



Carleton  
UNIVERSITY

## Generating Verifiable Test Scenarios

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

July 20<sup>th</sup>, SERP 2011


## The Context



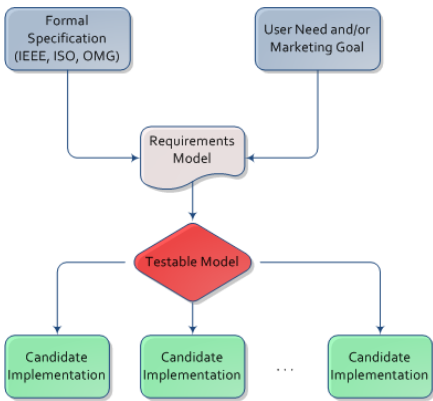
- A university is looking for a new administrative system.
- Several candidate systems are to be tested.
  - Existing systems, as well as outsourcing, are considered.
- Each such candidate system is a black-box to us:
  - We want to develop a single test plan that is **implementation independent**
  - 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...


Slide - 3


## Model-Based Testing

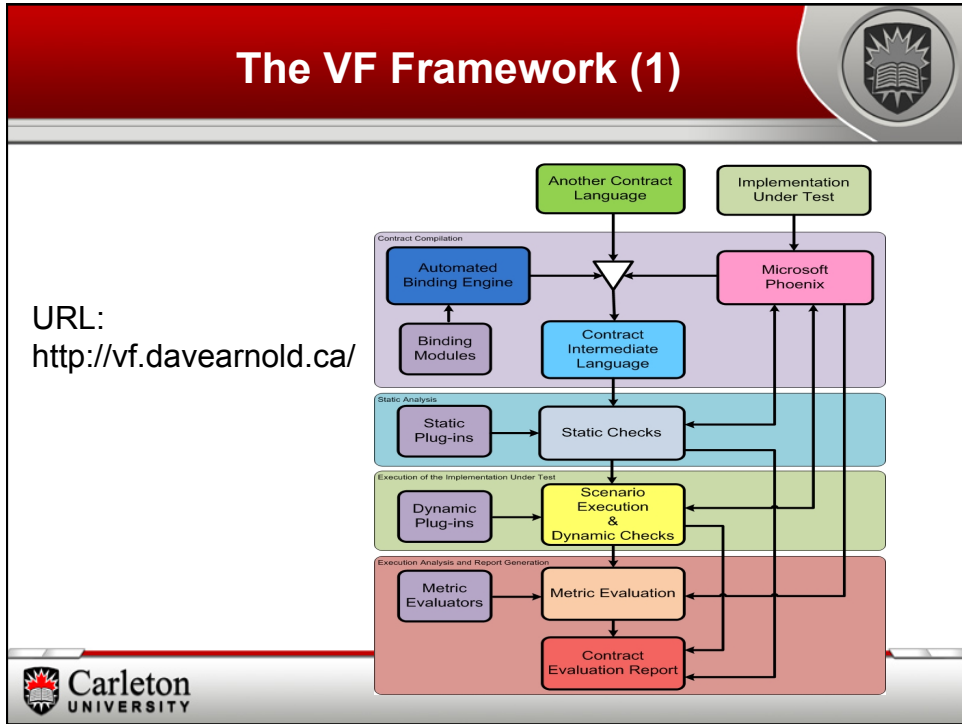


```

graph TD
    A[Formal Specification  
(IEEE, ISO, OMG)] --> B[Requirements Model]
    C[User Need and/or  
Marketing Goal] --> B
    B --> D{Testable Model}
    D --> E[Candidate Implementation]
    D --> F[Candidate Implementation]
    D --> G[Candidate Implementation]
    
```

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


Slide - 4



## The VF Framework (2)

```
Namespace DaveArnold.Examples.School
{
  MainContract University
  {
    Parameters
    {
      [1 to 100] Scalar Integer InstanceBind UniversityCourses;
      Scalar Integer MaxCoursesForFTSStudents = 4;
      Scalar Integer MaxCoursesForPFTSStudents = 2;
      Scalar Integer PassRate = 70;
      [1 to 12] Scalar Integer InstanceBind NumTermsToComplete;
    }

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

    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)
  }
}
```

The screenshot shows the Validation Framework application interface. The main window displays the contract definition code for the `University` contract. The `Parameters` section defines `UniversityCourses`, `MaxCoursesForFTSStudents`, `MaxCoursesForPFTSStudents`, `PassRate`, and `NumTermsToComplete`. The `new()` and `finalize()` methods include `Post` and `Pre` conditions. The `CreateCourse` method is also defined. The `Output` window at the bottom shows a successful build message: "Clean started: Project: New ACL Project1, Configuration: Build VF Runtime. Clean: 1 succeeded, 0 failed, 0 skipped".

## 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;
  }
}
```




Slide - 19

## 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?



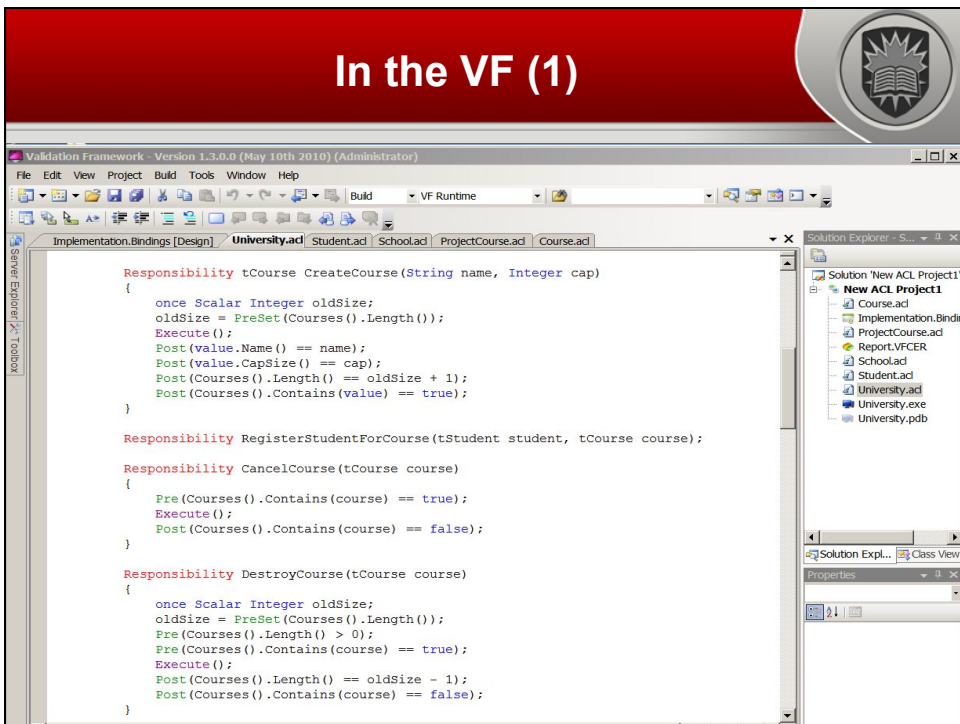
Slide - 20



Carleton UNIVERSITY

Slide - 21

## In the VF (1)



```
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);
}
```

## In the VF (2)

```
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.ad Student.ad School.ad* ProjectCourse.ad Course.ad
Server Explorer Toolbox
Solution Explorer - S...
Solution 'New ACL Project1'
  New ACL Project1
    Course.ad
    Implementation.Bindings
    ProjectCourse.ad
    Report.VFCER
    School.ad
    Student.ad
    University.ad
    University.exe
    University.pdb
Properties
ProjectCourse.ad File Prop...
Advanced
Build Action Compile
Custom Toc
Custom Toc
File Name ProjectCourse
Full Path C:\2-soheila-back
```

```
Import Core;

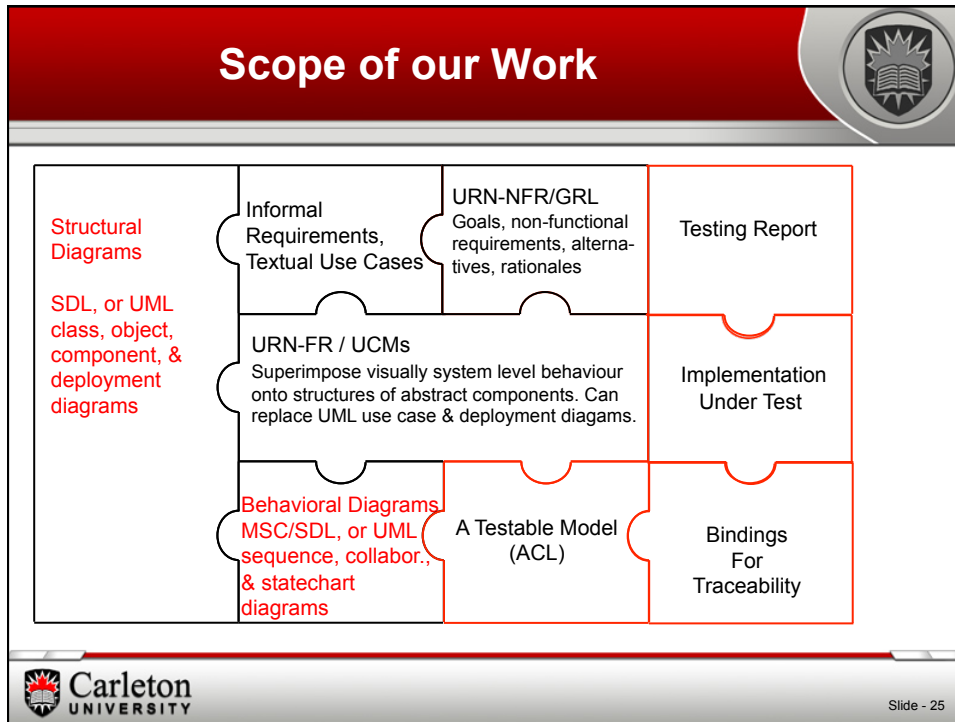
Namespace DaveArnold.Examples.School
{
  Contract ProjectCourse extends Course
  {
    Parameters
    {
      Scalar Boolean InstanceBind HasProject = { default true, false };
    }

    Observability Integer ProjectWeight();
    Observability Boolean HasProject()
    {
      Parameters.HasProject == true;
    }

    refine Observability Integer TotalMarks()
    {
      Scalar Integer markTotal = base.TotalMarks();
      choice (HasProject()) true
      {
        markTotal = markTotal + ProjectWeight();
      }
      value = markTotal;
    }
  }
}
```

## In the VF (3)

```
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.ad Student.ad School.ad* ProjectCourse.ad Course.ad
Server Explorer Toolbox
Solution Explorer - S...
Solution 'New ACL Project1'
  New ACL Project1
    Course.ad
    Implementation.Bindings
    ProjectCourse.ad
    Report.VFCER
    School.ad
    Student.ad
    University.ad
    University.exe
    University.pdb
Properties
Solution Expl... Class View
Properties
Interaction School
Contract University u;
(u.CreateStudents || u.CreateCourses);
Relation Cancelling
Contract University u;
Instance c;
c = u.CreateCourse(dontcare, dontcare),
(u.CancelCourse(c) ?),
observe(TermStarted);
Relation Students
Contract Student s;
Contract University u;
{
  s.RegisterForCourses,
  observe(TermStarted),
  s.TakeCourses,
  observe(CourseComplete) [u.Parameters.UniversityCourses],
  observe(TermEnded),
  observe(MarkRecorded) [u.Parameters.UniversityCourses],
  observe(TermComplete)
} [u.Parameters.NumTermsToComplete];
}
```




## 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

Carleton UNIVERSITY Slide - 27

## 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.

Carleton UNIVERSITY Slide - 28

## 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

Slide - 31

**Validation Framework Contract Evaluation Report**

Depending on the selection made in the left panel, this panel will display different elements of the evaluation report.

**Global Information**

Contract Project:	<b>ACL Project</b>
Implementation Under Test:	<b>HelloWorldIUT.exe</b>
Validation Framework Runtime:	<b>Runtime 1.0 - Version 0.7.0.0 (Nov 2008)</b>
Report Date:	<b>Sunday, November 30, 2008 at 11:41:08 PM</b>

---

Number Of Interactions:	<b>0 Passed, 0 Failed</b>
Number Of Contracts:	<b>1 Passed, 0 Failed</b>
Number Of Static Checks:	<b>0 Passed, 0 Failed</b>
Number Of Dynamic Checks:	<b>0 Passed, 0 Failed</b>

---

**Overall Result**      Pass

Re-Generate Report
*Clicking this button will build and evaluate the current contract project.*



Report Viewer [Report Viewer]\*

Execution Report for SizeCheck  
Displays information regarding an execution of the above invariant.

Global Information

Contract Name: \_\_\_\_\_

Overall Result: **Pass**      Number of Dynamic Checks: **0 Passed, 0 Failed**

Execution Type: **Entry**      Number of Built-in Checks: **2 Passed, 0 Failed**

Check	Result
context.size >= 0	Pass
context.size == Size()	Pass

## 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



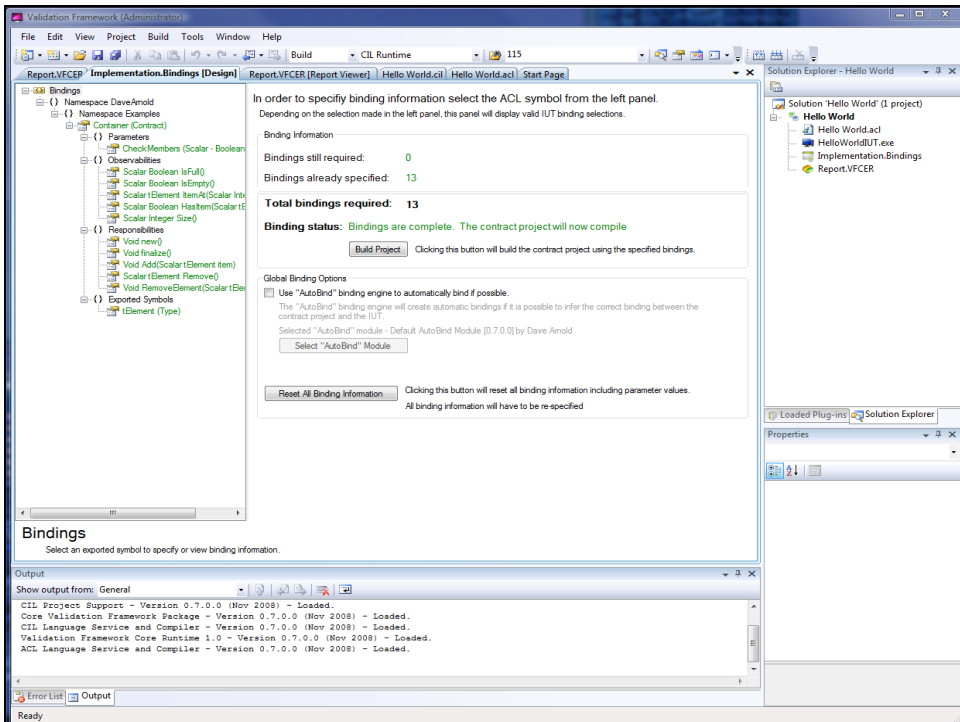
- Our TRM and supporting VF contribute in the areas of requirements engineering and validation by:
- Proposing a new set of requirements for a requirements model that supports operational validation. This set being the first, to the best of our knowledge, to include the following:
  - Capture of functional and non-functional requirements
  - Testability of the requirements model
  - Executability 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



Slide - 37



Validation Framework (Administrator)

Report.VFCER Implementation.Bindings [Design] Report.VFCER [Report Viewer] Hello World.ci Hello World.aci Start Page

In order to specify binding information select the ACL symbol from the left panel. Depending on the selection made in the left panel, this panel will display valid IUT binding selections.

Binding Information

Bindings still required: 0  
Bindings already specified: 13  
**Total bindings required: 13**

**Binding status:** Bindings are complete. The contract project will now compile

Clicking this button will build the contract project using the specified bindings.

Global Binding Options

Use "AutoBind" binding engine to automatically bind if possible.  
The "AutoBind" binding engine will create automatic bindings if it is possible to infer the correct binding between the contract project and the IUT.  
Selected "AutoBind" module - Default AutoBind Module [0.7.0.0] by Dave Arnold

Clicking this button will reset all binding information including parameter values.  
All binding information will have to be re-specified

Bindings

Select an exported symbol to specify or view binding information.

Output

Show output from: General

CIL Project Support - Version 0.7.0.0 (Nov 2008) - Loaded.  
Core Validation Framework Package - Version 0.7.0.0 (Nov 2008) - Loaded.  
CIL Language Service and Compiler - Version 0.7.0.0 (Nov 2008) - Loaded.  
Validation Framework Core Runtime 1.0 - Version 0.7.0.0 (Nov 2008) - Loaded.  
ACL Language Service and Compiler - Version 0.7.0.0 (Nov 2008) - Loaded.

Ready

The screenshot shows the 'Implementation.Bindings [Design]' window. On the left, a tree view shows the project structure under 'Namespace DaveArnold', including 'Container (Contract)' with its 'Observabilities' and 'Responsibilities'. The main area is titled 'Contract Binding - DaveArnold.Examples.Container' and is 'Bound to DaveArnold.VF.Examples.HelloWorld.Container'. The 'Implementation Information' pane shows a tree view of the implementation path: 'C:\Validation Framework\Examples\HelloWorld\Hello World\HelloWorldIUT.exe' > 'DaveArnold' > 'DaveArnold.VF' > 'DaveArnold.VF.Examples' > 'DaveArnold.VF.Examples.HelloWorld' > 'DaveArnold.VF.Examples.HelloWorld.Container'. Below this are 'Binding Options' with a checkbox for 'Show Recommended Bindings Only' and a 'Clear Binding' button.

**Contract - [DaveArnold.Examples].Container**  
Bound to DaveArnold.VF.Examples.HelloWorld.Container

The screenshot shows the 'Implementation.Bindings [Design]' window. On the left, the tree view highlights 'Scalar Boolean IsFull()' under 'Observabilities'. The main area is titled 'Observability Binding - Scalar Boolean IsFull()' and is 'Bound to method get\_IsFull : bool()'. The 'Implementation Information' pane shows a tree view of the implementation path: 'DaveArnold.VF.Examples.HelloWorld.Container' > 'method HasItem : bool(DaveArnold.VF.Examples.HelloWorld.ContainerItem)' > 'prop IsEmpty : instance bool()' > 'prop IsFull : instance bool()'. Below this are 'Binding Options' with a checkbox for 'Show Recommended Bindings Only' and a 'Clear Binding' button.

**Observability - Scalar Boolean IsFull()**  
Bound to method get\_IsFull : bool()

The screenshot shows the Visual Studio IDE with the 'Implementation.Bindings [Design]' window. On the left, a tree view shows the project structure under 'Namespace DaveArnold', including 'Container (Contract)' and 'tElement (Type)'. The 'Responsibilities' section for 'tElement (Type)' is expanded, showing 'Void Add(Scalar tElement item)' selected. The main pane displays 'Responsibility Binding - Void Add(Scalar tElement item)' with the text 'Bound to method Add : void(DaveArnold.VF.Examples.HelloWorld.ContainerItem)'. Below this, the 'Implementation Information' pane lists methods from 'DaveArnold.VF.Examples.HelloWorld.Container', including 'method Add : void(DaveArnold.VF.Examples.HelloWorld.ContainerItem)'. At the bottom, a caption reads 'Responsibility - Void Add(Scalar tElement item)' with a sub-caption 'Bound to method Add : void(DaveArnold.VF.Examples.HelloWorld.ContainerItem)'.

The screenshot shows the Visual Studio IDE with the 'Implementation.Bindings [Design]' window. On the left, the tree view is similar to the first screenshot, but 'tElement (Type)' is selected under 'Exported Symbols'. The main pane displays 'Type Binding - DaveArnold.Examples.Container::tElement' with the text 'Bound to DaveArnold.VF.Examples.HelloWorld.ContainerItem'. The 'Implementation Information' pane shows a tree view of the project structure, with 'DaveArnold.VF.Examples.HelloWorld.ContainerItem' selected. At the bottom, a caption reads 'Type - [DaveArnold.Examples].Container::tElement' with a sub-caption '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...