A Model Driven Standards Process

Art Griesser, Ph.D.

griesser@nist.gov

Electronics Engineer

NIST Semiconductor Electronics Division

Outline

1. Business case
   – Is it worth doing?
2. Requirements Model
   – What problem are we trying to solve?
3. Architectural Model
   – Highest level of design
   – Determines basic approach

Preparation

4. Design Model
   – The solution. The Standard.
5. Test Model
   – Meets requirements?
   – Conforms to design?
6. Prototype / Proof-of-concept
   – Prove it works, get buy-in

Standard is built here

Business Case

Scope List Actor Catalog Analysis Model Architectual Model

Use Case Catalog

done

iterate again

iteration N iteration N-1 iteration N-2

Detailed Use Cases Scope List Test Model Design Model Prototype

3
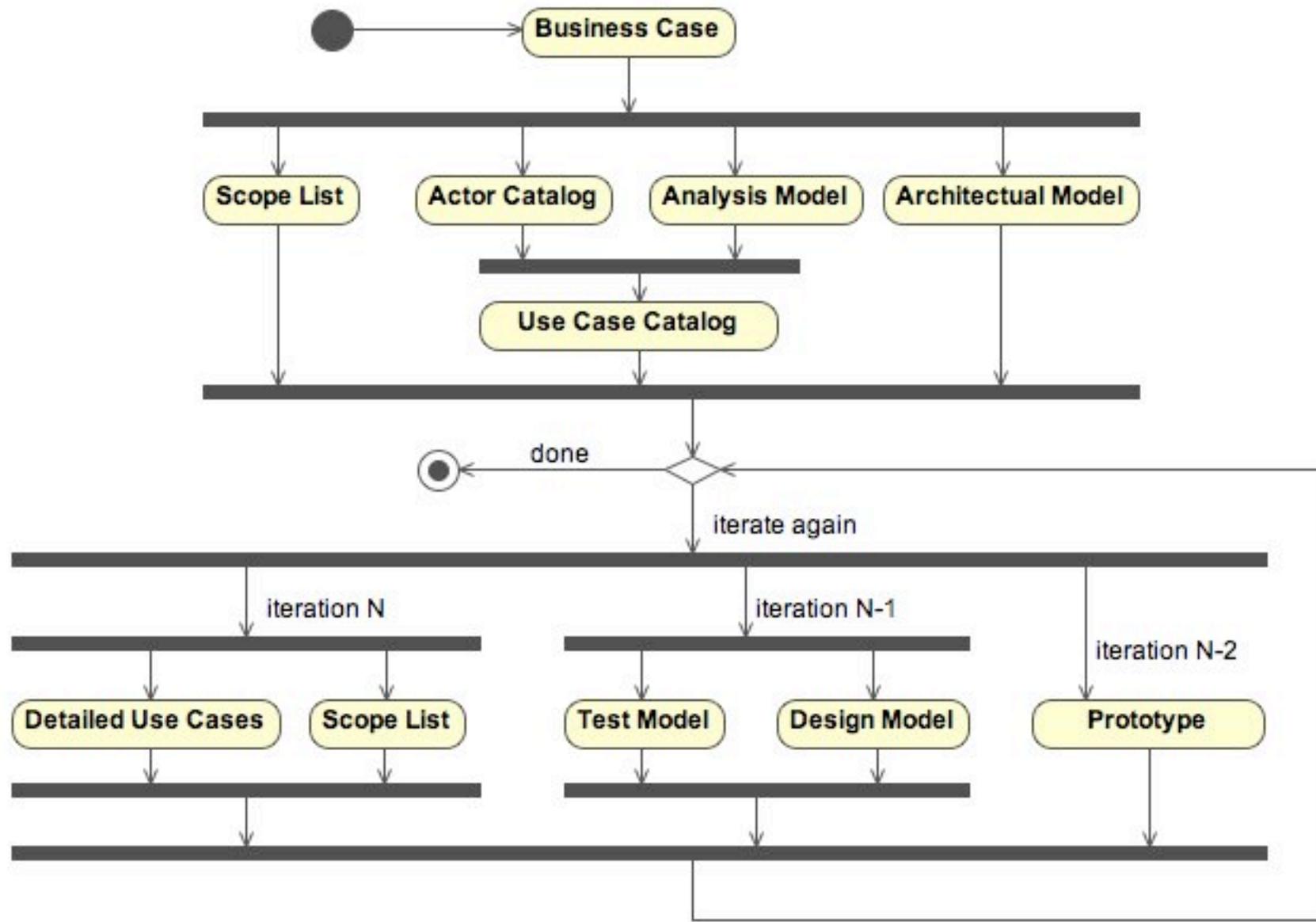
Why?

- **Not enough time already… how can we add:**
  - Business case
  - Requirements Model
  - Architectural Model
- **Because up-front preparation:**
  - Preemptively settles scope squabbles
  - Defines the boundaries of the solution
  - Reduces thrashing / rework later
- **Because tests + prototype prove design is valid**
- **Because it's cheaper in the long run**

What is a model?

- **A simplified representation**
  - Like a scale model of a building
  - Helps everyone imagine finished product
- **An abstraction**
  - Boiled down to some essential aspect
  - Clarifies that aspect
  - Helps participants think about that aspect
  - Helps explain that aspect
  - Helps orient new participants
  - Constrains later stages of development

1. Business Case

- **Sketches problem to be solved**
- **Estimates cost, risk, and benefits**
- **Management uses it:**
  - Go / NoGo decision
  - Get commitments from participants
  - Resource allocation
    - Domain experts
    - Prototype developers
    - Compliance test developers
- **After requirements capture, revisit**

Business Case Artifacts

- **Vision statement**
  - High level description of problems to be solved
- **Cost benefit analysis**
  - Ballpark benefits
    - Solution, scope not certain yet
  - Ballpark cost
    - Requirements not certain yet
    - Solution not certain yet
  - Cost estimation tools:
    - Comparison with finished standards
    - Function Point Analysis of prototype
    - **Co**nstructive **Co**st **Mo**del II
    - Putnam model
- **Risk Assessment**

2. Requirements Model

- **Understand the problem**
  - Business Case rarely clear enough
  - Who are stakeholders?
  - How do they benefit?
  - What's involved in solving the problem?
- **Stakeholder - developer contract**
  - Nails down scope
  - Nails down expected benefits
- **Input to Architecture Model and Design Model**

Requirements Model Artifacts

- **Scope in/out list**
- **Use cases** (next slide)
  - Specify how "actors" use implementation
- **Use case & actor catalog**
  - Organizes, categorizes use cases & actors
- **UML Use case diagrams**
  - Show actor - use case relationships
- **Analysis model**
  - Supports use cases
  - Provides background

What is a Use Case?

- **Describes how stakeholders derive value**
  - Contract between stakeholders & developers
- **Describes usage scenarios:**
  - Preconditions
  - Triggers
  - What happens (next slide)
  - Postconditions
- **Written in the vocabulary of the user**
  - Avoids implementation details
- **Includes**
  - "Business rules"
  - Issues and their resolution

Specifying "What Happens" I

- **UML Sequence diagram**
  - Emphasize: actor - system interaction
  - Shows interactions as function of time
  - Structure is not present
- **UML Collaboration diagram**
  - Emphasize: actor - system interaction
  - Shows interactions in context of structure
  - Sequence is present, but hard to follow

Specifying "What Happens" II

- **UML Statechart diagram**
  - Emphasize: internal states
  - Describes behavior resulting from internal states
  - Shows how internal states respond to stimuli
- **UML Activity diagram**
  - Emphasize: things that get done
  - Shows sequence of activities
  - Allows for parallel activities
  - A special form of state diagram, useful when:
    - States have activities
    - Automatically exit state when activity is finished

Many Kinds[1] of Use Cases

- **Scope:**
  - Business[2]   Probably not useful for standards
  - System[2]   How implementation is used
  - Component   Used by other components
- **Goal-Level:**
  - Summary   Organizes User-Goals
  - User-Goal   Why Actor uses system
  - Subfunction   Subgoal

                     Necessary but not interesting by itself

1. *Writing Effective Use Cases*, Alistair Cockburn
2. Cockburn distinguishes black box and white box versions

Analysis Model

- **Part of Requirements Model**
- **Analysis of use cases**
- **Describes domain "things"**
  - Classifies
  - Shows structure
  - Describes relationships
  - Describes behaviors

Analysis Model Artifacts

- **Domain Structure**
  - UML Class diagrams
    - Abstract: considers all instances together as a class
    - Enumerates (but does not describe) behaviors
  - UML Object diagrams
    - Concrete: shows individual instances
    - Rarely necessary
    - Supplements, explains class diagram
    - Can be mixed into class diagrams
- **Domain Behavior**
  - Our old friends from "Specifying what happens"
    - UML Sequence diagrams
    - UML Collaboration diagrams
    - UML Statechart diagrams
    - UML Activity diagrams

3. Architectural Model

- **Selects physical & logical components meeting:**
  - Functional requirements
  - Non-functional requirements
    - Performance, security, reliability
  - Reuse goals, fit to other standards
- **Specifies**
  - Protocols
    - Communication
    - Data access
  - Component dependencies
  - Component logical-to-physical mapping
  - Facade / Interface behaviors

Supports
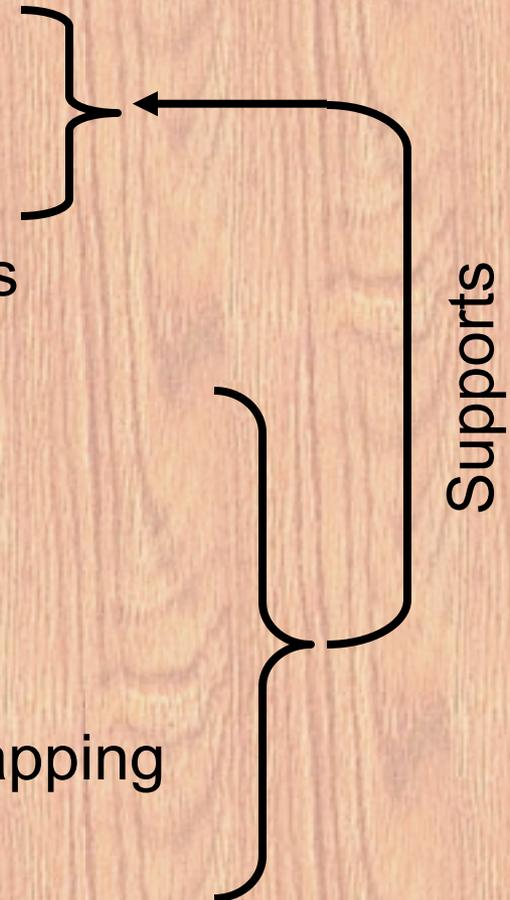
Architectural Model Artifacts

- **UML Component Diagram**
  - Dependencies among components
  - Composition of components
- **UML Deployment Diagram**
  - Allocates logical components to physical components
- **Component-Scoped use cases**
- **Alternatives considered**
  - Why rejected

4. Design Model

- **Describes the solution**
- **Everything should be traceable to use cases**
- **Contains information from Analysis Model**
  - Sometimes a direct copy
  - Sometimes almost unrecognizable
  - Does not contain peripheral domain objects
- **Contains extras (unknown to domain experts)**
  - Abstractions
  - Factorizations   Perhaps meta-info parameterized
  - Patterns[1]

1. *Design Patterns*, by Erich Gamma, et al

Design Model Artifacts

- **The same diagram types as Analysis Model**
  - Structure
    - UML Class diagrams
    - UML Object diagrams
  - Behavior
    - UML Sequence diagrams
    - UML Collaboration diagrams
    - UML Statechart diagrams
    - UML Activity diagrams
- **The content is different**
  - We are describing the solution instead of the domain

Caution - Standards Specific

- **Need to distinguish**
  - General part of design
    - Applicable to all implementations
    - Must be tested against all implementations
      - Ensures implementation interoperability
  - Parts of design specific to prototype
    - Helps build the prototype
    - Must be tested only against the prototype

- **Diagrams could distinguish by stereotype**
  - Stereotypes could be color-coded
  - Color coding could conflict with other classifications[1]

1. *Java Modeling In Color With UML: Enterprise Components and Process*, Peter Coad, et al

5. Test Model

- **Tests for conformance to standard**
  - Applicable to all implementations
  - Conformance to general part of design
    - Ensures interoperability with other implementations
  - Conformance to use cases
    - Ensures value delivered to stakeholders
    - Ensures interoperability with other standards
- **May define additional tests for prototype**
  - Often "white-box" tests
  - Embarrassing if vendors test against buggy prototype

Test Artifacts

- **Test case**
  - Traceable to one use case, or to design
  - One use case can result in many test cases
  - Required: executable code
  - Optional: human readable description
  - Specifies:
    - Initial condition of system
      - Load (on system under test, and/or infrastructure)
    - Event or stimulus
    - Response
    - Timing of response
    - Final condition of system
- **Separate**
  - Prototype specific & general tests
  - Integration tests & component-level regression tests

6. Prototype

- **Mitigates risk**
- **Ensures the standard will:**
  - Be self consistent
  - Operate correctly
  - Perform
  - Interoperate with other standards
- **Reduces extraneous content of standard**
- **Resolves ambiguities**
- **Reference for implementation interoperability tests**
- **Key testbed component**
- **Keeps participants focused**
- **Promotes buy-in**
- **Jump starts vendor implementations**

UML References

- Martin Fowler. *UML Distilled: A Brief Guide to the Standard Object Modeling Language, Third Edition*. Boston: Addison Wesley (2003)

- Official specifications: http://www.uml.org/

- J. Rumbaugh, I. Jacobson, G. Booch. *The Unified Modeling Language Reference Manual, Second Edition*. Boston: Addison Wesley (2004)

UML References

Authoritative at one time… dated, but useful

- J. Rumbaugh et al. *Object-Oriented Modeling and Design*. Prentice-Hall (1990)
  - Describes OMT, a UML predecessor
- G. Booch, J. Rumbaugh, I. Jacobson. *The Unified Modeling Language User Guide.* Boston: Addison Wesley (1998)
- I. Jacobson, G. Booch, J. Rumbaugh. *The Unified Software Development Process.* Boston: Addison Wesley (1999)
- J. Warmer, A. Kleppe. *The Object Constraint Language: Getting Your Models Ready for MDA, Second Edition.* Boston: Addison Wesley (2003)