RAPITA SYSTEMS LTD

ALTRAN

UNIVERSITY OF OXFORD

Automated Testing of SPARK Contracts

AUTOSAC NATEP Research Project

Ada Europe 2016

Ian Broster (Rapita)
Martin Brain (University of Oxford)
Stuart Matthews, Andrew Hawthorn, Florian Schanda (Altran)

Rolls-Royce

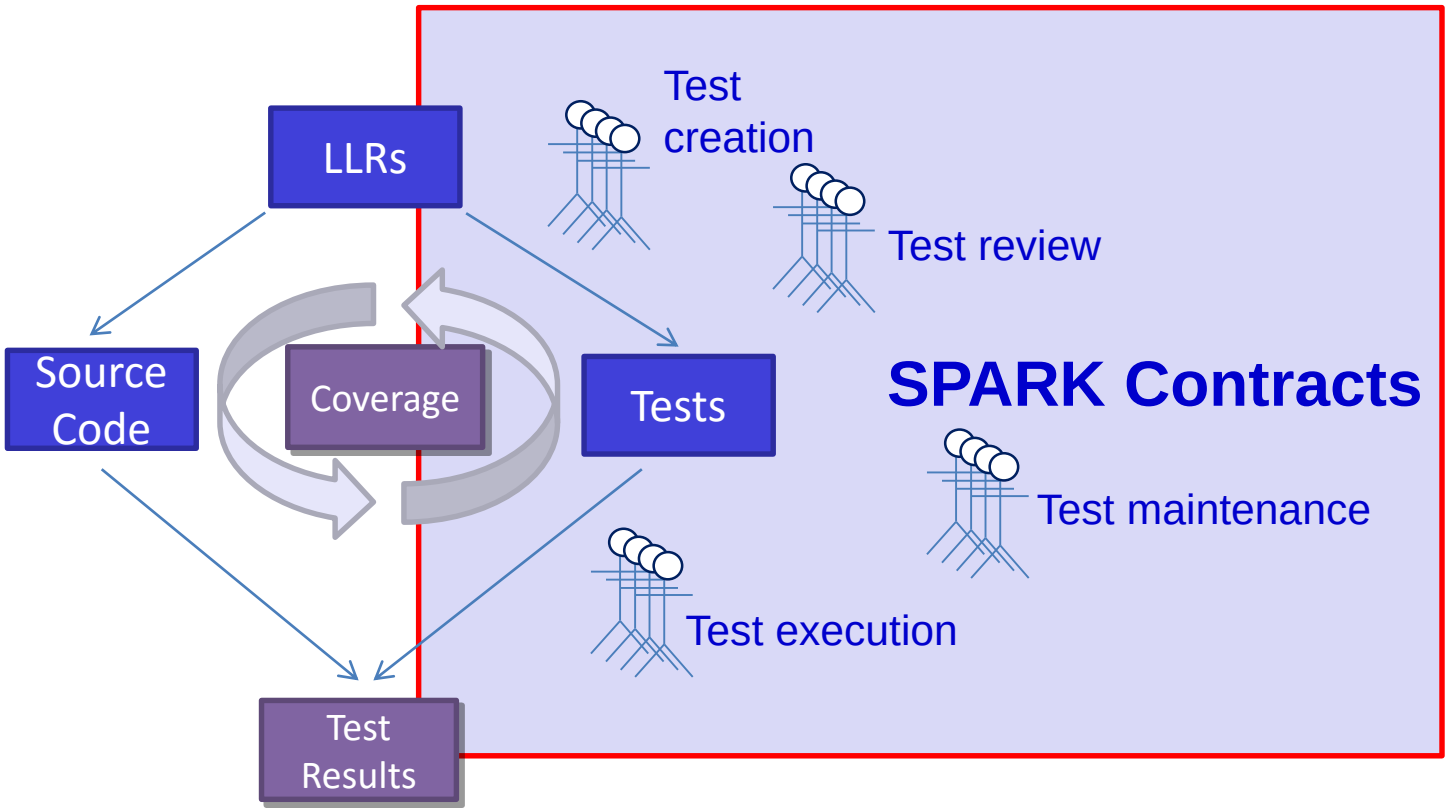
MBDA MISSILE SYSTEMS

Aerospace Growth Partnership

Department for Business Innovation & Skills

Automated Testing of SPARK Contracts

Reduce time and effort to test low-level requirements (LLRs) of safety-critical SPARK code



Automation using SPARK Contracts

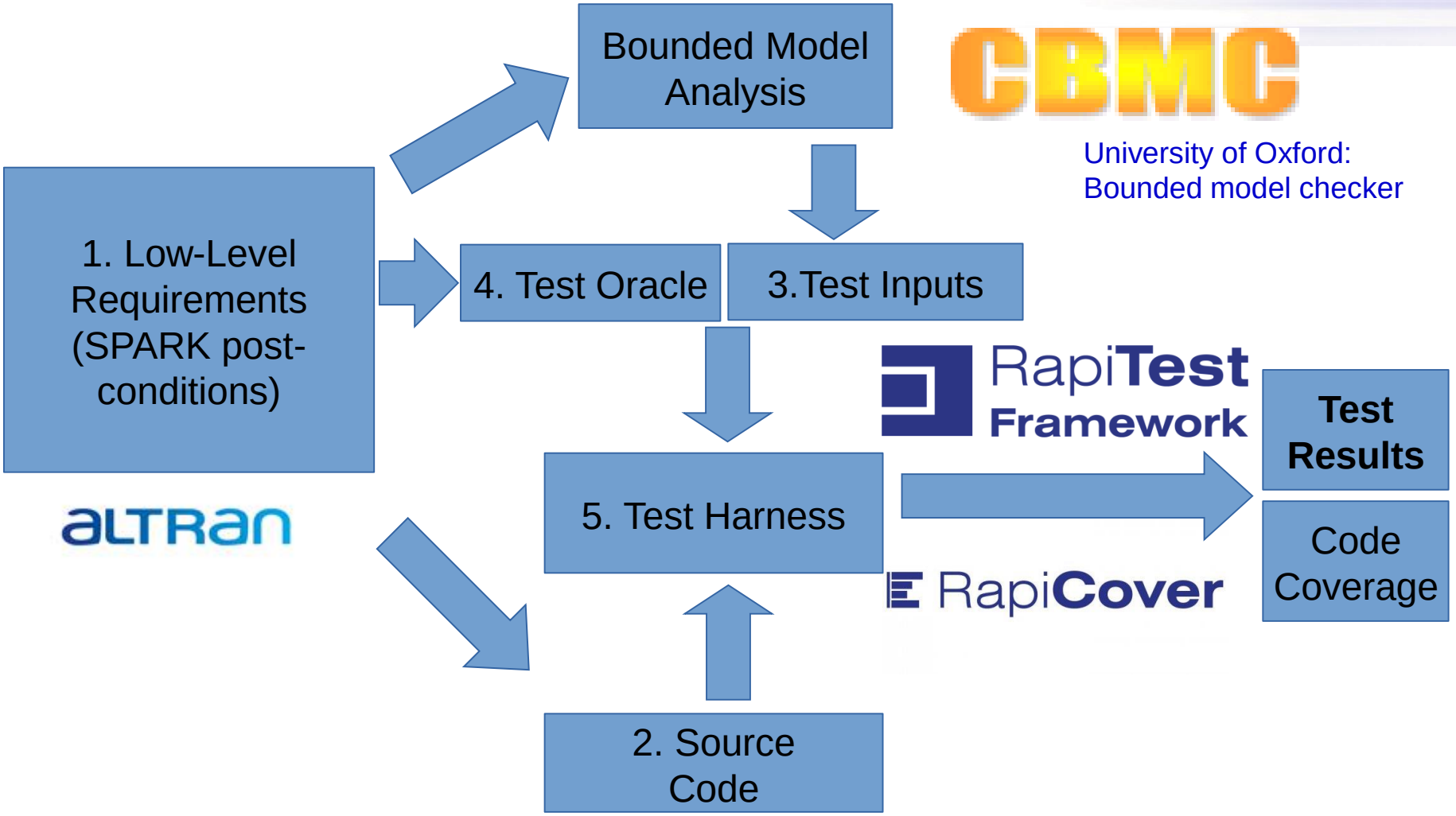
SPARK contracts can describe a subprogram specification well

We can use them for:

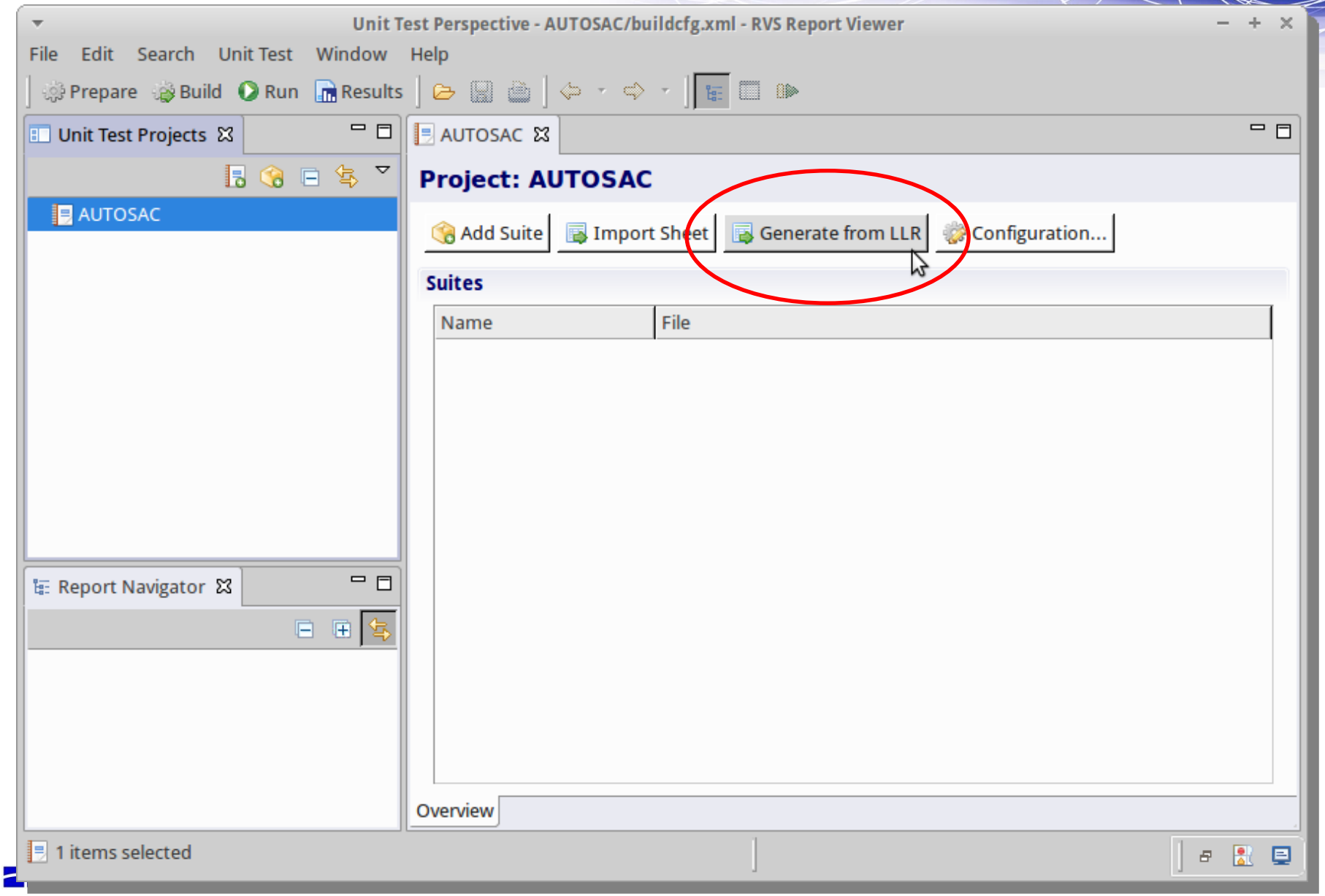
- Low-level requirements (LLR)
- Auto-generation of unit tests based on LLR
- Verification of Test Results

```
function Days_In_Month ( M : Month_T; Year : Year_T)
return Days_In_Month_T
with Post =>
  Days_In_Month'Result =
     (case M is
      when September | April | June | November => 30,
      when February =>
            (if Year mod 100 = 0 then
                (if Year mod 400 =0 then 29 else 28)
             else
                (if Year mod 4 = 0 then 29 else 28)),
      when others => 31);
```

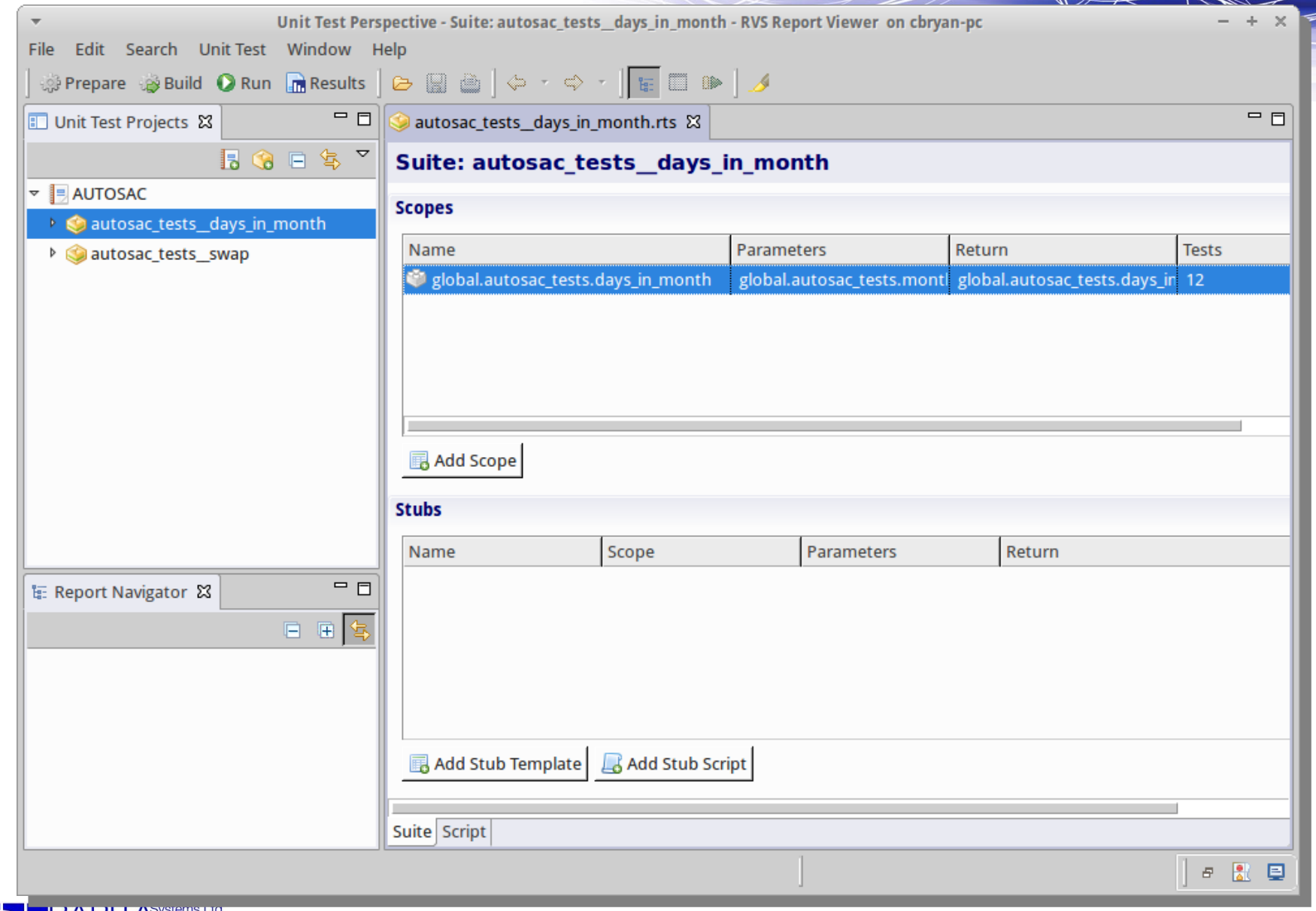
AUTOSAC Tool chain



Integration with RapiTest Framework



Tests have been generated by CMBC



Exploring “coverage” of the post-condition

CBMC

- Bounded model checker from University of Oxford.

“Explores post-condition to provide test inputs that cover all the post condition”

1. Coverage of the post-condition

- **i.e. generates test inputs that should exercise each part of post condition**

```
Case M is when September | April | June | November => 30,
```

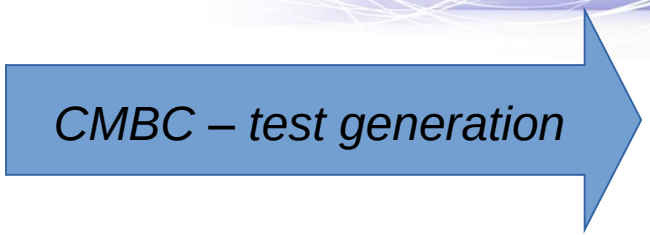
- **Generates a test input for either: Sept, April, June or November.**

2. Boundary coverage for inputs

(e.g. Integer’First, Integer’Last, and intermediate values etc)

3. Test cases that explore discontinuities in non-deterministic post conditions (future work)

Test generation



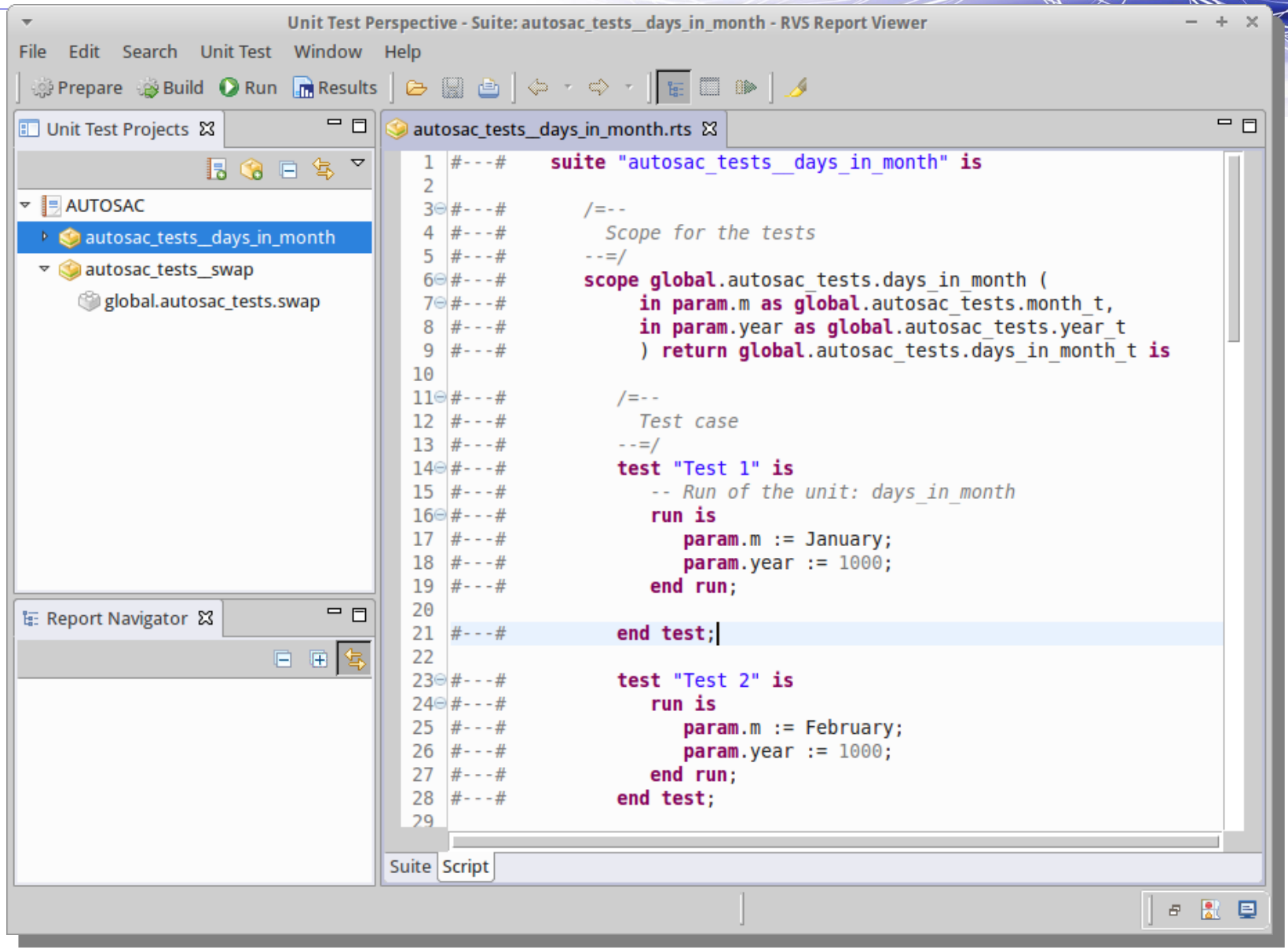
Ada Specification

```
function Days_In_Month (M : Month_T;
                        Year : Year_T)
return Days_In_Month_T
 with Post =>
 Days_In_Month'Result =
 (case M is
  when September | April | June | November =>
    30,
  when February =>
    (if Year mod 100 = 0 then
    (if Year mod 400 =0 then 29 else 28)
else
    (if Year mod 4 = 0 then 29 else 28)),
     when others => 31);
```

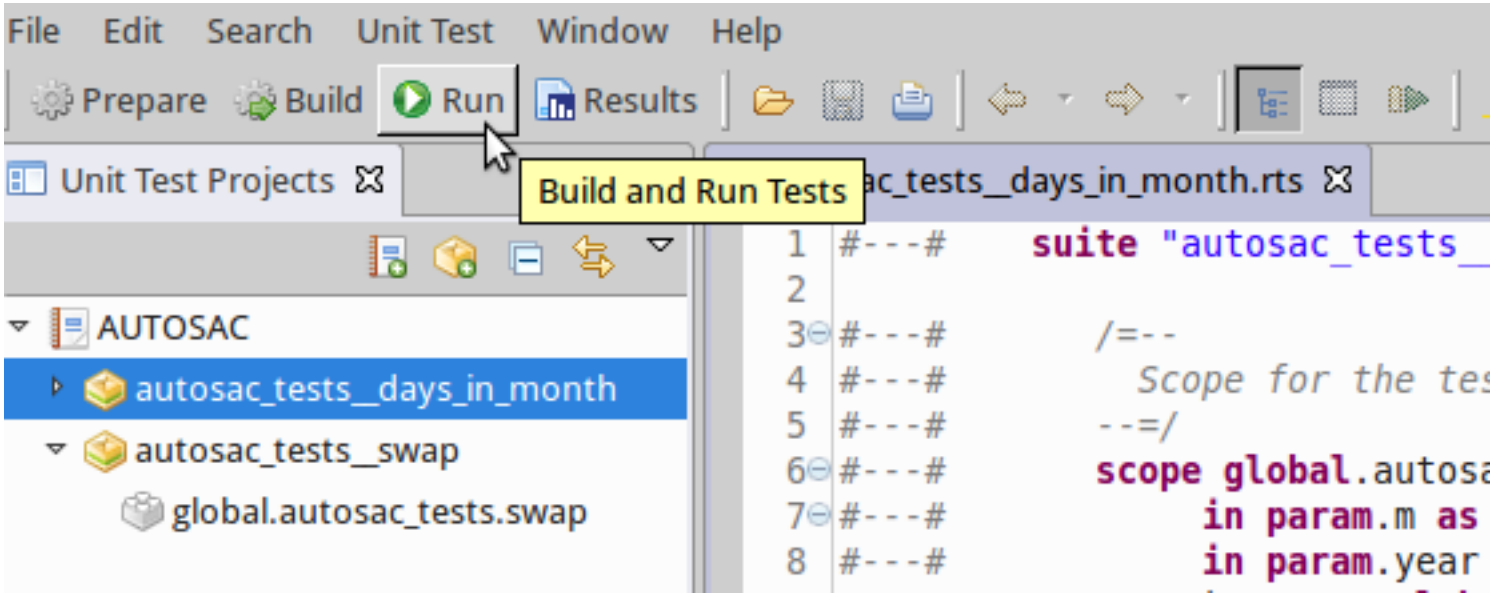
RapiTest Framework Script

```
suite "autosac_tests__days_in_month" is

scope global.autosac_tests.days_in_month (
     in param.m as global.autosac_tests.month_t,
     in param.year as global.autosac_tests.year_t)
 return global.autosac_tests.days_in_month_t is

     /=--
       Test case
     --=/
     test "Test 1" is
        -- Run of the unit: days_in_month
        run is
            param.m := January;
            param.year := 1000;
        end run;
      end test;
      test "Test 2" is
         run is
          param.m := February;
          param.year := 1000;
        end run;
     end test;
```

Test script in RapiTest Framework



Execute the unit-tests



Unit tests auto-generated

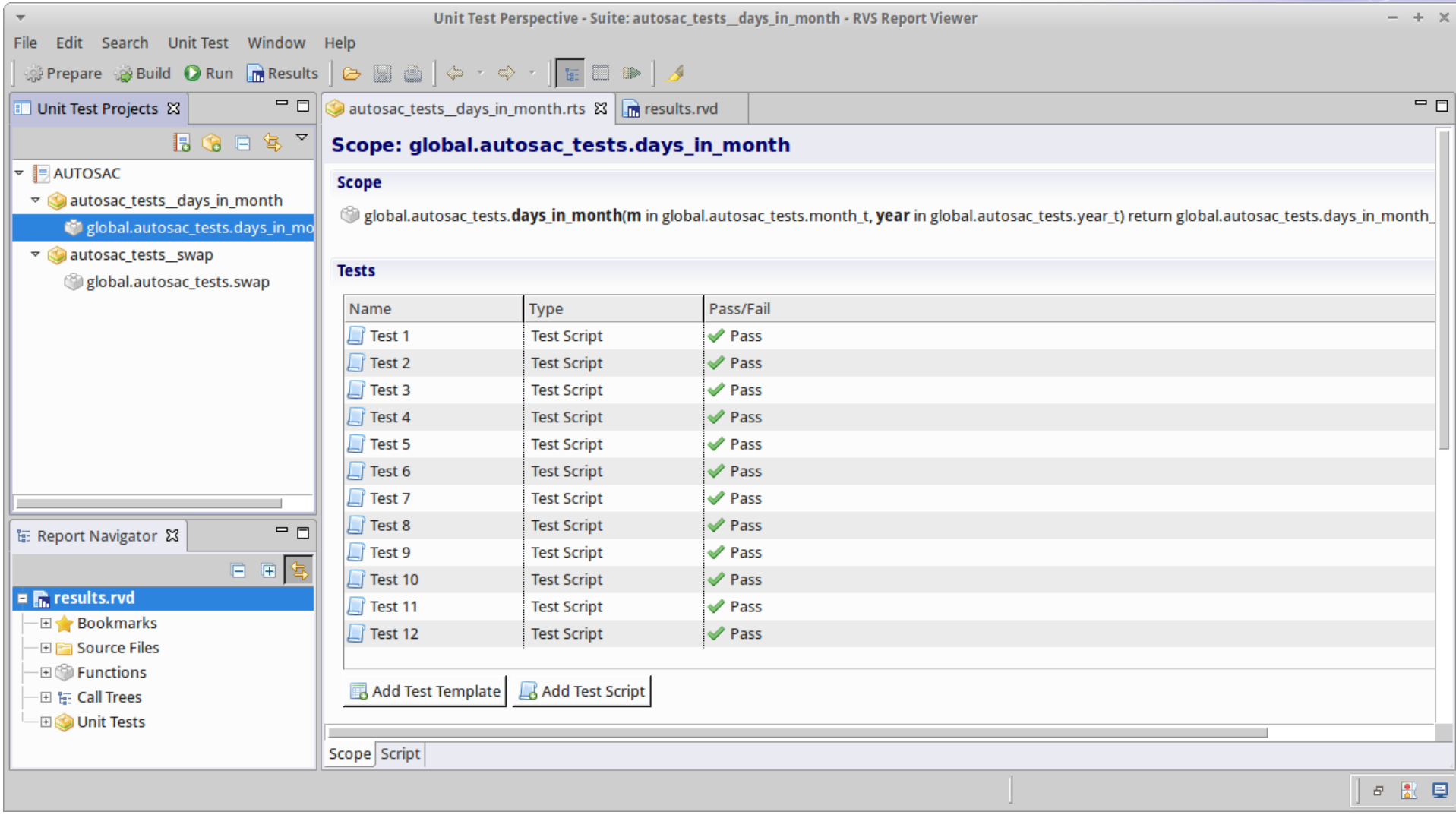
RapiTest Framework Script

```
suite "autosac_tests__days_in_month" is

scope global.autosac_tests.days_in_month (
     in param.m as global.autosac_tests.month_t,
     in param.year as global.autosac_tests.year_t)
 return global.autosac_tests.days_in_month_t is

     /=--
       Test case
     --=/
     test "Test 1" is
        -- Run of the unit: days_in_month
        run is
            param.m := January;
            param.year := 1000;
        end run;
      end test;
      test "Test 2" is
         run is
          param.m := February;
           param.year := 1000;
        end run;
     end test;
```

Driver Code

```
<...>

-- Adding tests
    RVS_RTS_Ext.Begin_Test(1738044706);
declare
     RVS_RTS_LOCAL_VAR_1733935649 :
     standard.autosac_tests.days_in_month_t :=
     standard.autosac_tests.days_in_month( m =>
     standard.autosac_tests.month_t`
        (autosac_tests.january),
     year => 1000 );
begin
      null; -- Any post-call assertions here...
end;


<...>
```

Test Results



Test results? Post Condition Is the Test Oracle

```
function Days_In_Month (M : Month_T;
                        Year : Year_T)
return Days_In_Month_T
 with Post =>
 Days_In_Month'Result =
 (case M is
  when September | April | June | November =>
    30,
  when February =>
    (if Year mod 100 = 0 then
    (if Year mod 400 =0 then 29 else 28)
else
    (if Year mod 4 = 0 then 29 else 28)),
     when others => 31);
```

Test success means post-condition evaluates true.

Q: how completely can the post condition describe the test result?

Test cases to explore discontinuities

```
function I_sqrt (M : Natural) return integer
with Post =>
I_sqrt'Result ** 2 <= M
and
(I_sqrt'Result + 1) ** 2 > M;
```

i_sqtr(63)=7

49 <= 63

64 > 63

Human tester might look at tests like:

- 0, 1, 2, 63, 64, 65, Integer'Last

How can a computer seek similar results?

- Currently CBMC would only produce one test case + out of range errors

(Future work)

Applicability to DO-178C

6.4.3c: This testing method should concentrate on demonstrating that each software component complies with its low-level requirements. Requirements based low-level testing ensures that the software components satisfy their low-level requirements. Typical errors revealed by this testing method include:

1 Failure of an algorithm to satisfy a software requirement;

2 Incorrect loop operations

3 Incorrect logic decisions

4 Failure to process correctly legitimate combinations of input conditions

5 Incorrect responses to missing or corrupted input data

6 Incorrect handling of exceptions, such as arithmetic faults or violations of array limits

7 Incorrect computation sequence

8 Inadequate algorithm precision, accuracy, or performance

Independence of compiler?

Common mode failure: the compiler ?

- Compiler generates test code
- AND compiler generates test-oracle (executing post-conditions). Normally the test results generated by a tester (greater independence)

Risk of common-mode failure in the compiler?

- The diversity of the specification and implementation would be enforced through coding standards that kept a separation and hence diversity between contracts and implementation.

End-to-end toolchain established and working on basic examples

- CMBC -> analysing SPARK post-conditions
- RapiTest Framework -> test scripts, execution, coverage etc.
- SPARK examples and case studies in preparation

Looking forward to seeing evaluation in case studies.

Conclusion

Basic idea:

- Use power of SPARK post conditions to generate tests
  - (Or at least get a head-start!)

How?

- SPARK -> CMBC ->RapiTest Framework

Benefit

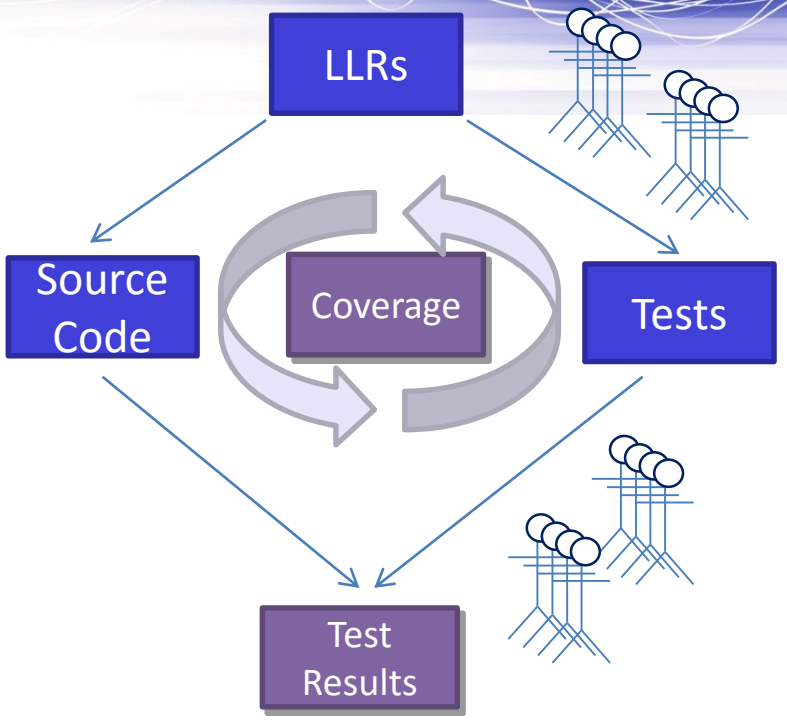
- Reduce manual effort on creating tests, reviewing tests, executing tests, maintaining tests

Status

- Prototype – 2 case studies coming up

Looking for beta test…

- ianb@rapitasystems.com



www.RapitaSystems.com/blog