

Agile Specifications

Linking Requirements and Architecture Using Executable Examples

Mario Cardinal

Software Architect

<http://mariocardinal.com>



Who am I ?

- Software architect who works with the **Urban Turtle** team



Professional Scrum Developer (.NET)

- An innovative program for developers from Microsoft and the founders of Scrum



Learn how to use modern engineering practices to develop an increment of complete, potentially shippable functionality using Visual Studio 2010, ALM, and the Scrum framework

- Training course and certification available by Pyxis
<http://pyxis-tech.com/psd>


- Free **Urban Turtle** for students
<http://urbanturtle.com>



Extend Microsoft TFS with scrum project management

Why are we here?

I have a dream

- Communicating input specifications accurately with minimal effort
 - Verifying, repeatedly, and at any time that the software application under construction meets the specifications
 - Automating the verification using input specifications
- 

Agenda

Simplicity is the ultimate sophistication

- Requirements specifications (input)
 - Expressing goals with user stories
 - Specifying user stories using examples
 - Connecting examples with code
 - Verifying code correctness with examples
- Architecture specifications (output)
 - Partitioning complexity into responsibilities
 - Isolating responsibilities behind visible interface
 - Designing visible interface
- Linking Requirements and Architecture
 - Automating the verification using input specifications


Requirements specifications

Step1. Tackle complexity

- Describe goals using user stories
- Discover goals by exploring user roles
 - “As a <**user role**>, I want <**goal**> so that <**benefits**>”
- Plan and prioritize goals using a backlog
 - Derive scope from goals
 - Remove dependencies from goals
- Manage and track works using goals

Specify user stories using examples

Step2. Illustrate goals

- Illustrate user stories using concrete examples
 - Specify only when the user story is committed
 - Specify collaboratively using examples
 - Refine examples using visual aids
 - Evolves examples from release to release
- 

Connect examples with code

Step 3. Automate verification

- Automate using glue code
 - Fixture code (FIT)
 - UI automation (Selenium)
 - Output analysis (TextTest)
- Automate using an internal DSL
 - Scenario + Step definition (Cucumber)
 - Examples are written using GWT syntax
 - Potential impedance mismatch between examples and code
 - Table mapping + Architecture specification
 - Examples are written in plain English
 - Ubiquitous language between examples and code

Verify code correctness with examples

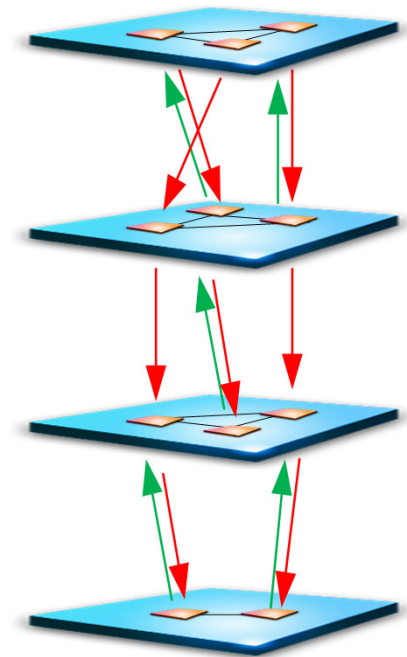
Step4. Ensure implementation correctness

- Execute code on a test environment
 - Runner
 - Remote execution
- Enhance examples with test results
 - **Pass**, **Fail** or **Not implemented**
- Create a living documentation

Architecture specifications

Partitioning complexity into responsibilities

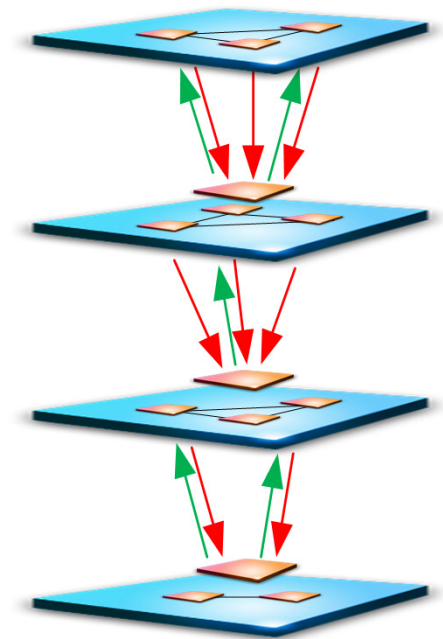
- Tackle complexity by achieving simplicity
- Achieve simplicity using proven heuristics
- Partition complexity into layers
- Divide layers into responsibilities



Architecture specifications

Isolating responsibilities behind visible interface

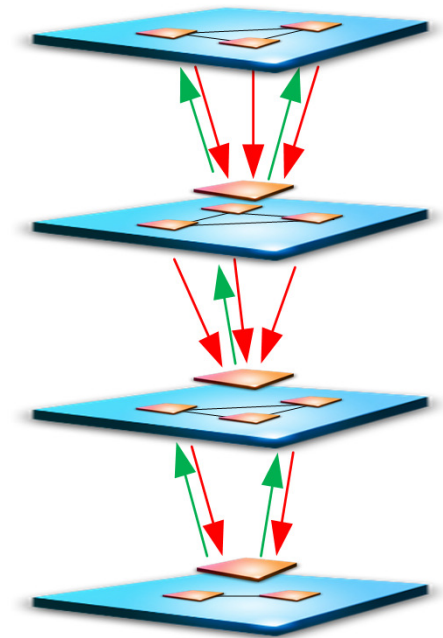
- Consist in breaking the dependency chain using a visible interface
- Visible interface is a façade
 - Act as a single point of coupling between layers
 - Prevent modifications to propagate to other layers
- Visible interface factor out details regarding each layer
 - Help understand the hidden behavior encapsulate inside a layer



Architecture specifications

Isolating responsibilities behind visible interface

- Allow substitution of implementation for various testing purpose
 - Replacing an implementation with a mockup without affecting dependent clients
 - Mockup is a prototype that looks and works just like the real thing
- Allow testing a layer in a test-bed without having to assemble the whole system



Architecture specifications

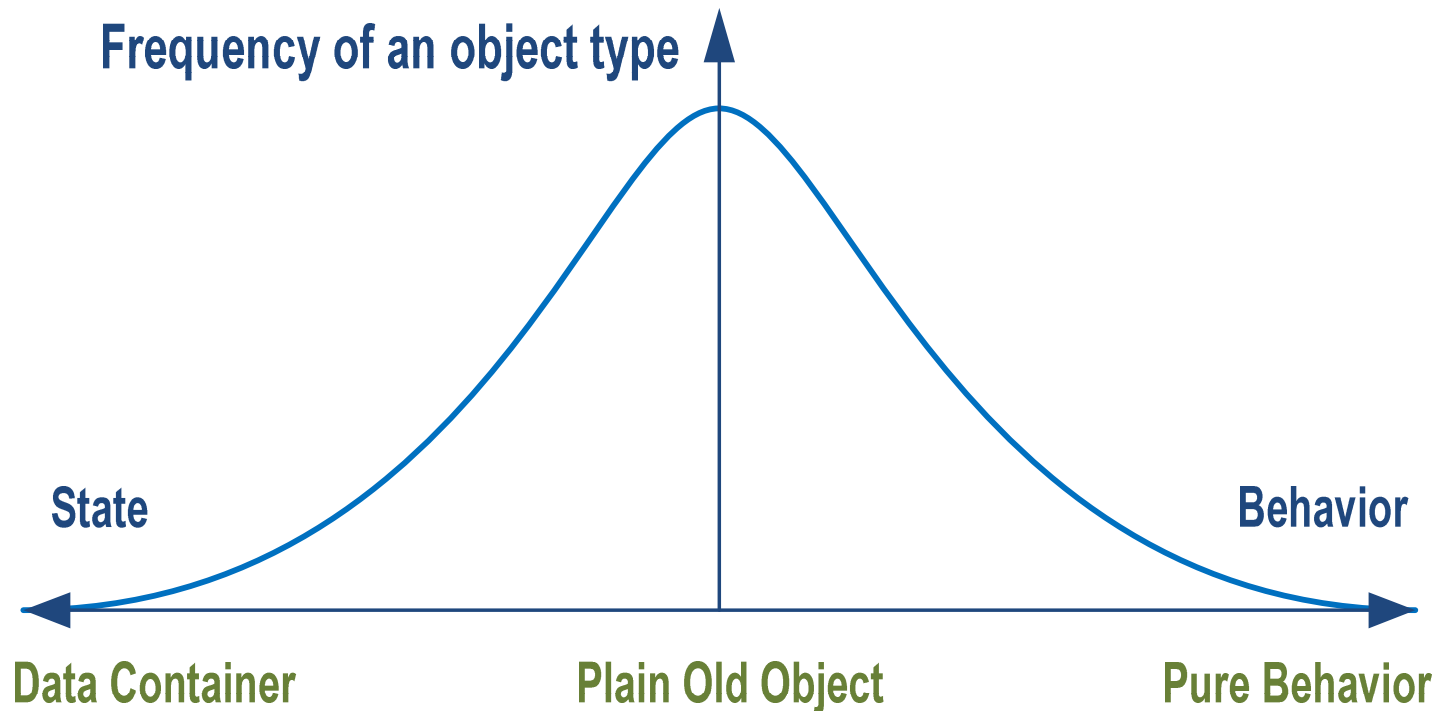
Designing visible interface

- Objects programming language provides formal abstractions not only for designing the syntax but also for designing the contract and semantics of the visible interface

interface = syntax + contract + semantic

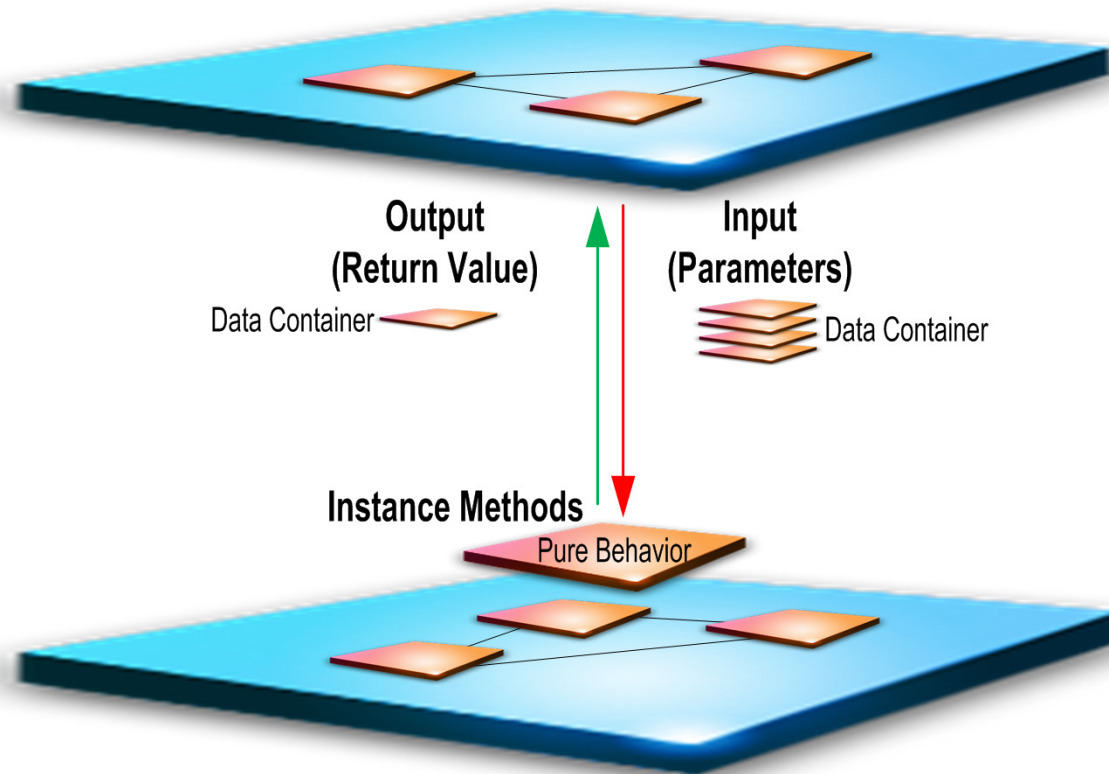
Architecture specifications

Visible interface syntax



Architecture specifications

Visible interface syntax



Architecture specifications

Visible interface syntax

```
public interface IPureBehavior
{
    DataContainer1 Operation1(Guid identifier);
}

public class DataContainer1
{
    public DataContainer1(Guid input1, DateTime input2, string input3)
    {
        IdentifierAttribute = input1;
        DateAttribute = input2;
        StringAttribute = input3;
    }
    public Guid IdentifierAttribute { get; private set; }
    public DateTime DateAttribute { get; private set; }
    public string StringAttribute { get; private set; }
}

public class LocateKeyException : Exception
{
    public LocateKeyException(Guid _identifier)
        : base("Unable to find item with key value " + _identifier)
    {
    }
}
```


Architecture specifications

Visible interface contract

- This category of semantics (before and after execution) has been popularized under the name “Design by Contract” by Bertrand Meyer in his seminal book Object Oriented Software Construction
- Collaboration before and after execution must be managed by a set of clauses, which form a contract:
 - ❑ **Preconditions** are conditions that must be met on entry to the method before execution
 - ❑ **Postconditions** are conditions that must be true after execution at all normal exit points of the method
 - ❑ **Class invariants** are conditions that the object instance must satisfy at any time

Architecture specifications

Visible interface semantic

- Illustrate responsibilities using examples
 - Add tables mapping in architecture specification
 - Assign a class and a method to the table
 - Assign input parameters and return value by adding columns
 - Define object type using columns header
 - Define composite object by splitting columns
 - List the case by adding rows
 - Create a mockup
 - Write minimal implementation to execute examples with success

Linking Requirements and Architecture

Verify requirements with the domain layer

- Divide domain layer into responsibilities
- Illustrate responsibilities using examples from requirements
 - Specify the responsibility of the visible interface
 - Specify only when the responsibility is committed
 - Specify collaboratively using examples
- Verify and refine responsibilities with examples
 - Table mapping from requirement specifications
 - Mockup code
- Evolves examples from release to release

Conclusion

- Is this only theory?
- What are the tool available?
 - Green Pepper (Java, .NET, C++)
 - Cucumber (Ruby)



Build the right
software!

<http://www.greenpeppersoftware.com>

Q & A

Do not hesitate to contact me
mcardinal@mariocardinal.com

