Safety Critical Flight Software

Code Coverage Utilization

Nate Uitenbroek

Outline

- Background
- Safety Critical Software
- Classifying Standards
- Contrast Commercial Aviation and Space Flight
- Observations
- Orion Specific Application (DRACO)

My Background

- NASA / L3 Communications
  - Orion Flight Software Architect
  - Orion Software Systems Engineering and Integration
- Honeywell
  - Orion C&DH Flight Software Lead
    - NPR 7150.2 Level A
  - ISS MDM Application Test Environment field support engineer (MATE)
    - Software Development and Integration Lab Software Verification Facility - SDIL-SVF
- Rockwell Collins
  - Boeing 767 Display Head Module Software Development and Test Lead
    - DO-178B Level A Flight Software development and test

Safety Critical Software

- What is safety critical software
  - Safety Critical software performs functions critical to human survival
- Classifying Standards
  - NASA NPR 7150.2
    - NASA Software Engineering Requirements
  - RTCA/DO178B
    - Software Considerations in Airborne Systems and Equipment Certification

NPR 7150.2 Software Classification

- Class A – Human Rated Software Systems
  - Applies to all Space Flight Software Subsystems (Ground and Flight) developed and/or operated for NASA to support human activity in space and that interact with NASA human space flight systems
- Examples of Class A software for human rated space flight systems
  - guidance, navigation and control; life support systems; crew escape; automated rendezvous and docking; failure detection, isolation and recovery and mission ops
- Levels B, C, D, F, G and H also exist to cover
  - non-human, mission support, general purpose and desktop software

DO178B Software Levels

- Level A - Software whose anomalous behavior, as shown by the system safety assessment process, would cause or contribute to a failure of system function resulting in a catastrophic failure condition for the aircraft
  - Catastrophic Failure - Failure conditions which would prevent continued safe flight and landing
- Level B - Software whose anomalous behavior as shown by the system safety analysis process, would cause or contribute to a failure of system function resulting in a hazardous/severe-major failure condition for the aircraft
  - Hazardous/Severe-Major Failure - Failure condition that would reduce the capability of the aircraft or ability of the crew to cope with adverse conditions to the extent that would be:
    1. A large reduction in safety margins or functional capabilities
    2. Physical distress or higher workload such that the flight crew could not be relied on to perform their duties accurately or completely
    3. Adverse effect on occupants including serious or potentially fatal injuries to a small number of those occupants

Comparison

767 FSW	Orion FSW	Comparison
Test procedures are correct	Test procedures are correct	Similar process and checklists are used
Test results are correct and discrepancies explained	Test results are correct and discrepancies explained	Similar process and checklists are used
Test coverage of high level requirements is achieved	Test coverage of high level requirements is achieved	Similar process and checklists are used
Test coverage of low level requirements is achieved	Test coverage of verification success criteria is achieved	Orion derives verification success criteria from design constrains that are linked to requirements, while commercial aviation approaches leverage design level shall statements. The results are very similar.
Test coverage of software structure is achieved Level A - Modified Condition/Decision Level B – Decision Coverage	Test coverage of software structure is achieved Class A - Modified Condition/Decision	Collection of code coverage in commercial aviation is required during the requirements based testing campaign. Space flight requirements are less prescriptive and allow tailoring. Orion has chosen to collect code coverage during unit test rather than verification
Test coverage of software structure (data and control coupling) is achieved	Test coverage of software structure (data and control coupling) is achieved	Orion is still developing its approach to testing data and control coupling and it is planned to be similar to commercial aviation

Objectives should be satisfied with Independence

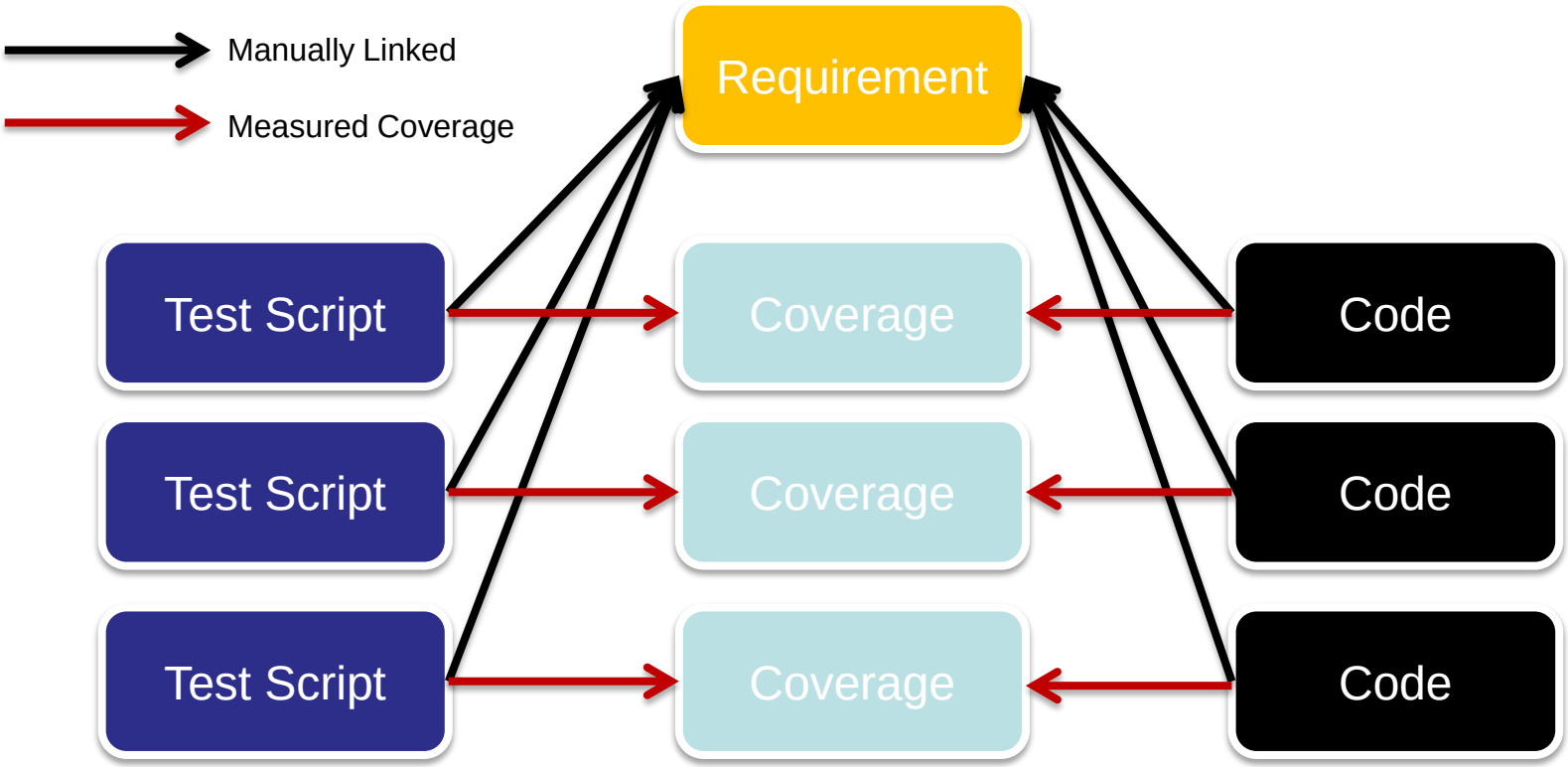
Observations

- Boeing 767 Display Unit Flight Software
  - Code coverage metrics utilized to measure verification test coverage
  - Requirements based test campaign
  - Unit under test is the flight load
- Orion Flight Software
  - Code coverage metrics utilized to measure unit test coverage
  - Code structure based tests
  - Unit under test is the class with stubs and drivers

Structural Coverage Analysis Resolution

- Shortcomings in requirements-based test cases
  - Supplement test cases or change test procedures
- Inadequacies in software requirements
  - Software requirements should be modified and additional test cases developed
- Dead / Deactivated Code
  - The code could be removed and analysis performed to assess the need for re-verification
  - Analysis and Testing could be done to show that there are no means by which the code can be executed in the normal target computer environment
  - Show that the execution of the code would not lead to catastrophic anomalies

Coverage Metrics Measure Test Campaign Rigor



Code coverage measurements confirm that the manually linked code was adequately exercised during the requirements based testing efforts

DRACO

- Database and Reporting Application for Code Coverage on Orion (DRACO)
  - NASA developed tool that leverages a flight computer emulation to execute tests and measure code coverage
- Concept of Operations
  - Monitor the executable flight software in the target computer memory via probes / tooling
  - Execute a suite of tests to exercise the flight software
  - Collect memory locations of executed lines of code
  - Correlate memory locations back to the source code to determine source code coverage of a particular run
  - Create reports that allow selection and aggregation of coverage metrics from multiple test runs
  - Produce annotated source code listings that allow testers to improve the coverage of their tests
  - Produce aggregate reports showing test campaign effectiveness

Annotated Source Code

```
39
40  CDHProcess1Hz::CDHProcess1Hz(int argfoo) : Thread(RateGroup::OneHz, false), numberOfFailures(0) {
41      //#[ operation CDHProcess1Hz(int)
42      foo = argfoo;
43      //#]
44  }
45
46  CDHProcess1Hz::~CDHProcess1Hz() {
47      //#[ operation ~CDHProcess1Hz()
48      //#]
49  }
50
51  bool CDHProcess1Hz::create() {
52      //#[ operation create()
53      if (Thread::Create())
54      {
55          return true;
56      }
57      else
58      {
59          // Thread creation has failed
60          // Indicate failure with class attributes
61          numberOfFailures++;
62          state = 0xDEAD;
63          return false;
64      }
65      //#]
66  }
67
68  void CDHProcess1Hz::Execute() {
69      //#[ operation Execute()
70      //#]
71  }
```

Code Coverage Metrics Report

Orion Code Coverage Report Summary

Report Generation Date:	5 August, 2016 02:42PM
FSW Version:	26a
XML Report File:	/var/www/html/ktomlin1/codecover/source/draco/reportFiles/report_320.default.xml
JSON Coverage Files:	/var/www/html/ktomlin1/codecover/source/draco/26a.mode3.test.py.2016-08-05_14-40-41.json

Partition Overview

Name	Testruns	Instructions Hit	Total Instructions	% Coverage
cdh	1	6125	16313	37.55%

cdh Partition Summary

Source File	Testruns Reported	Instructions Hit	Total Instructions	% Coverage
scp/partitions/cgCdhPartition/CdhPsm/src/cdhPsm/PartitionInhibitEnableHandler.cpp	1	5	28	17.86 %
scp/partitions/cgCdhPartition/CdhPsm/src/cdhPsm/CDHProcess5Hz.cpp	1	7	13	53.85 %
scp/partitions/cgCdhPartition/CdhPsm/src/cdhPsm/CDHProcess40Hz.cpp	1	96	171	56.14 %
scp/partitions/cgCdhPartition/CdhPsm/src/cdhPsm/CDHProcess1Hz.cpp	1	7	13	53.85 %
scp/partitions/cgCdhPartition/CdhPsm/src/cdhPsm/CDHFactory.cpp	1	98	103	95.15 %
scp/partitions/cgCdhPartition/CdhPim/src/cdhPim/TimeUtilities.cpp	1	16	27	59.26 %
scp/partitions/cgCdhPartition/CdhPim/src/cdhPim/TimeReferenceSource.cpp	1	54	97	55.67 %

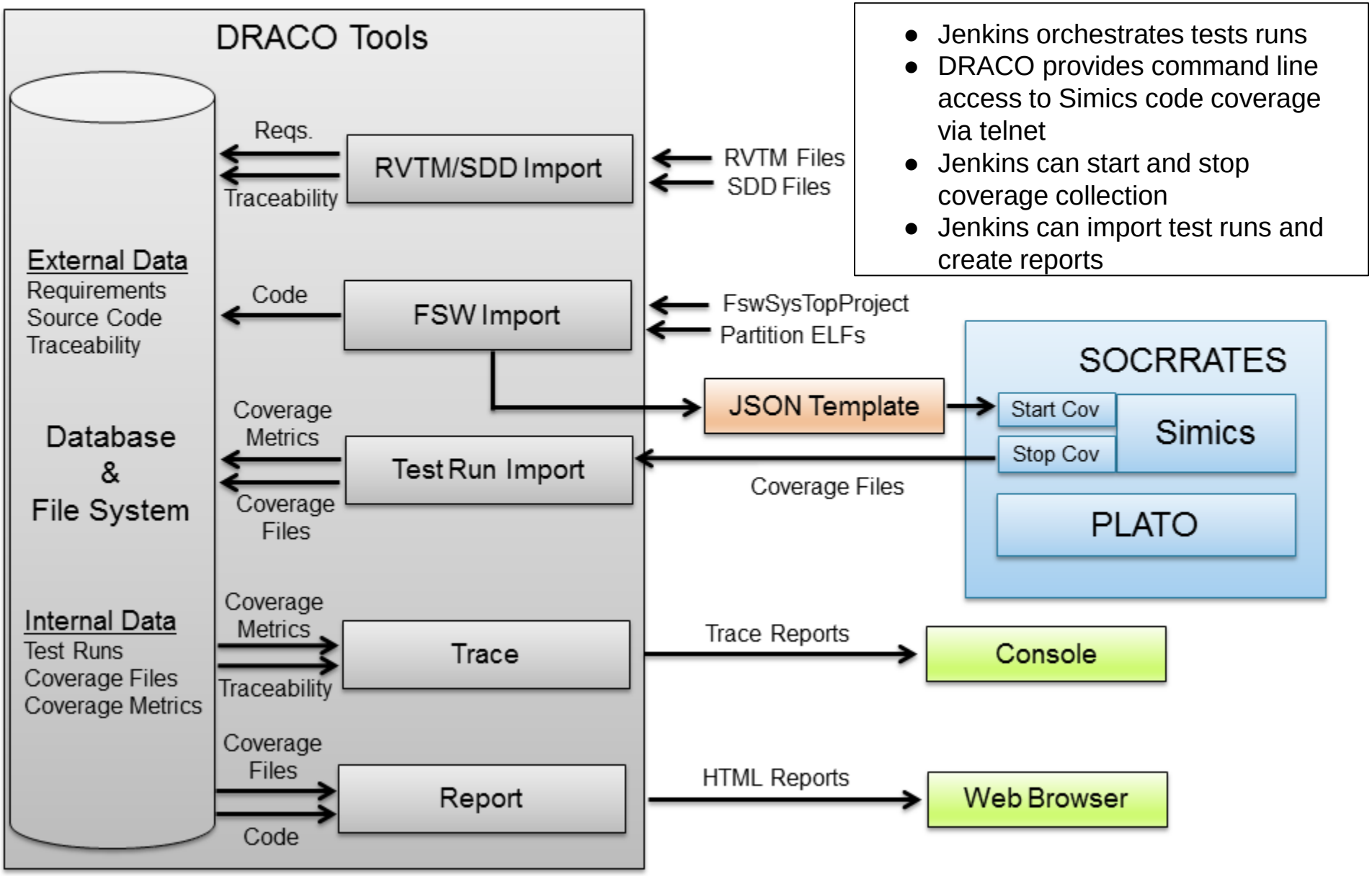
Value to Orion

- Currently there are limited objective measures of comprehensiveness of the verification test campaign
- Incremental verification strategy increases the need to understand individual test coverage to evaluate the comprehensiveness of the regression test suite
- Increases the confidence in Orion flight software ensuring successful Orion EM-1 and EM-2 missions
- Provides objective approach to measuring code coverage on any project that uses emulation models

Complexity and Innovation

- Track execution of software via address monitoring
- Breakpoints initiate a handler that records addresses that were executed
- Post processing translates addresses to source lines
- Database warehouses coverage metrics data
- Reports graphically display results
- Features:
  - Automated test execution and reporting
  - Merge multiple test runs into single report
  - Trace reporting to determine expected coverage
  - Web based interaction for test scheduling, report generation, and analysis

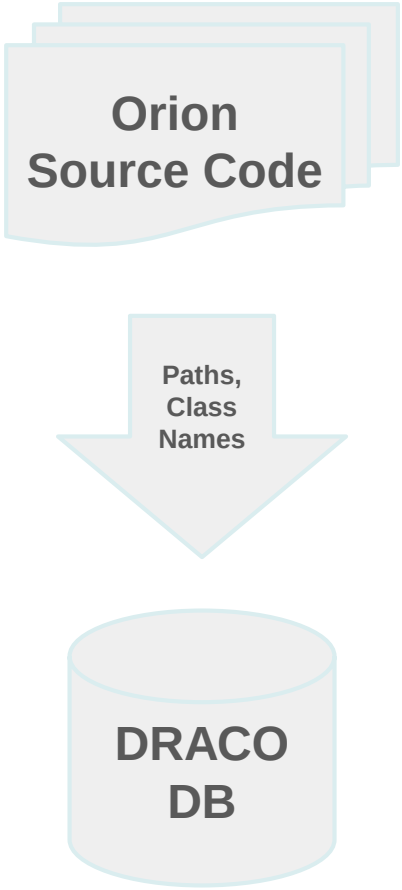
DRACO Architecture



- Jenkins orchestrates tests runs
- DRACO provides command line access to Simics code coverage via telnet
- Jenkins can start and stop coverage collection
- Jenkins can import test runs and create reports

Flight Software Import

- Parses Orion FSW and finds associations between files and class names
- Finds partition association
- Stores associations between path, class name, partition, and flight software version



Template Generation

- Address to source line mapping is obtained from DWARF / ELF
- DWARF / ELF is generated during compilation and contains debug information
- The template is used by DRACO for setting breakpoints and for generating reports


```
            "addr": "0x210b0e4"
        },
        {
            "ln": "116",
            "hc": "",
            "addr": "0x210b0f0"
        },
        {
            "ln": "119",
            "hc": "",
            "addr": "0x210b100"
        }
    ],
    "common/stFaultMgmt/src/MatlabFmToolkit/FMToolkit_TwoInputFoidFailAnd.cpp": [
        {
            "ln": "56",
            "hc": "",
            "addr": "0x213d5dc"
        },
        {
            "ln": "45",
            "hc": "",
            "addr": "0x213d5b8"
        },
        {
            "ln": "46",
            "hc": "",
            "addr": "0x213d5c8"
        },
        {
            "ln": "47",
            "hc": "",
            "addr": "0x213d5cc"
        },
        {
            "ln": "48",
            "hc": "",
            "addr": "0x213d5d0"
        },
        {
            "ln": "49",
            "hc": "",
            "addr": "0x213d5d4"
        },
        {
            "ln": "50",
            "hc": "",
            "addr": "0x213d5d8"
        },
```

Simics Start

- Simics uses a configuration file to define code coverage objects for each partition based on an address range
- Start command sets a breakpoint on each address of interest
- Breakpoint handler records each address hit in address dictionary for stop command to write out

start command

partition object

test script name

```
simics> cov-start code_coverage = cdh script = DRACO_EM1_FSW_ALL_ANOOP_CMDS_0000.py
MODE: 3
Opening coverage file...
Creating dictionary of addresses...
Done.
```

```
@with open("DRACO/codecover/source/draco/resources/partition_ranges.json") as partition_ranges:
        ranges=json.load(partition_ranges)

@try:
        for partition in ranges:
                SIM_create_object("code_coverage", partition,
                [['cpu_obj', conf.vmc750.cpu], ['mem_space', conf.vmc750.phys_mem],
                ['lo_addr', int(ranges[partition]['lo_addr'],16)], ['size', int(ranges[partition]['size'],16)],
                ['cov_template', 'DRACO/codecover/source/draco/sample/'+partition+'.26a.covDataTemplate.json'],
                ['cov_output', 'code_coverage_output/'+partition+'_26a_output']])
```

Simics Start: Modes

- **Mode 1: Heat Map on Partition**
  - Aggregates hit counts for each address to create a “heat map” of coverage
  - Slowest speed but generates the most detailed coverage data
- **Mode 2: Heat Map on List of C++ Source Files**
  - Sets breakpoints on every address of C++ source files defined in XML input
  - Same detailed coverage as mode 1 but only for specified files which allows targeting specific files and a faster execution speed
- **Mode 3: Coverage on Partition (default coverage option)**
  - Sets temporary breakpoints on entire partition
  - Only documents whether or not address/source line was hit
  - Fastest speed, manageable performance impact when targeting individual partitions

Simics Stop

- Reads hit counts from address dictionary and writes to JSON coverage file for the testrun
- Cleans up breakpoints

stop command

partition object

```
running> cov-stop code_coverage = cdh
Deleting breakpoints...
Done.
running> stop
simics>
```


```
{
    "duration": 23.583982364999997,
    "files": {
        "common/cgDiu/scp/src/diu/UTL_Platform_S
            {
                "addr": "0x2140734",
                "hc": "1",
                "ln": "43"
            },
            {
                "addr": "0x2140754",
                "hc": "1",
                "ln": "44"
            },
            {
                "addr": "0x2140774",
                "hc": "1",
                "ln": "45"
            },
            {
                "addr": "0x2140794",
                "hc": "1",
                "ln": "46"
            },
            {
                "addr": "0x21407b4",
                "hc": "1",
                "ln": "50"
            },
            {
                "addr": "0x21407c0",
                "hc": "1",
```

Import Coverage

- Get coverage file (filled in JSON template) from Simics
- Parse file, gather coverage metrics per C++ source file
- Import metrics, store file
- Generate default report file

Generate Reports

- Report file (XML) specifies test runs to report
  - Option to merge test runs
  - Option to report of specific files
- Combine coverage data by partition
  - Optionally, only pay attention to specified files
- Create report summary
- Create annotated source file reports with hit/miss highlighting

Trace Reports

- Combine internal and external data
  - Traceability data from RVTM/SDD import
  - Coverage data from test run import
- Source trace:
  - Given a source file, what test script should cover it?
  - How well do each of those test scripts cover this file?
- Script trace:
  - Given a test script, what source files should it cover?
  - How well does the script cover those files?

Running Simics from DRACO

Simics

Simics Control

File Edit Run Debug Tools Windows Help

Honeywell P/N PS64001663-124 Release-v4_8a10_SCP

System: VMC750 - SCP System

Processor: 1 ppc750fx, 400 MHz

Memory: 0 B

Ethernet: 1 of 3 connected

Storage: No disks

2 min 55.801 s

Simics Command Line

```
Wind River Simics 4.8 (build 4605 linux64) Copyright 2010-2015 Intel Corporation
Simics search path is now empty.
The current Simics search path is:
   /app_home/app/socrrates/development/Code_Coverage/Workspace_Latest_fcm1d/
targets
   /app_home/app/socrrates/development/Code_Coverage/Workspace_Latest_fcm1d/
targets/vmc750
   /app_home/app/socrrates/development/Code_Coverage/Workspace_Latest_fcm1d/
targets/vmc750/images
Setting step rate to 1/1 steps/cycle
Setting module type to FCM #1 in System Status Buffer
[tcon0 info] {vmc750.cpu 0x3b0fc 70320407054} Telnet frontend accepting
connections on port 8199

SECURITY WARNING: Port 8199 will be open for anyone to access.
```

DRACO

```
[ncluck@tooldev draco]$ ./draco.py simics
Usage:
 draco.py simics "[any command to pass to Simics CLI]"
OR
 draco.py simics cov-start [partition name] -r
 draco.py simics cov-start [partition name] -hc fname = [file name]
 draco.py simics cov-start [partition name]
 draco.py simics run
 draco.py simics cov-stop [partition name]

The default IP is 139.169.30.241 and the default port is 8192.
However, if you need to send commands to a different IP address or port, add the
 following arguments to each command you run:

 draco.py simics -i [ip address] -p [port] [simics commands listed above]

[ncluck@tooldev draco]$
```

DRACO and PLATO

```
# Establish telnet connection to simics
try:
    tn = telnetlib.Telnet(telnet_ip, telnet_port)
except Exception as e:
    print "\n\n\n############################## Cannot connect to %s ##############################\n\n\n" % hostPort
    print "You may not have Simics running." + str(e)
    sys.exit(1)

# Call start coverage
tn.write('start-cov code_coverage = cdh -v script=EM1-FSW-ALL-ANOOP-CMDS-0000.py\n')

# Initialize the partition for testing
for partition in singleControlModulePartitions:

    partition.initialize()
    # Only print the header once
    sapi.scp_printHeader = dontPrintHeader

    # BMP and FDO can't be inhibited or enabled, so only send a NoOp
    if isinstance(partition, EnabledPartition):
        partition.noOp()
    else:
        partition.inhibit()
        partition.noOp()

        partition.enable()
        partition.noOp()

# Call stop coverage
tn.write('stop-cov code_coverage = cdh\n')

# Clean-up
cleanUp(test_case, singleControlModulePartitions)
sapi.scp_requestShutdown()

return
```

Where is DRACO being use?

- Currently, where is the software being used?
  - JSC – Kedalion lab to measure Orion regression test suite coverage to assist Software Functional Manager COFR assessment of the flight software
  - Industry – Web based access is currently under development for Lockheed Martin to remotely run tests, create reports and review analysis
- Where and how else could the software be used?
  - Any project using Simics emulations could use this capability
  - Demonstrated to Windriver for inclusion in their product offering

Future Plans for DRACO

- Orion regression test assessment to begin Fall 2017
- Team of 3 to 5 interns to support test execution and metrics collection
- Reports and analysis to be provided to Lockheed Martin
- Tuning of the regression test suite to be an ongoing activity through EM-1 verification campaign (2019)
- Program support planned for 4 interns year round to run tests and maintain DRACO tooling

Backup data

4. Team Members & Awards

- Team Members
  - NTR
    - Nathan Uitenbroek
    - Cassidy Matousek
    - Alex Blankenberger
    - Luke Doman
    - Kiran Tomlinson
    - Natalie Cluck
  - Recent Contributors
    - Erik Vanderwerf
    - Robin Onsay
    - Sumaya Asif

5. Development & Release History

- Development Start – June 2016
- Initial Release – August 2016
- Incremental Improvements
  - Test Automation and Integration with Jenkins – December 2016
  - Web interface and reporting enhancements – May 2017
- Next Release - May 2017

7. Form NF 1679 status

- e-NTR #: 1472574999 Status: NASA Accepted

My Accepted Entries (1)

1. Database and Reporting tool for Code coverage on Orion (DRACO)

Last Modified On: August 30 2016 14:34:29 PDT **Center:** JSC **e-NTR #:** 1472574999 NTR Status

Report Date: 2016-08-30 **Case #:** MSC-26217-1 **NASA Accepted**

PDF View Previous Versions Update View Comments (1) Duplicate Submit for Review Delete

Database and Reporting tool for Code coverage on Orion (DRACO)

top

8. NPR 7150.2B Compliance

- DRACO has been developed using Agile development processes commensurate with its classification as NPR-7150.2B Class E software
- In many cases the team has chosen to follow processes that align more closely with Class C software to increase the quality
  - This includes the use of automated requirements based tests with traceability
  - Peer reviews of all development and test artifacts have been performed and captured
    - requirements, architecture, implementation, test scripts, test results

NPR 7150.2 Software Classification

Table 1.2.1-1 Software Classification Levels and Definitions	
Level	**Definition**
Class A: Human Rated Software Systems	• Applies to all space flight software subsystems (Ground and Flight) developed and/or operated by or for NASA to support human activity in space and that interact with NASA human space flight systems. • Space flight system design and associated risks to humans are evaluated over the program's life cycle, including design, development, fabrication, processing, maintenance, launch, recovery and final disposal. Examples of Class A software for human rated space flight include but are not limited to: guidance; navigation and control; life support systems; crew escape; automated rendezvous and docking; failure detection, isolation and recovery; and mission ops.
Class B: Non-Human Space Rated Software Systems	• Flight and Ground software that must perform reliably in order to accomplish primary mission objectives. Examples of Class B software for nonhuman (robotic) spaceflight include, but are not limited to, propulsion systems; power systems; guidance navigation and control; fault protection; thermal systems; command and control ground systems; planetary surface operations; hazard prevention; primary instruments; or other subsystems that could cause the loss of science return from multiple instruments.
Class C: Mission Support Software	• Flight or Ground software that is necessary for the science return from a single (noncritical) instrument or is used to analyze or process mission data or other software for which a defect could adversely impact attainment of some secondary mission objectives or cause operational problems for which potential workarounds exist. • Class C software must be developed carefully, but validation and verification effort is generally less intensive than for Class B. Examples of Class C software include, but are not limited to, software that supports prelaunch integration and test, mission data processing and analysis, analysis software used in trend analysis and calibration of flight engineering parameters, primary/major science data collection and distribution systems, major Center facilities, data acquisition and control systems, aeronautic applications, or software employed by network operations and control (which is redundant with systems used at tracking complexes).

NPR 7150.2 Software Classification

Class D: Analysis and Distribution Software	• Nonspace flight software. Software developed to perform science data collection, storage and distribution; or perform engineering and hardware data analysis. • A defect in Class D software may cause rework but has no direct impact on mission objectives or system safety. Examples of Class D software include, but are not limited to, software tools; analysis tools, and science data collection and distribution systems.
Class E: Development Support Software	• Nonspace flight software. Software developed to explore a design concept; or support software or hardware development functions such as requirements management, design, test and integration, configuration management, documentation, or perform science analysis. • A defect in Class E software may cause rework but has no direct impact on mission objectives or system safety. Examples of Class E software include, but are not limited to, earth science modeling, information only websites (nonbusiness/info technology); science data analysis; and low technical readiness level research S/W.

Levels F, G and H also exist to cover general purpose and desktop software

DO178B Software Levels

Software Level Definitions

Software level is based upon the contribution of software to potential failure conditions as determined by the system safety assessment process. The software level implies that the level of effort required to show compliance with certification requirements varies with the failure condition category. The software level definitions are:

a. Level A: Software whose anomalous behavior, as shown by the system safety assessment process, would cause or contribute to a failure of system function resulting in a catastrophic failure condition for the aircraft.

b. Level B: Software whose anomalous behavior, as shown by the system safety assessment process, would cause or contribute to a failure of system function resulting in a hazardous/severe-major failure condition for the aircraft.

c. Level C: Software whose anomalous behavior, as shown by the system safety assessment process, would cause or contribute to a failure of system function resulting in a major failure condition for the aircraft.

d. Level D: Software whose anomalous behavior, as shown by the system safety assessment process, would cause or contribute to a failure of system function resulting in a minor failure condition for the aircraft.

e. Level E: Software whose anomalous behavior, as shown by the system safety assessment process, would cause or contribute to a failure of system function with no effect on aircraft operational capability or pilot workload. Once software has been confirmed as level E by the certification authority, no further guidelines of this document apply.

DO178B Failure Categories

a. Catastrophic: Failure conditions which would prevent continued safe flight and landing.

b. Hazardous/Severe-Major: Failure conditions which would reduce the capability of the aircraft or the ability of the crew to cope with adverse operating conditions to the extent that there would be:

(1) a large reduction in safety margins or functional capabilities,

(2) physical distress or higher workload such that the flight crew could not be relied on to perform their tasks accurately or completely, or

(3) adverse effects on occupants including serious or potentially fatal injuries to a small number of those occupants.

c. Major: Failure conditions which would reduce the capability of the aircraft or the ability of the crew to cope with adverse operating conditions to the extent that there would be, for example, a significant reduction in safety margins or functional capabilities, a significant increase in crew workload or in conditions impairing crew efficiency, or discomfort to occupants, possibly including injuries.

d. Minor: Failure conditions which would not significantly reduce aircraft safety, and which would involve crew actions that are well within their capabilities. Minor failure conditions may include, for example, a slight reduction in safety margins or functional capabilities, a slight increase in crew workload, such as, routine flight plan changes, or some inconvenience to occupants.

e. No Effect: Failure conditions which do not affect the operational capability of the aircraft or increase crew workload.

Software Verification Process

th just first couple columns to DO178B

	Objective		Applicability by SW Level				Output		Control Category by SW level			
	Description	Ref.	A	B	C	D	Description	Ref.	A	B	C	D
1	Test procedures are correct.	6.3.6b	●	○	○		Software Verification Cases and Procedures	11.13	②	②	②	
2	Test results are correct and discrepancies explained.	6.3.6c	●	○	○		Software Verification Results	11.14	②	②	②	
3	Test coverage of high-level requirements is achieved.	6.4.4.1	●	○	○	○	Software Verification Results	11.14	②	②	②	②
4	Test coverage of low-level requirements is achieved.	6.4.4.1	●	○	○		Software Verification Results	11.14	②	②	②	
5	Test coverage of software structure (modified condition/decision) is achieved.	6.4.4.2	●				Software Verification Results	11.14	②			
6	Test coverage of software structure (decision coverage) is achieved.	6.4.4.2a 6.4.4.2b	●	●			Software Verification Results	11.14	②	②		
7	Test coverage of software structure (statement coverage) is achieved.	6.4.4.2a 6.4.4.2b	●	●	○		Software Verification Results	11.14	②	②	②	
8	Test coverage of software structure (data coupling and control coupling) is achieved.	6.4.4.2c	●	●	○		Software Verification Results	11.14	②	②	②	

Structural Coverage

Structural Coverage Analysis Resolution

Structural coverage analysis may reveal code structure that was not exercised during testing. Resolution would require additional software verification process activity. This unexecuted code structure may be the result of:

a. Shortcomings in requirements-based test cases or procedures: The test cases should be supplemented or test procedures changed to provide the missing coverage. The method(s) used to perform the requirements-based coverage analysis may need to be reviewed.

b. Inadequacies in software requirements: The software requirements should be modified and additional test cases developed and test procedures executed.

c. Dead code: The code should be removed and an analysis performed to assess the effect and the need for reverification.

d. Deactivated code: For deactivated code which is not intended to be executed in any configuration used within an aircraft or engine, a combination of analysis and testing should show that the means by which such code could be inadvertently executed are prevented, isolated, or eliminated. For deactivated code which is only executed in certain configurations of the target computer environment, the operational configuration needed for normal execution of this code should be established and additional test cases and test procedures developed to satisfy the required coverage objectives.

Structural Coverage

```
if (Condition1 && Condition2) { OutcomeA; }
else { OutcomeB; }
```

Decision Coverage - Every point of entry and exit in the program has been invoked at least once and every decision in the program has taken on all possible outcomes at least once.

Condition1	Condition2	Outcome
True	True	OutcomeA
False	True	OutcomeB

Condition/Decision Coverage - Every point of entry and exit in the program has been invoked at least once, every condition in a decision in the program has taken on all possible outcomes at least once, and every decision in the program has taken on all possible outcomes at least once.

Condition1	Condition2	Outcome
True	True	OutcomeA
False	False	OutcomeB

Modified Condition/Decision Coverage - Every point of entry and exit in the program has been invoked at least once, every condition in a decision in the program has taken all possible outcomes at least once, every decision in the program has taken all possible outcomes at least once, and each condition in a decision has been shown to independently affect that decision's outcome. A condition is shown to independently affect a decision's outcome by varying just that condition while holding fixed all other possible conditions.

Condition1	Condition2	Outcome
True	True	OutcomeA
False	False	OutcomeB
True	False	OutcomeB
False	True	OutcomeB

Structural Coverage

if (Condition1 && Condition2) { OutcomeA; } else { OutcomeB; }

Decision Coverage

Condition1	Condition2	Outcome
True	True	OutcomeA
False	True	OutcomeB

Condition/Decision Coverage

Condition1	Condition2	Outcome
True	True	OutcomeA
False	False	OutcomeB

Modified Condition/Decision Coverage

Condition1	Condition2	Outcome
True	True	OutcomeA
True	False	OutcomeB
True	False	OutcomeB
False	False	OutcomeB