Carleton UNIVERSITY

Generating Verifiable Test Scenarios

Jean-Pierre Corriveau and Wei Shi
Carleton University, Ottawa, Canada

July 20th, SERP 2011

The Context

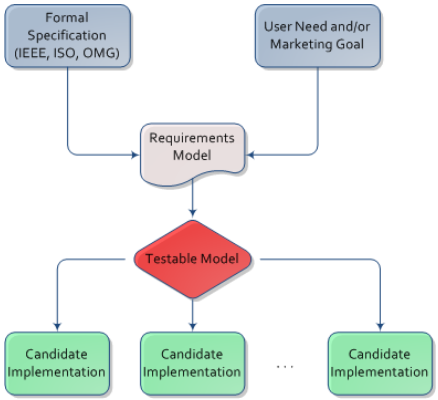
- A university is looking for a new administrative system.
- Several candidate systems are to be tested.
  - Existing systems, as well as outsourcing, are considered.
- Each such candidate system is a black-box to us:
  - We want to develop a single test plan that is implementation independent
  - Vendors are not giving us their source code.
    - So, for such requirements testing, they bear the onus of developing tests corresponding to our test plan
- This does not sound like rocket science but…

Key Questions

- How are we to capture our requirements?
- How are we to obtain a test plan from our requirements?
- How are we to know that the actual test suite corresponds to our test plan?

- Immediate question: where to start?
  - Debugging is NOT our problem
  - Nor are formal methods, program verification, assertion deduction!
  - Unit testing is not relevant
  - Testing automation frameworks assume you have tests…
  - TDD and other code-centric methods do not capture requirements…

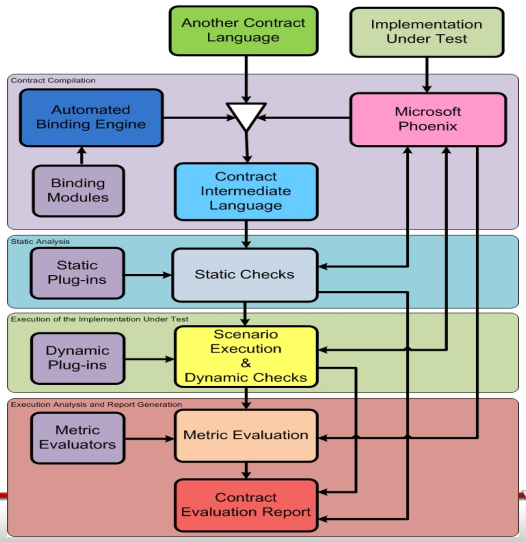
Model-Based Testing



The MBT Challenge: capture a testable requirements model at a level of abstraction that is implementation independent yet executable!

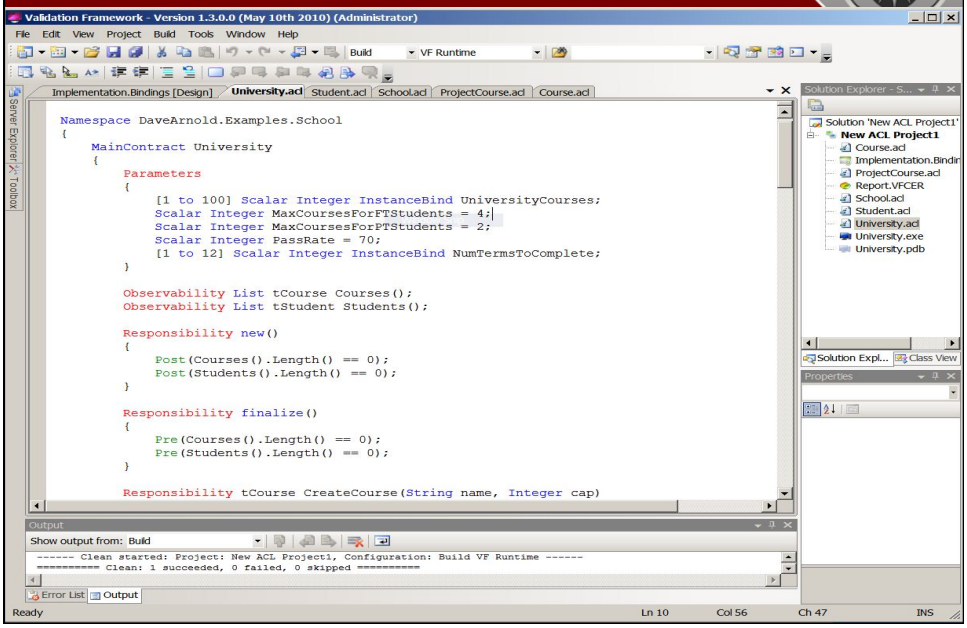
The VF Framework (1)

URL:
http://vf.davearnold.ca/



Carleton UNIVERSITY

The VF Framework (2)



Our Approach

- We support a testable requirements model capable of automatically generating executable checks.
- This testable model supports:
  - The capture of functional and non-functional requirements
  - Testability of the requirements model AND traceability to code through automatically generated bindings
  - Traceability and executability of the generated checks
  - Semantics rooted in the notions of *responsibilities* and *scenarios* from the URN
  - Abstraction of the testable model over several possible implementations
- To the best of our knowledge NO other existing tool for requirements validation offers all of the above characteristics…

Gist

- Each entity of the problem (not the solution) space has a contract associated with it.
- A contract is essentially expressed as a set of i) responsibilities, ii) scenarios, and iii) an optional invariant that every instance of each class bound to this contract must obey.
- For scenario testing:
  - Each scenario defines a grammar of responsibilities.
  - Each responsibility being bound, indirectly a scenario defines a grammar of proc. Calls
  - The VF automatically instruments in the IUT the monitoring code for each scenario.
  - A contract may have several instances and each instance may have several instances of its scenarios. All of this is taken care of by the VF.
- Controlled scenario testing: For each scenario
  - Path sensitization variables (PSV) are identified by the VF
  - The user must supply for each PSV a range or a set of possible values
  - ACL test scenarios are generated and must be bound to corresponding test procs.

Back to our Example

Consider some of the scenarios found in the administrative system of a university:

- Depending on her status (FT, PT, LOA), a student can register in a number of courses between the minimum and the maximum allowed by her status, provided each selected course is not full.
- In each course she takes, a student has to complete assignments and/or projects and/or a midterm and/or a final. Temporal ordering is unknown. Furthermore, a student can drop a course at any point during the term…
- At the end of the term, the mark of each student in each course offered that term must be reported to the university.

The paper are such scenarios to be specified and validated?

An Example Contract

```
Import Core;

Namespace My.Examples.School
{
   Contract Course
   {
         Parameters
         {        Scalar Boolean InstanceBind HasFinal = {default true, false  };
         }

         Observability Integer MarkForStudent (tStudent student);

         IsFullCheck { Students().Length() <= CapSize();  }

         Responsibility AddStudentNoPreReqCheck(tStudent s)
                  { Pre(Students().Contains(s) == false); Execute();
                    Post(Students().Contains(s) == true);
                  }
```

An Example Contract

```
Scenario ReportMarks{
    Trigger(observe(TermEnded)),
    once Scalar Contract University u = instance;
     each (Students())
//built-in variable iterator accesses students one at a time
       {u.ReportMark(context, iterator, MarkForStudent(iterator))},
Terminate(fire(MarksRecorded));      }

Exports
{ Type tStudent conforms Student { University::tStudent; }
}}} //end of Course contract }
```


An Example Contract

```
Namespace Examples.School {
Contract Student  {
Scenario RegisterForCourses {
Scalar tCourse course;      Contract University u = instance;
Trigger(observe(CoursesCreated), IsCreated()),
choice(IsFullTime()) true
   {  ( atomic
   { course = SelectCourse(u.Courses()),
      //via bindpoint get the instance for the selected course
      choice(course.bindpoint.IsFull())  true
         {  course = SelectCourse(u.Courses()),
            redo}   //until a course is not full
    },
u.RegisterStudentForCourse(context, course),
   RegisterCourse(course)
  )  [0-u.Parameters.MaxCoursesForFTStudents]
     //repeat up to the max # of courses for a full time
   }
```

An Example Contract

```
alternative(false)  //student is part-time (PT)
{ ( atomic
    {  course = SelectCourse(u.Courses()),
       choice(course.bindpoint.IsFull())  true
        { course = SelectCourse(u.Courses()),
          redo  }
     },
    u.RegisterStudentForCourse(context, course),
    RegisterCourse(course)
    )[0-u.Parameters.MaxCoursesForPTStudents]
 },
Terminate();  } //end of scenario RegisterForCourses
```


An Example Contract

```
Scenario TakeCourses {
failures = 0; //number of failures in the current term
Trigger(observe(TermStarted)),
parallel // for all courses of that term
{  Contract Course course = instance;
   Check(CurrentCourses().Contains(course.bindpoint));
   atomic
     { (parallel            //can do assignmnts, midterm, proj concurrently
          { (DoAssignment(course.bindpoint)) [course.Parameters.NumAssignments]  }
     |
         (DoMidterm(course.bindpoint)) [course.Parameters.NumMidterms]
     |   (DoProject(course.bindpoint))  [course sameas ProjectCourse &&
                 course.Parameters.HasProject]),
      (DoFinal(course.bindpoint) [course.Parameters.HasFinal]              }
    alternative( not observe(LastDayToDrop))
      { DropCourse(course.bindpoint) }
}[CurrentCourses().Length()];
Terminate();  } //end of scenario TakeCourses
```

An Example Contract

```
Import Core;

Namespace DaveArnold.Examples.School
{
   MainContract University
   {
         Parameters
         {
                  [1-100] Scalar Integer InstanceBind UniversityCourses;
                  Scalar Integer MaxCoursesForFTStudents = 4;
                  Scalar Integer MaxCoursesForPTStudents = 2;
                  Scalar Integer PassRate = 70;
                  [1-12] Scalar Integer InstanceBind NumTermsToComplete;
         }

         Observability List tCourse Courses();
         Observability List tStudent Students();
```


An Example Contract

```
Responsibility new() {
   Post(Courses().Length() == 0);
   Post(Students().Length() == 0);
}
Responsibility finalize() {
   Pre(Courses().Length() == 0);
   Pre(Students().Length() == 0);
}

Responsibility tCourse CreateCourse(String name, Integer cap) {
   once Scalar Integer oldSize;
   oldSize = PreSet(Courses().Length());
   Post(value.bindpoint.Name() == name);
   Post(value.bindpoint.CapSize() == cap);
   Post(Courses().Length() == oldSize + 1);
   Post(Courses().Contains(value) == true);
}
```

An Example Contract

```
Responsibility ReportMark (tCourse course, tStudent student, Integer mark) {
     choice(mark) < Parameters.PassRate
       { student.bindpoint.failures = student.bindpoint.failures + 1; }

Responsibility RegisterStudentForCourse(tStudent student, tCourse course);

Responsibility CancelCourse(tCourse course)  {
          Pre(Courses().Contains(course) == true));
          Post(Courses().Contains(course) == false));              }

Responsibility CalculatePassFail()          {
          each(Students())
           choice(iterator.bindpoint.failures) >= 2
                    FailStudent(iterator);
           alternative
                    PassStudent(iterator);              }
```


An Example Contract

```
Scenario Term {
    Trigger(new()),
    (
          CreateCourse()[Parameters.UniversityCourses],
          TermStarted(),
          fire(TermStarted),
          LastDayToDrop(),
          fire(LastDayToDrop),
          TermEnded(),
          fire(TermEnded),
          observe(MarksRecorded)[Parameters.UniversityCourses],
          CalculatePassFail(),
          DestroyCourse()[Parameters.UniversityCourses],
          fire(TermComplete)
    )+,
    Terminate(finalize());
}
```

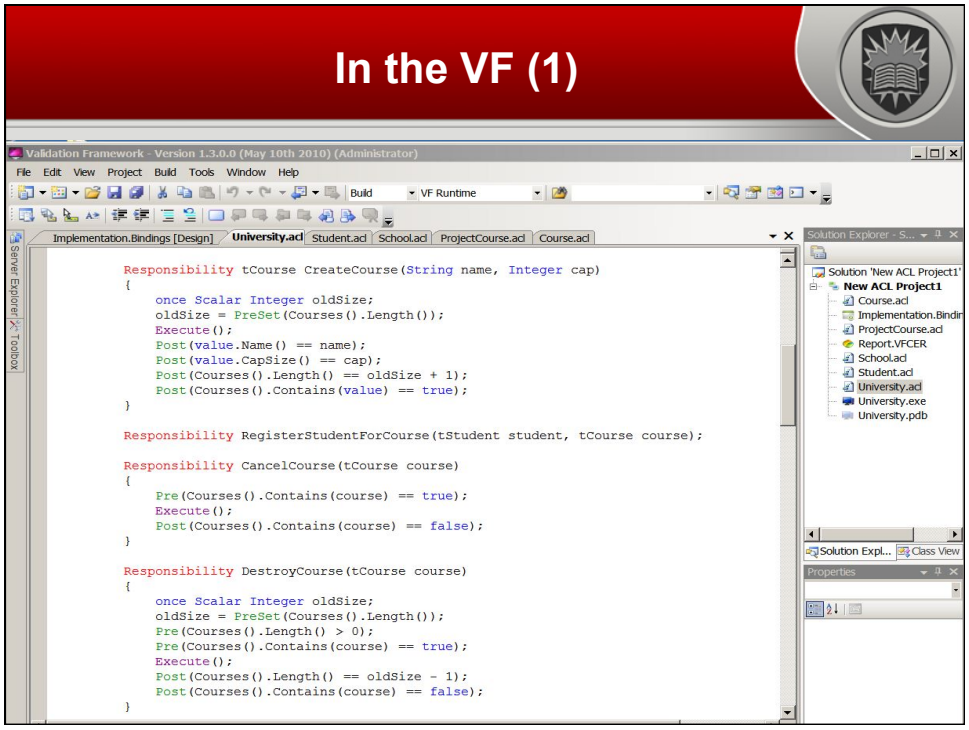
An Example Contract

```
Exports
{
    Type tCourse conforms Course
    {
          Student::tCourse;
    }
    Type tStudent conforms Student
    {
          Course::tStudent;
    }
}
```

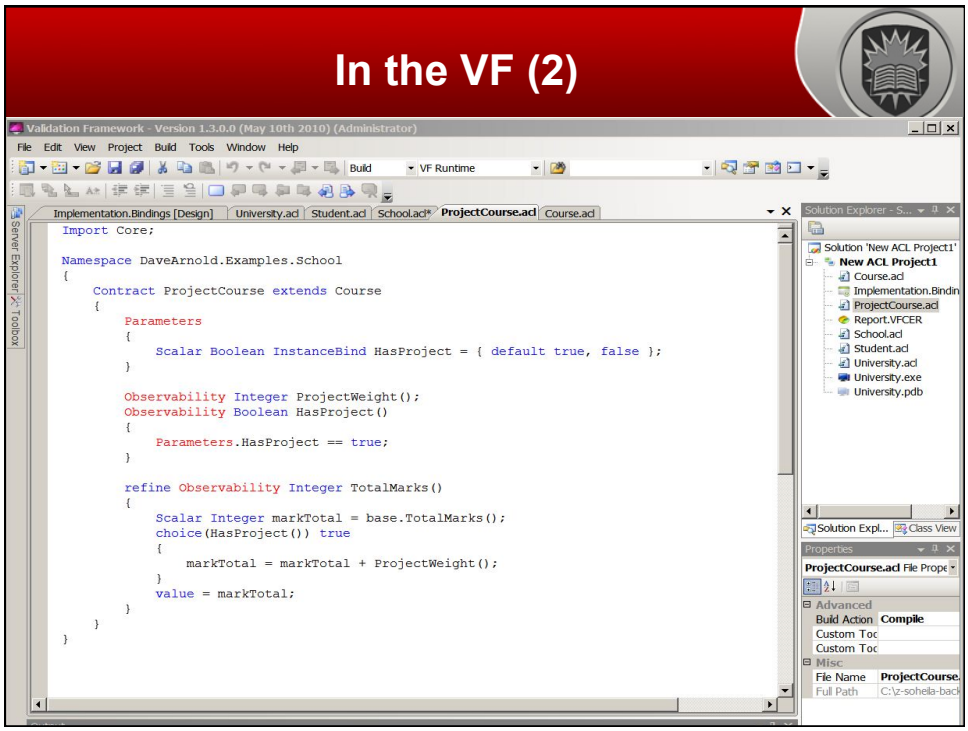

Conclusion

- Challenges for state-based tools:
  - Multiple and Dynamic instances
  - Concurrent states
  - Loops
  - Observationally complete wrapper for the IUT…
  - Traceability (esp. in light of state explosion solutions)
  - Scalability
- Questions / Comments?

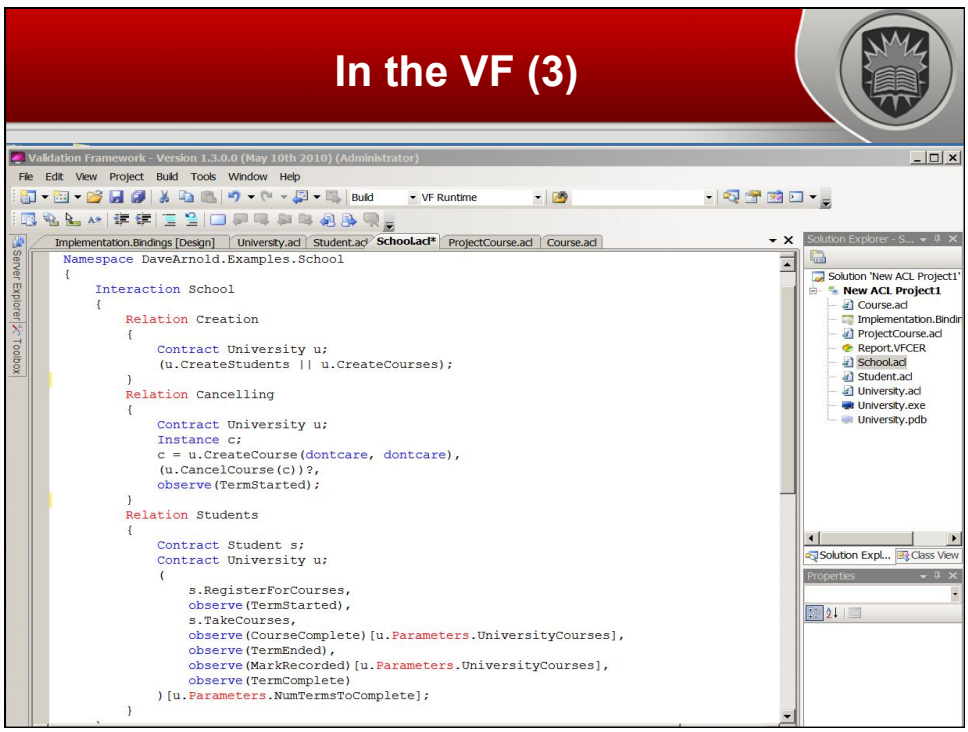

Carleton
UNIVERSITY
Slide - 21


In the VF (1)
Validation Framework - Version 1.3.0.0 (May 10th 2010) (Administrator)
File Edit View Project Build Tools Window Help
Build
VF Runtime
Implementation.Bindings [Design]
University.acl
Student.acl
School.acl
ProjectCourse.acl
Course.acl
Responsibility tCourse CreateCourse(String name, Integer cap)
{
once Scalar Integer oldSize;
oldSize = PreSet(Courses().Length());
Execute();
Post(value.Name() == name);
Post(value.CapSize() == cap);
Post(Courses().Length() == oldSize + 1);
Post(Courses().Contains(value) == true);
}
Responsibility RegisterStudentForCourse(tStudent student, tCourse course);
Responsibility CancelCourse(tCourse course)
{
Pre(Courses().Contains(course) == true);
Execute();
Post(Courses().Contains(course) == false);
}
Responsibility DestroyCourse(tCourse course)
{
once Scalar Integer oldSize;
oldSize = PreSet(Courses().Length());
Pre(Courses().Length() > 0);
Pre(Courses().Contains(course) == true);
Execute();
Post(Courses().Length() == oldSize - 1);
Post(Courses().Contains(course) == false);
}
Solution Explorer - S...
Solution 'New ACL Project1'
New ACL Project1
Course.acl
Implementation.Bindin
ProjectCourse.acl
Report.VFCER
School.acl
Student.acl
University.acl
University.exe
University.pdb
Solution Expl...
Class View
Properties
Server Explorer
Toolbox

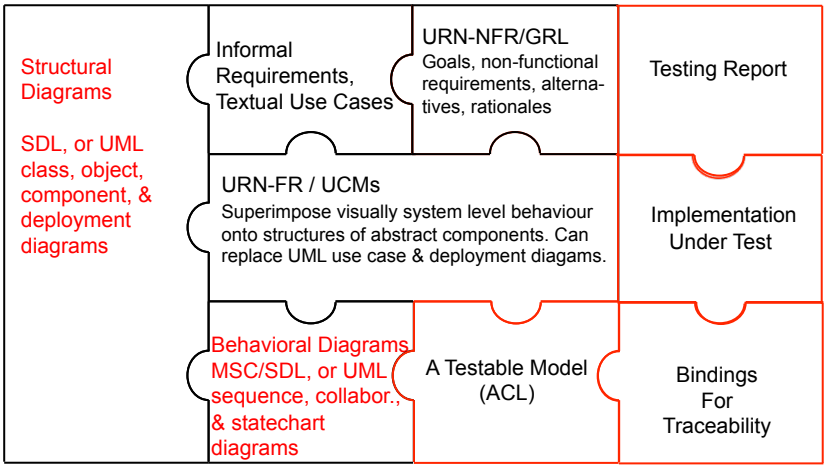
In the VF (2)



In the VF (3)



Scope of our Work



Carleton UNIVERSITY

Slide - 25

On Capturing Requirements

We distinguish 3 ‘schools’:

- Formal
  - Require hard-to-find expertise
  - Unified? Executable? Traceable to code?
- Code-based
  - Modeling is minimized => no testable model
  - Agile methods (e.g., TDD) advocate intensive collaboration
- Model-Based
  - Testable? Unified? Executable?
  - Full code generation DOES require implementation-aware designers!

Carleton UNIVERSITY

Bindings

- We need to connect the testable requirements model (in ACL) to the Implementation Under Test (IUT)
  - This is accomplished through the notion of *bindings*
  - Bindings are a mapping between an ACL element and a IUT element:
    - Contracts → Types (Classes, Structs)
    - Observabilities → A single method or property
    - Responsibilities → One or more methods

Bindings

- In order to reduce the dependency on manual binding
  - We use binding extension modules to **infer** as many bindings as possible
    - Modules can be written by third-party developers.
  - When a binding cannot be inferred, a short list of possible bindings is presented, and the user is asked to make a selection
    - Each of our two sample modules achieve 95% of the required bindings
- Each binding extension module can target different implementation styles or development methodologies.

CIL Generation

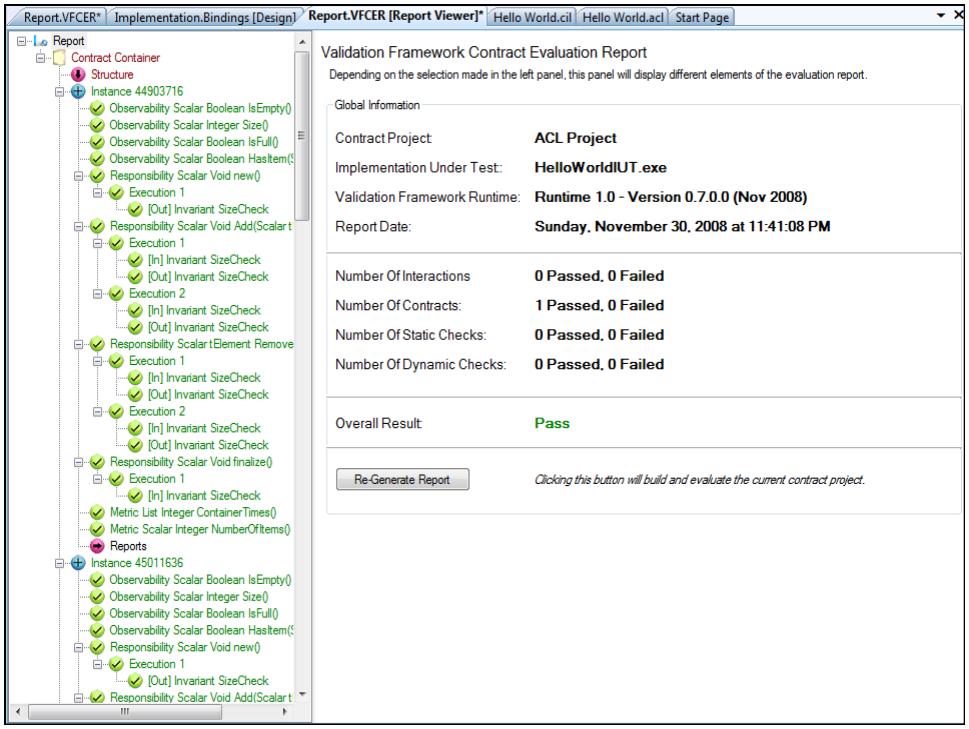
- Once the binding process is complete, the ACL and binding tables are used to generate a Contract Intermediate Language (CIL) representation.
  - Low level stack-based language
  - Designed so that other high level contract languages can be used with the runtime
    - Possibly graphical representations

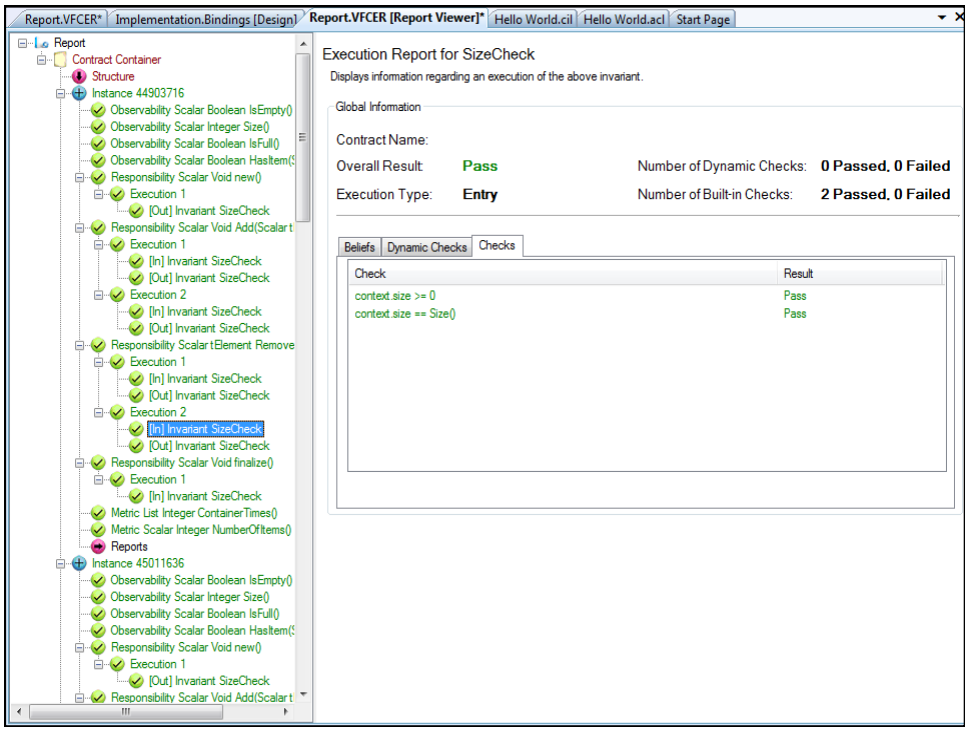
Execution

- The CIL is then compiled and executed against the IUT
  - Static checks are evaluated
  - The IUT is launched against the runtime
    - Execution is monitored for responsibilities and scenarios
    - Observabilities are invoked as needed by the runtime
    - Metric information is captured (Performance, Security, etc)
  - Metric evaluators determine results based on gathered metric information
- The result is a Contract Evaluation Report (CER)…

Contract Evaluation Report

- The CER provides information on the IUT's execution:
  - Static evaluation results
  - For each object instance
    - Information pertaining to any dynamic checks
    - Information regarding the pass/fail of observabilities, responsibilities, and scenarios
      - Preconditions
      - Post-conditions
      - Invariants
      - Beliefs
      - Dynamic Checks
    - The result of metric analysis





Additional Features

- The VF is also able to support
  - Contract refinement/inheritance
  - Atomic / parallel scenario blocks
  - Support for execution against web applications
- The VF consists of 1,355 classes totaling over 260,000 lines of C# and C++ source code

Validation of Our Approach

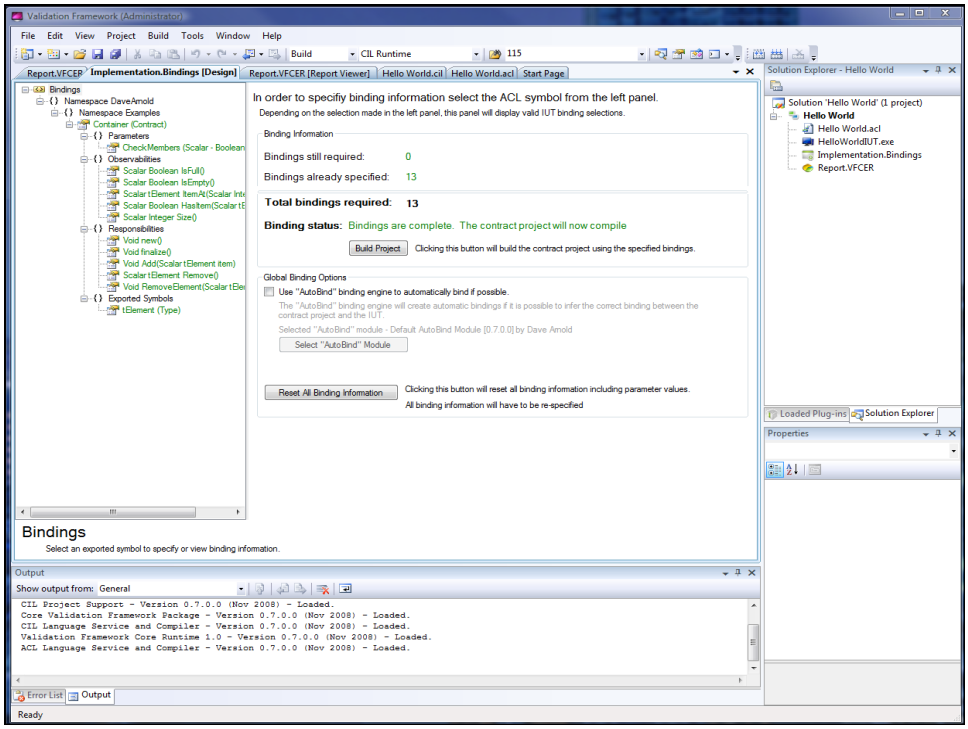
- Validation of our approach included
  - Individual testing of the ACL and CIL compilers
    - 1,516 individual tests performed
  - Five case studies
    - Basic container
    - Advanced container
    - Web login
    - Grocery store
    - University course registration and term operation
  - Use by a group of graduate students
    - Verify existing case studies
    - Develop small to medium size projects

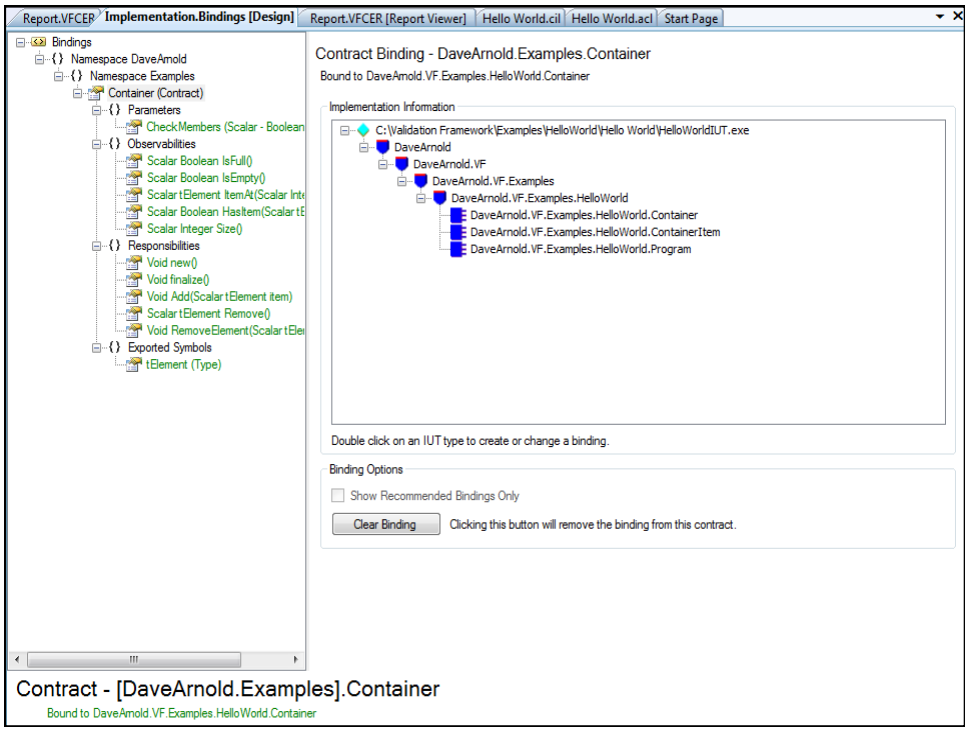
Contributions

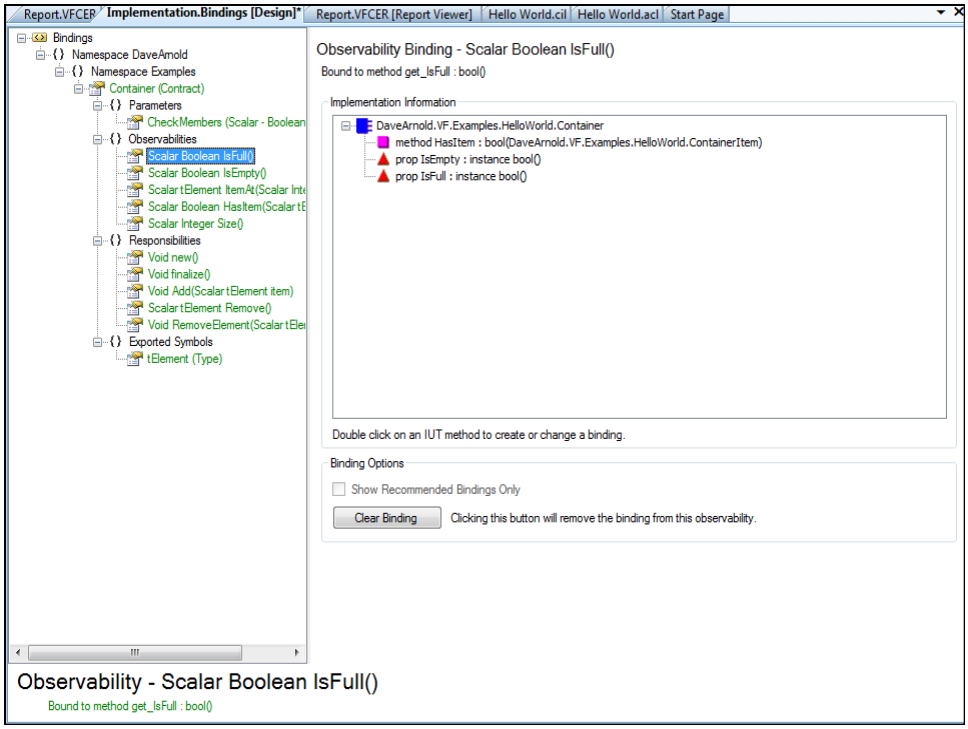
- Our TRM and supporting VF contribute in the areas of requirements engineering and validation by:
- Proposing a new set of requirements for a requirements model that supports operational validation. This set being the first, to the best of our knowledge, to include the following:
  - Capture of functional and non-functional requirements
  - Testability of the requirements model
  - Executabilty of checks generated from this testable model
  - Semantics rooted in the notions of responsibilities and scenarios
  - Abstraction of the testable model over several possible implementations
  - Openness to support specific static checks, dynamic checks, and metric evaluators
- Defining a TRM that satisfies these requirements (the ACL)
- Providing an open VF supporting the specification and execution of the TRM

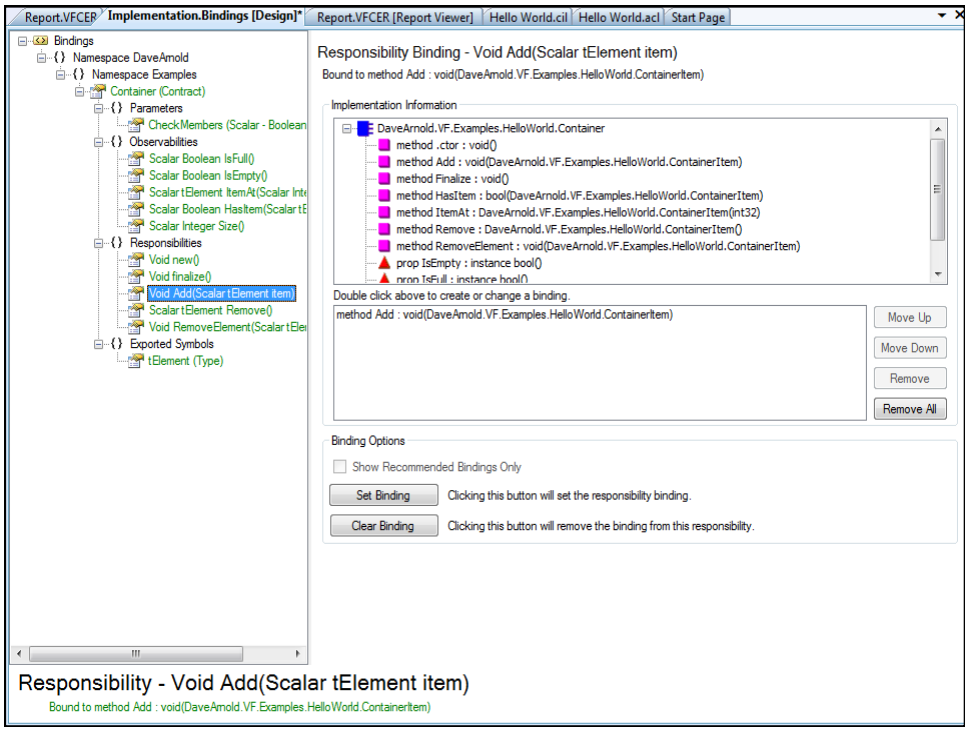
Future Work

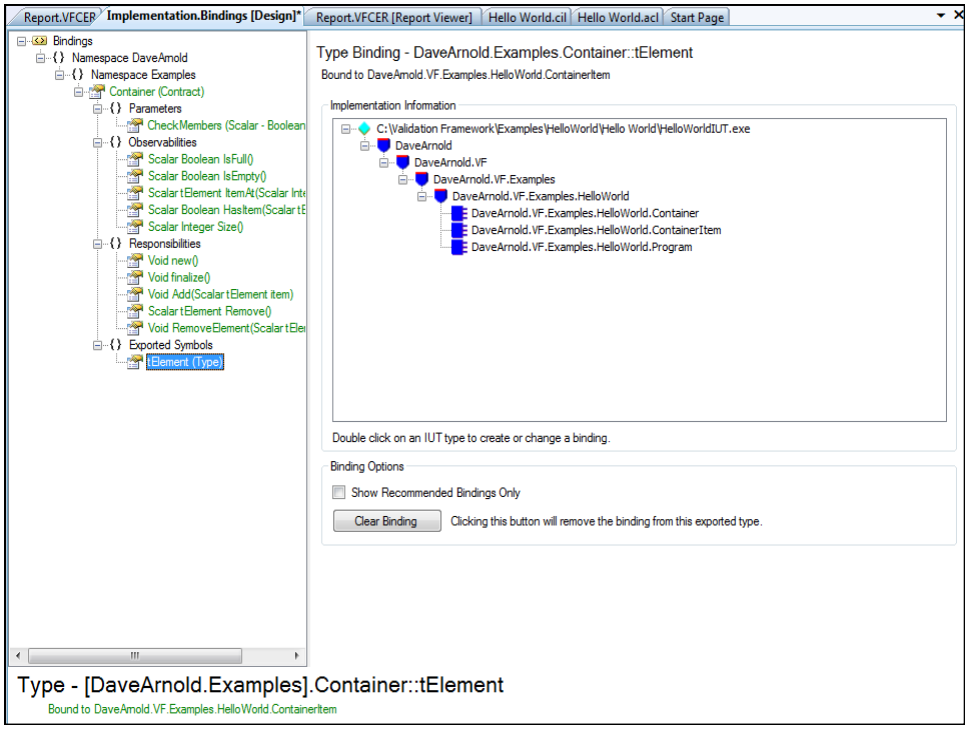
- Extension Through Openness
  - Additional high-level contract languages
    - Possibly domain specific
  - The creation of more AutoBind modules
  - The creation of more checks
    - Static checks
    - Dynamic checks
    - Metric evaluators




Report.VFCER
Implementation.Bindings [Design]
Report.VFCER [Report Viewer]
Hello World.cil
Hello World.acl
Start Page
Bindings
Namespace DaveArnold
Namespace Examples
Container (Contract)
Parameters
CheckMembers (Scalar - Boolean
Observabilities
Scalar Boolean IsFull()
Scalar Boolean IsEmpty()
Scalar tElement ItemAt(Scalar Int
Scalar Boolean HasItem(Scalar tE
Scalar Integer Size()
Responsibilities
Void new()
Void finalize()
Void Add(Scalar tElement item)
Scalar tElement Remove()
Void RemoveElement(Scalar tEle
Exported Symbols
tElement (Type)
Contract Binding - DaveArnold.Examples.Container
Bound to DaveArnold.VF.Examples.HelloWorld.Container
Implementation Information
C:\Validation Framework\Examples\HelloWorld\Hello World\HelloWorldIUT.exe
DaveArnold
DaveArnold.VF
DaveArnold.VF.Examples
DaveArnold.VF.Examples.HelloWorld
DaveArnold.VF.Examples.HelloWorld.Container
DaveArnold.VF.Examples.HelloWorld.ContainerItem
DaveArnold.VF.Examples.HelloWorld.Program
Double click on an IUT type to create or change a binding.
Binding Options
Show Recommended Bindings Only
Clear Binding
Clicking this button will remove the binding from this contract.
Contract - [DaveArnold.Examples].Container
Bound to DaveArnold.VF.Examples.HelloWorld.Container


Report.VFCER
Implementation.Bindings [Design]*
Report.VFCER [Report Viewer]
Hello World.cil
Hello World.acl
Start Page
Bindings
Namespace DaveArnold
Namespace Examples
Container (Contract)
Parameters
CheckMembers (Scalar - Boolean
Observabilities
Scalar Boolean IsFull()
Scalar Boolean IsEmpty()
Scalar tElement ItemAt(Scalar Int
Scalar Boolean HasItem(Scalar tE
Scalar Integer Size()
Responsibilities
Void new()
Void finalize()
Void Add(Scalar tElement item)
Scalar tElement Remove()
Void RemoveElement(Scalar tEle
Exported Symbols
tElement (Type)
Observability Binding - Scalar Boolean IsFull()
Bound to method get_IsFull : bool()
Implementation Information
DaveArnold.VF.Examples.HelloWorld.Container
method HasItem : bool(DaveArnold.VF.Examples.HelloWorld.ContainerItem)
prop IsEmpty : instance bool()
prop IsFull : instance bool()
Double click on an IUT method to create or change a binding.
Binding Options
Show Recommended Bindings Only
Clear Binding
Clicking this button will remove the binding from this observability.
Observability - Scalar Boolean IsFull()
Bound to method get_IsFull : bool()


Report.VFCER
Implementation.Bindings [Design]*
Report.VFCER [Report Viewer]
Hello World.cil
Hello World.acl
Start Page
Bindings
Namespace DaveArnold
Namespace Examples
Container (Contract)
Parameters
CheckMembers (Scalar - Boolean
Observabilities
Scalar Boolean IsFull()
Scalar Boolean IsEmpty()
Scalar tElement ItemAt(Scalar Int
Scalar Boolean HasItem(Scalar t
Scalar Integer Size()
Responsibilities
Void new()
Void finalize()
Void Add(Scalar tElement item)
Scalar tElement Remove()
Void RemoveElement(Scalar tEle
Exported Symbols
tElement (Type)
Responsibility Binding - Void Add(Scalar tElement item)
Bound to method Add : void(DaveArnold.VF.Examples.HelloWorld.ContainerItem)
Implementation Information
DaveArnold.VF.Examples.HelloWorld.Container
method .ctor : void()
method Add : void(DaveArnold.VF.Examples.HelloWorld.ContainerItem)
method Finalize : void()
method HasItem : bool(DaveArnold.VF.Examples.HelloWorld.ContainerItem)
method ItemAt : DaveArnold.VF.Examples.HelloWorld.ContainerItem(int32)
method Remove : DaveArnold.VF.Examples.HelloWorld.ContainerItem()
method RemoveElement : void(DaveArnold.VF.Examples.HelloWorld.ContainerItem)
prop IsEmpty : instance bool()
Double click above to create or change a binding.
method Add : void(DaveArnold.VF.Examples.HelloWorld.ContainerItem)
Move Up
Move Down
Remove
Remove All
Binding Options
Show Recommended Bindings Only
Set Binding
Clicking this button will set the responsibility binding.
Clear Binding
Clicking this button will remove the binding from this responsibility.
Responsibility - Void Add(Scalar tElement item)
Bound to method Add : void(DaveArnold.VF.Examples.HelloWorld.ContainerItem)


Report.VFCER
Implementation.Bindings [Design]*
Report.VFCER [Report Viewer]
Hello World.cil
Hello World.acl
Start Page
Bindings
Namespace DaveArnold
Namespace Examples
Container (Contract)
Parameters
CheckMembers (Scalar - Boolean
Observabilities
Scalar Boolean IsFull()
Scalar Boolean IsEmpty()
Scalar tElement ItemAt(Scalar Int
Scalar Boolean HasItem(Scalar t
Scalar Integer Size()
Responsibilities
Void new()
Void finalize()
Void Add(Scalar tElement item)
Scalar tElement Remove()
Void RemoveElement(Scalar tEle
Exported Symbols
tElement (Type)
Type Binding - DaveArnold.Examples.Container::tElement
Bound to DaveArnold.VF.Examples.HelloWorld.ContainerItem
Implementation Information
C:\Validation Framework\Examples\HelloWorld\Hello World\HelloWorldIUT.exe
DaveArnold
DaveArnold.VF
DaveArnold.VF.Examples
DaveArnold.VF.Examples.HelloWorld
DaveArnold.VF.Examples.HelloWorld.Container
DaveArnold.VF.Examples.HelloWorld.ContainerItem
DaveArnold.VF.Examples.HelloWorld.Program
Double click on an IUT type to create or change a binding.
Binding Options
Show Recommended Bindings Only
Clear Binding
Clicking this button will remove the binding from this exported type.
Type - [DaveArnold.Examples].Container::tElement
Bound to DaveArnold.VF.Examples.HelloWorld.ContainerItem

My Background

Jean-Pierre Corriveau

- At Nortel: one of the original creators of ObjecTime in 1986 (ROSE-RT)
  - Developed the notions of ports, optionals and plugIns
- In collaboration with F. Bordeleau and J.-P. Z-Zapata:
  - Worked with Raytheon on modeling for compliance the software radio specification
  - Developed a metamodel for the specification and one for design models
  - Suggested compliance consists in verifying correctness of mappings between both
- In collaboration with staff at Amazon and Bitheads:
  - Discussed outsourcing:
    - Testing essentially left to the developers!
    - Testing reduces mostly to unit testing…