Contract–based design and verification using SPARK 2014

Simon Buist  Stuart Matthews  Thomas Wilson

12 June, Ada Europe 2019, Warsaw
Agenda

• Introduction
• Context of the system
• Worked example
  • Test–driven development (TDD)
  • Contracts
• How contracts affected design & verification
• Benefits of using contracts
Introduction

• This talk details the practical experience of using SPARK 2014 contracts in the implementation of a critical system.
• It is a high safety-integrity system compliant with UK DEF STAN 00-56.
Figure 1: Boiler
Figure 2: SCADE Requirement
Comparing TDD with Contracts

TDD
SCADE → English Specification → Test → Implementation

Contracts
SCADE → Contract → Static analysis → Implementation
Worked example: TDD

English specification

[Pressure is ... of Pressure_1 and Pressure_2]

If [Status = Failed_Safe]
    in any previous cycle then [Status = Failed_Safe]
Otherwise, if [Pressure > Safety_Threshold] then
    [Status = Failed_Safe]
Otherwise, if [Pressure > Warning_Threshold] then
    [Status = Warning]
Otherwise, if [Pressure <= Warning_Threshold] then
    [Status = OK]
Otherwise, [Status = Failed_Safe]
Worked example: TDD

API

```plaintext
function Update (Old_State : State_T;
    Pressure_1 : Base_Types.Float64;
    Pressure_2 : Base_Types.Float64)

return Result_T;
```
Worked example: TDD

```plaintext
procedure Test_Calculate_Pressure
is
    Test_Initilise;
    Test_Step_Covers ("S.Calculate_Pressure.Scenario.1");
    Set_State (Old_State => State_T'(...),
                Pressure_1 => Base_Types.Float64 (0.0),
                Pressure_2 => Base_Types.Float64 (1.0));
    Check_Result(
        Result_T'(Boiler_Monitor_State => Monitoring;
                       Status => OK;
                       Valve => Closed));
    Test_Initilise; -- another test step...
end Test_Calculate_Pressure;
```
Worked example: Contracts

Figure 3: Contracts

prove AoRTE & prove functional correctness
Worked example: Contracts

Figure 4: SCADE Requirement
Worked example: Contracts

```plaintext
function Update (...) return Result_T
with Post => (
(Calculate_Pressure.Result_T'(
   State => Update'Result.State.Calculate_Pressure_1_State,
   Output_1 => Update'Result.State.Pressure) = Calculate_Pressure.Update (
   Old_State => Old_State.Calculate_Pressure_1_State, Input_1 => Pressure_1,
   Input_2 => Pressure_2)) and then
(if Old_State.Boiler_Monitor_States = Monitoring then
(if (Update'Result.State.Pressure > Constants.SCADE.Safety_Threshold) then
   Update'Result.State.Boiler_Monitor_States = Fail_Safe
elsif (Update'Result.State.Pressure > Constants.SCADE.Warning_Threshold) then
   Update'Result.State.Boiler_Monitor_States = Warning
else Update'Result.State.Boiler_Monitor_States = Old_State.Boiler_Monitor_States))
and then
(if Old_State.Boiler_Monitor_States = Warning then
(if (Update'Result.State.Pressure > Constants.SCADE.Safety_Threshold) then
   Update'Result.State.Boiler_Monitor_States = Fail_Safe
elsif (Update'Result.State.Pressure <= Constants.SCADE.Warning_Threshold) then
   Update'Result.State.Boiler_Monitor_States = Monitoring
else Update'Result.State.Boiler_Monitor_States = Old_State.Boiler_Monitor_States))
and then
(if Old_State.Boiler_Monitor_States = Fail_Safe then
   Update'Result.State.Boiler_Monitor_States = Old_State.Boiler_Monitor_States)
and then
(if Update'Result.State.Boiler_Monitor_States = Monitoring then
   Update'Result.Valve = Update'Result.State.Closed and then
   Update'Result.Status = Update'Result.State.Ok) and then
(if Update'Result.State.Boiler_Monitor_States = Warning then
   Update'Result.Valve = Update'Result.State.Closed and then
   Update'Result.Status = Update'Result.State.Warning) and then
(if Update'Result.State.Boiler_Monitor_States = Fail_Safe then
   Update'Result.Valve = Update'Result.State.Opened and then
   Update'Result.Status = Update'Result.State.Failed_Safe));
```
Worked example: Contracts

(if Old_State.Boiler_Monitor_States = Monitoring then
  (if (Update'Result.State.Pressure > Constants.SCADE.Safety_Threshold) then
    Update'Result.State.Boiler_Monitor_States = Fail_Safe
  elseif (Update'Result.State.Pressure > Constants.SCADE.Warning_Threshold) then
    Update'Result.State.Boiler_Monitor_States = Warning
  else Update'Result.State.Boiler_Monitor_States = Old_State.Boiler_Monitor_States))

13 Contract–based design and verification using SPARK 2014
function Update (Old_State : State_T;
   Pressure_1 : Base_Types.Float64;
   Pressure_2 : Base_Types.Float64) return Result_T
is
   Result : Result_T;
begin
   Result.State.Pressure := Calculate_Pressure.Update (
      Old_State => Old_State.Calculate_Pressure_1_State,
      Input_1 => Pressure_1,
      Input_2 => Pressure_2).Output_1;
   Result.State.Calculate_Pressure_1_State := Calculate_Pressure.Update (
      Old_State => Old_State.Calculate_Pressure_1_State,
      Input_1 => Pressure_1,
      Input_2 => Pressure_2).State;
   Result.State.Boiler_Monitor_States := (if
      (Old_State.Boiler_Monitor_States = Monitoring) then (if (Result.State.Pressure > Constants.SCADE.Safety_Threshold) then Fail_Safe else (if (Result.State.Pressure > Constants.SCADE.Warning_Threshold) then Warning else Old_State.Boiler_Monitor_States)) else (if (Old_State.Boiler_Monitor_States = Warning) then (if (Result.State.Pressure > Constants.SCADE.Safety_Threshold) then Fail_Safe else (if (Result.State.Pressure <= Constants.SCADE.Warning_Threshold) then Monitoring else Old_State.Boiler_Monitor_States)) else Old_State.Boiler_Monitor_States));
   return Result;
end Update;
Comparing TDD with Contracts

Figure 5: TDD

procedure Update (...) is
  local1 := operator_x(input1, input2);
  local2 := operator_y(local1, local3);
  local3 := operator_z(local2, input4);
  local4 := operator_w(local3, local5, input5);
  local5 := operator_v(local4, local6);
  if state = state_1 then
    if input6 = value_6 then
      state := state_2;
      output := output_1;
    else
      state := state_3;
      end if;
  else
    state := state_4;
  end if;
  if local7 = state_7 then
    if not local7 = state_7 then
      state := state_5;
      end if;
  else
    state := state_6;
  end if;
  if not local8 = state_8 then
    if not local8 = state_8 then
      state := state_9;
      end if;
  else
    state := state_10;
  end if;
  output := state = state_11;
end update;

procedure TC_Test_Feature ("*:in out Test") is
  Test_Initialize;
  Test_Cover ("S:WhateverSpecFeature_1");
  x := 1;
  y := 2;
  Set_State (x => x, y => y);
  Check_Thing_True;
  Check_Other_Thing;
  Test_Initialize;
  Test_Cover ("S:WhateverSpecFeature_2");
  ...
end TC_Test_Feature;
Comparing TDD with Contracts

Figure 6: Contracts

SCADE to Contracts

prove AoRTE & prove functional correctness
How contracts affected verification

Verification of the system takes a hybrid approach, using both proof and test to establish functional correctness of the implementation. The SPARK 2014 contracts play a role in both these verification activities.
How contracts affected verification: dynamic (testing)

• Run-time checking of the contracts ensures they are always met during system testing, because we’re using Ada 2012 contracts.
• Even though functional correctness had been proven, the run-time checking found an error in a low-level interrupt handler.

Verification: delivered executable

• Using the flag \--gnata, we left the contracts built-in to the delivered executable.
• We designed the system so that any failure of such a run-time check will have the effect of putting the system into a safe state.
Run-time checking of contracts

When we used 64-bit floating point operations within interrupt handlers for the first time, if the interrupt handler interrupted a floating point operation then the top 32-bits of the registers could be corrupted.

Figure 7: Register corruption
Conclusion

- Zero defects found in code derived from SCADE specifications.
- Leaving run-time checks in found fault on target bootloader.
- We found a viable & practicable technique for proving correctness against the SCADE specification.

Acknowledgement: This work was supported by the SECT-AIR project, funded by the Aerospace Technology Institute and Innovate UK, as project number 113099.