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.



Embedded Protection Subsystem



4 Contract-based design and verification using SPARK 2014





Figure 2: SCADE Requirement

Comparing TDD with Contracts

TDD

 $\text{SCADE} \rightarrow \text{English Specification} \rightarrow \text{Test} \rightarrow \text{Implementation}$

Contracts

 $\text{SCADE} \rightarrow \text{Contract} \rightarrow \text{Static analysis} \rightarrow \text{Implementation}$





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]
```

7 Contract-based design and verification using SPARK 2014

API



8 Contract-based design and verification using SPARK 2014

Test

```
procedure Test_Calculate_Pressure
is
   Test_Initialise;
   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_Initialise; -- another test step...
end Test_Calculate_Pressure;
```

Worked example: Contracts review SCADE to - ** SJuncs.Operator1 ** - [A direct implementation of the Scade Operator1 operator is provided.] Contracts ackage Operator 1 -- Update -- Description: Performs one cycle of execution for the operator. function Update (Old State : State T. Input_1 : Boolean Input 2 : Boolean Input 3 : Boolean SCADE input_4 : Boolean Input 5 : Base Types Int 32) return Result T with Post => ((if Old_State.SM_1 = State_1 then to Of Onput 4 and then Onput 5 = Constants.SCADE.value 1)] then Update'Result.State.SM 1 = State 2 Impl. els# ((mout 4 and then (mout 5 = Constants SCADE Value 2)) Update'Result.State.SM 1 = State 4 elst ((input 4 and then (input 5 = Constants SCADE Value 2)) precedure update (...) Update/Result.State.SM_1 = State_3 Update'Result.State.SM 1 = Old State.SM 11) and then local_1 := Operator_2.update(rp) local_2 := Operator_3.update(local_3 := Operator_4.update(L) local_4 := Operator_4.update(L) local_5 := Operator_5.update(L) Troat 31 inpat_3); Local_2).output_1; Local_2).output_2; pragma Annotate (Xcov, Exempt Off); if State = State_1 then if Input_4 then if input_5 = value_1 then State is State 2: elsif Local_4 then State := State 4 6168 state := state 3: end if; and if;

prove AoRTE & prove functional correctness

Figure 3: Contracts

elsif State = State_3 then

if Local_4 then
State := State_4;
elsif not Local_5 then
State := State_5;

end 1f; elsif state = state_4 then if not local_4 then state := state_5; end if;

Worked example: Contracts



Figure 4: SCADE Requirement

Worked example: Contracts

```
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));
12 Contract-based design and verification using SPARK 2014
```

Worked example: Contracts



Autogenerated Ada body

```
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));
Result.Valve := (if (Result.State.Boiler_Monitor_States = Monitoring) then
Result.State.Closed else (if (Result.State.Boiler_Monitor_States = Warning)
then Result.State.Closed else Result.State.Opened));
   Result Status := (if
(Result.State.Boiler_Monitor_States = Monitoring) then Result.State.Ok else (if
(Result.State.Boiler_Monitor_States = Warning) then Result.State.Warning else
Result.State.Failed Safe)):
     return Result:
end Update;
```

14 Contract-based design and verification using SPARK 2014

Comparing TDD with Contracts



Figure 5: TDD

15 Contract-based design and verification using SPARK 2014

Comparing TDD with Contracts



Figure 6: Contracts

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



19 Contract-based design and verification using SPARK 2014

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.