] The modified Graybox method will address real-time performance testing.

The original Graybox methodology did not address real-time or system level testing. The major concentration of the methodology was to provide a system that allowed test engineers to create ample test suites that thoroughly tested a software product. The emphasis of this paper will be to include real-time performance verification and validation in the methodology without altering the intent of a Graybox test. As "recently as 1998, in one Fortune 100 company, performance testing was conducted while one test engineer sat with a

stopwatch, timing the functionality that another test engineer was executing manually". [Elfriede99]

The Graybox method enabled the engineer to state the inputs and the expected results in terms of system requirements. The engineer can state the system performance requirements along with the functional and structural requirements to form a complete and thorough test of the deliverable system. "NASA stated that on-board software reliability must be such that, during the entire shuttle program, not one vehicle is lost or damaged beyond repair because of software problems". [Jelinski72] In order to achieve this level of reliability the software must be thoroughly tested at both the functional and performance levels.

Applying performance testing earlier in the project testing lifecycle will give confidence that the correct product will be delivered and operates correctly at the performance stated in the requirements. Testing of performance characteristics must be moved closer to the beginning of the development cycle. "If equal emphasis was given up front to testing, then I believe that good code would follow, because the things that we recognize as good in coding are the same things that contribute to easy testing". [Beizer84]

3.0 Computer Systems

Computer systems can be placed in one of many categories. The categories are normally based on the type of processing required and the speed at which the user expects a response. Batch or non real-time systems require a minimal amount of human intervention and are heavily dependent on the amount of time spent determining the answer to a problem. As the requirement for human interaction increases or the amount of time spent on solving a problem is reduced, computer systems move into a realm of real-time processing. "Man machine interactive analysis and processing are preferred to batch processing for more efficient analysis". [Hanaki81]

As the systems engineer moves into the real-time realm, new and interesting problems arise. The computer system must not only produce the correct answer, but also must meet strict timing constraints that do not exist in a batch-oriented non real-time system. "Real-Time processing implies dedicated, special-purpose computers often running many processing elements in parallel". [Onoe81]

This multi-tasking, multi-computer environment presents special challenges when testing computer software that must meet specific timing requirements in this hybrid environment. "A single CPU cannot afford to deal with various kinds of image data or various processing phases in an interactive multi-user environment". [Isaka81] The original Graybox technique requires the stopping and starting of the computer system to verify internal code coverage and functional requirements. "It is usually desirable that a real-time system should operate without stopping". [Martin67]

4.0 Graybox Software Testing

The Graybox methodology is a ten step process for testing embedded computer software (refer to Table 1. Ten Step Graybox Methodology). The methodology starts by identifying all the

input and output requirements to a computer system. This information is captured in the software requirements documentation.

Table 1. Ten Step Graybox Methodology

Step	Description
1	Identify Inputs
2	Identify Outputs
3	Identify Major Paths
4	Identify Sub-function (SF) X
5	Develop Inputs for SF X
6	Develop Outputs for SF X
7	Execute Test Case for SF X
8	Verify Correct Result for SF X
9	Repeat Steps 4:8 for other SF
10	Repeat Steps 7&8 for Regression

The Graybox methodology utilizes automated software testing tools to facilitate the generation of test unique software. Module drivers and stubs are created by the toolset to relieve the software test engineer from having to manually generate this code. The toolset also verifies code coverage by instrumenting the test code. "Instrumentation tools help with the insertion of instrumentation code without incurring the bugs that would occur from manual instrumentation". [Beizer84]

By operating in a debugger or target emulator, the Graybox toolset controlled the operation of the test software. The Graybox methodology has moved out of a debugger into the real world and into real-time. The methodology can be applied in real-time by modifying the basic premise that inputs can be sent to the test software via normal system messages and outputs are then verified using the system output messages.

In real-time, data must be captured, logged, time-tagged and then compared to a-priori data generated by a simulation that has knowledge of timed data. "The timed entry call is one place where an upper bound is placed on the time duration for some action to occur". [Volz88] Specifying the time at which an event must occur and verifying that the output is correct with respect to time verifies the performance requirement.

The software test system must have some concept of time. There are several timing systems that are available in a real-time system. The test engineer could use the CPU clock as a timing source. The frame counter is another source for timing data. The last alternative is an off-CPU chip system such as an IRIG timer. Once a timing system is adapted, the data must be time tagged and compared to the expected results in the time domain.

An alternative to any absolute timing generated by an on-board or off-board system would be an Object Level Timing (OLT) system. OLT involves generating a relative time from start of an object's existence. Using an OLT reference allows repeatability in the testing of real-time systems. OLT is "a methodology for the statement of timing requirements". [Coolahan88]

5.0 System Simulations

Automated Test case generation has always been the desire of software test engineers. "There is no simple way to automatically generate test cases". [Beizer83] The best and most efficient method of software test case generation can be realized by inserting specialized test data capture code into an existing system simulation.

In the aerospace industry, specialized computer simulations are generated to develop algorithms and software requirements. Existing simulations can be interactive or batch systems, but the timing information is based on a frame counter that is used to simulate time. Generating test case data from the simulation enables the test engineer access to precise output data timed to a specific event. The "speed and accuracy of performing various tasks" can be verified by analyzing the time tagged output data. [Spencer85]

Any product developed must meet two sets of expectations. The first expectation is that of the developing company. If a company has internal standards that must be met, these expectations will be tested by the test organization long before the customer ever sees the delivered product. The other set of expectations belongs to the customer. "The successful product must be satisfactory both to the user/customer and to the producing company". [Spencer85]

Performance requirements must be stated in quantitative terms. The output of a message must be generated before some elapsed time interval based on a measurable/observable event. The measurable event can be the receipt of an input message or the transition to a particular state. Performance requirements stated in terms of events and interval timing can be verified by observing the OLT of a specific message object. "Performance measures and requirements are quantitative – that is, numbers. A performance specification consists of a set of specified numbers. The systems actual performance is measured and compared to the specification". [Beizer84]

Care must be exercised in the generation of test case data. In a real-time system it may not be possible to verify the entire contents of a program as is the case when executing in a debugger. The test engineer must selectively determine the best candidates for data capture to achieve the best performance and requirements coverage. "We should discipline ourselves not to collect data aimlessly; rather, we should collect data by planned experiments". [Grenander72]

The Graybox methodology involves software proof of correctness. Once test case data is obtained and the system is exercised using the simulation generated test case data, the output is verified against the expected results. Care must be exercised in the dependence on the simulated results. A computer system verified with a simulation with corrupted algorithms will only verify that the algorithms in the simulation have been faithfully implemented in the real-time system. "It is only too easy to take the impressive output of a simulation as fact, when the input does not justify this. The possible error of input must be translated into terms of possible error of results". [Martin67]

6.0 Graybox Testing in Real-Time

Applying the Graybox methodology in real-time is simply adding a time component to the expected output data value. A normal Graybox test for a system that generates sines based on angular displacements would consist of the input angle in degrees and the expected output sine value. The test "y = Sin(x) for x = 30" would be 0.5. This test has no time component associated

with the answer. Therefore whenever the response is generated it would be compared to the expected result of 0.5. This could be 1 millisecond or 10 seconds from the start of the test. In a real-time system this test would be unacceptable.

A real-time test would be stated as "y(t) = sin(x,t)". The added time element "t" assures the expected value will be compared in the OLT domain. Using Figure 1. Sin of X, the major Object is X. A component of object X is the x.sine. The OLT values of x.sine are read from the curve. The OLTs are obtained from the X-axis. Therefore at the creation of the X object, the value of x.sine should be 0 at X's OLT (x.olt) of 1. At the x.olt of 10 the value of x.sine should be –0.5.

During the execution of the simulation, one "step was to examine the raw data with the intention of determining trends. An additional step for examining raw data was to display them using a plot package". [Feeley72] These steps will help the engineer visualize the data being collected and can be used as a method of reducing the amount of data being extracted in real-time. "Software instrumentation while giving precise and copious information, cannot be inserted into a program wholesale lest the system spend all its time processing artifact and instrumentation rather than doing honest work". [Beizer84]

During a real-time test, the system is examined for the values of x.olt, x.sine, x.olt. This time-stamp/value/time-stamp group is logged and then post-processed immediately after the test to verify that the time-stamped value lies on the curve between the beginning and ending time stamps. "The processes of logging, counting, timing and sampling can be done by software: by inserting code". [Beizer84]

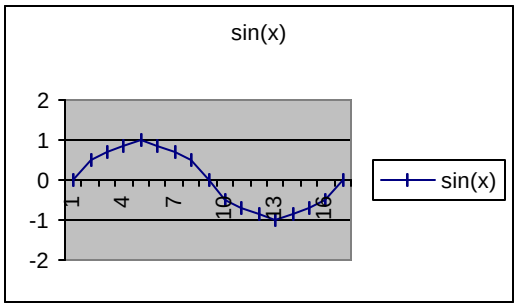


Figure 1. Sin of X strip chart

In real-time the Graybox methodology consists of the five steps presented in Table 2. Five Step Graybox Real-Time Methodology.

Table 2. Five Step Graybox Real-Time Methodology

Step	Description
1	Identify all System Performance Requirements
2	Instrument System Simulation for Object Level Time data
3	Execute System Simulation to Obtain Test Case data
4	Monitor Behavior of Real-Time Program
5	Verify Correct Values based on Time-Stamps

To perform a real-time Graybox test all test data must be prepared before hand from the system simulation and fed to the real-time program as input system messages. "The online load generator takes transactions from the scenario file and attempts to issue them to the tested system at the time marked for that transaction". [Beizer84] Because the precise time for object creation cannot be duplicated for each run, the OLT will

guarantee that the object data is consistent from run to run.

7.0 Conclusion

Depending on the nature of the application, a real-time system falls in one of two major groups. "There are two types of real-time systems, namely, soft real-time and hard real-time systems". [Cheng88]

Soft real-time is defined as a real-time system where it is desirable that system processes occur rapidly, but few if any real time constraints are placed on the system. A requirement might state that the system respond within a reasonable time limit. This could be interpreted by one system engineer as one second and by another system engineer as one millisecond.

A hard real-time system places response times and process time constraints on a majority of the system processes especially at the system interfaces. "Hard real-time systems are characterized by the presence of tasks that have timing constraints". [Zhao88]

To verify real-time software, a system level simulation is required. The simulation is used to generate the test case data that is fed to the real-time software. The output messages are extracted, time-tagged, logged and then compared to the expected results in the time domain. When the expected results are successfully compared in the time domain, the real-time performance characteristics can be verified. "Similarly this technique will be useful for verification and validation of programs". [Enomoto81]

By adding the element of time a non real-time Graybox test can be converted into a real-time test capable of verifying and validating real-time embedded applications.

Author Biography

Andre' Coulter is the Tools and Technology Leader at Lockheed Martin Missiles and Fire Control - Orlando. Mr. Coulter has over 22 years of software development experience and has spent over 12 years investigating and developing automated testing tools in support of Lockheed Martin embedded systems applications in Ada, Java, Perl, Fortran, C, and assembly language. Mr. Coulter is a graduate of Bowie State University, Bowie, Md. in 1978 with a BS in Bus Admin. He also taught computer programming at Drake State College in Huntsville Ala.
Email andre.c.coulter@lmco.com

Bibliography

[Beizer83] Boris Beizer, Software Testing Techniques, 1983 Van Nostrand Reinhold Company Inc., New York, pg 67

[Beizer84] Boris Beizer, Software System Testing and Quality Assurance, 1984 Van Nostrand Reinhold Company Inc., New York, pgs (238, 258, 263, 309, 312)

[Cheng88] Sheng-Chang Cheng, John A. Stankovic (ed), "Scheduling Algorithms for Hard Real-Time Systems – A Brief History", Hard Real-Time Systems, 1988 Computer Society Press of the IEEE, Washington, D.C., pg 151

[Coolahan88] James E. Coolahan, John A. Stankovic (ed), "Timing Requirements for Time-Driven Systems Using Augmented Petri Nets", Hard

Real-Time Systems, 1988 Computer Society Press of the IEEE, Washington, D.C., pg 78

[Coulter99] André Coulter, "Graybox Software Testing Methodology – Embedded Software Testing Technique", 18th Digital Avionics Systems Conference Proceedings, 1999 IEEE, pg (10.A.5-2)

[Elfriede99] Dustin Elfriede, Automated Software Testing, 1999 Addison Wesley Longman, pg 39

[Enomoto81] H. Enomoto, Morio Onoe (ed), "Image Data Modeling and Language for Parallel Processing", Real-Time/Parallel Computing Image Analysis, 1981 Plenum Press, New York pg 101

[Feeley72] John W. Feely, Walter Freiberger (ed), "A Computer Performance Monitor and Markov Analysis for Multiprocessor System Evaluation", Statistical Computer Performance Evaluation, 1972 Academic Press Inc., London, pg 177

[Grenander72] U. Grenander, Walter Freiberger (ed), "Quantitative Methods for Evaluating Computer System Performance: A Review and Proposals", Statistical Computer Performance Evaluation, 1972 Academic Press Inc., London, pg 15

[Hanaki81] S. Hanaki, Morio Onoe (ed), "An Interactive Image Processing and Analysis System", Real-Time/Parallel Computing Image Analysis, 1981 Plenum Press, New York, pg 219

[Isaka81] J. Isakai, Morio Onoe (ed), "A Compound Computer System for Image Data Processing", Real-Time/Parallel Computing Image Analysis, 1981 Plenum Press, New York, pg 257

[Jelinski72] Z. Jelinski, Walter Freiberger (ed), "Software Reliability Research", Statistical Computer Performance Evaluation, 1972 Academic Press Inc, London, pg 467

[Martin67] James Martin, Design of Real-Time Computer Systems, 1967 Prentis-Hall Inc, New Jersey, pgs (257,373)

[Onoe81] Morio Onoe (ed), Real-Time/Parallel Computing Image Analysis, 1981 Plenum Press, New York pg vii

[Spencer85] Richard H. Spencer, Computer Usability Testing and Evaluation, 1985 Prentis-Hall Inc, New Jersey, pgs (13,50,98)

[Volz88] Richard Volz, John A. Stankovic (ed), "Timing Issues in the Distributed Execution of Ada Programs", Hard Real-Time Systems, 1988 Computer Society Press of the IEEE, Washington, D.C., pg 130

[Zhao88] Wei Zhao, John A. Stankovic (ed), "Preemptive Scheduling Under Time and Resource Constraints", Hard Real-Time Systems, 1988 Computer Society Press of the IEEE, Washington, D.C., pg 225

André Coulter

Andre Coulter is the Tools and Technology Leader at Lockheed Martin Missiles and Fire Control - Orlando. Mr. Coulter has over 22 years of software development experience and has spent over 12 years investigating and developing automated testing tools in support of Lockheed Martin embedded systems applications in *Ada, Java, Perl, Fortran, C,* and *assembly language*.

Mr. Coulter is a graduate of Bowie State University, Bowie, Md. in 1978 with a BS in Business Administration. He also taught computer programming at Drake State College in Huntsville Ala. Email andre.c.coulter@lmco.com