

A Model Driven Standards Process

Art Griesser, Ph.D.

griesser@nist.gov

Electronics Engineer

NIST Semiconductor Electronics Division

Cloud background footage Copyright www.freestockfootage.com - All Rights Reserved

Wood background image Copyright © 2002-2003, KTN 3D - All Rights Reserved

Official contribution of the National Institute of Standards and Technology; not subject to copyright in the United States

Certain commercial equipment, instruments, or materials are identified in this paper to foster understanding. Such identification does not imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the materials or equipment identified are necessarily the best available for the purpose.

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
 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
-
- Preparation
- Standard is built here

2

The value of UML diagrams is under-appreciated by standards participants: this talk will describe these UML diagrams, and show how they fit into a robust formal standardization process based on the best practices of software development.

Items in blue have UML diagrams of one sort or another.

The numbers show the ordering for an ideal first iteration. In practice some activities may overlap, and some may be revisited. For example, once the requirements model is finished, you have a better idea of the costs, so the business case should be checked again. In some cases you may be able to skip much of a step: for example it may be possible to borrow much of an architecture from another standard. You might also split the standard into sections that are developed sequentially. In this case, the high level view of the requirements model would be completed along with the architecture. Then, for each section, the remainder of step 2 and steps 4-6 would be performed. If the sections are small, you are using eXtreme Programming.

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**

Let's have a quick poll. Please raise your hand if you have ever participated in a standard that did *not* have either rework or problem/solution scope problems.

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

4

Example: “essential aspects” for building might be air-handling, electrical wiring, appearance, and thermal considerations.

Our analogous models are Requirements, Architectural, Design, and Test (some add Robustness). Many experts also talk about “Implementation models”. Sometimes they refer to artifacts that can fit into other models. Sometimes they refer to artifacts that accompany an implementation. I think the implementation is what you are modeling, not just another model.

SEMI Standards Discovery Maps

[Maps](#)

[Overview](#)

[Viewing
Requirements](#)

[Summary
Pages](#)

[SEMI
Links](#)

[International
SEMATEC
Links](#)

[Other
Useful
Links](#)

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**

6

Management = Standards body management, participant management

Without participation commitments from vendors+consumers, why bother? “Build it and they will come” is not working.

Note: a “business case” is not the same thing as a “business-scoped use case”

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
 - **Constructive Cost Model II**
 - Putnam model
- **Risk Assessment**

7

The uncertainties are large, but what else can management base its decisions on? Scrying bird entrails?

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

Actors may be people or machines

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

10

Business rule: constraint required by policy or regulation.

The Preconditions, Postconditions, and Business rules are usually less precise than they could be, because they must be in a form understandable to domain experts. Developers should probably recast these into **Object Constraint Language** or something equivalent.

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

Collectively, these are called Interaction Diagrams

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 Kinds¹ of Use Cases

- **Scope:**
 - Business² Probably not useful for standards
 - System² 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

13

Scope and Goal-level are independent dimensions. They form a “Cartesian product” of nine use cases types. Cockburn actually includes more values for each dimension: for example he distinguishes between black box and white box versions of the Business and System scope. I prefer to have only black box use cases... except for Component-scope use cases, which are inherently white box.

Component use cases are arguably architectural artifacts, rather than requirements artifacts.

I think the most important use cases are System Goal-level (in blue).

“Component” scope use cases are arguably part of the Architecture Model, rather than Requirements Model

Note: a “business-scoped use case” is not the same thing as a “business case”

Analysis Model

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

Can you really hope to solve or even understand a problem without this?

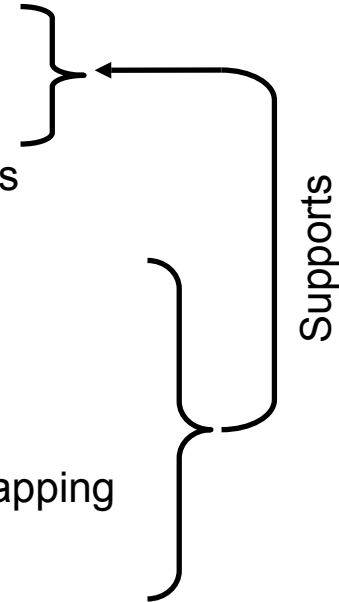
Analysis Model Artifacts

- **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

Collectively, these describe the domain.

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



Ideally based on the Requirements model, but sometimes based on business case.

Constrains the design to the solution space envisioned by the architect: input to design model.

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
 - Patterns¹
- } Perhaps meta-info parameterized

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

We have discussed these artifacts already in other contexts: they are listed again as reminder.

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 classifications¹

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

We have discussed these artifacts already in other contexts: they are listed again as reminder.

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

When people think about testing implementations of standards, they usually stop at interoperability: they don't think to include conformance to use cases (which are often missing from standards anyway).

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

The integration tests are separate because they are meaningless if the component-level tests fail.

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**

By “Perform” I mean that implementations will be fast enough to be usable.

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)

I have not yet looked at the second edition of the reference manual.

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)

Since “*The Unified Modeling Language Reference Manual*” has been updated, perhaps the last three will be updated as well.