An Aspect-Oriented Strategy for Evaluating Software Architectures that Evolve

2 March 2005

Phillip Schmidt
Phillip.P.Schmidt@aero.org
Agenda

- Motivation
- Some current strategies
- New aspect-oriented (AO) strategy
- Comparison with scenario-based approaches
- Experiences applying the strategy
Architectural Value

- Your architectural value will be reflected in how you perceive architecture:
  - Strategic tool?
  - Something in the minds of competent people?
  - General preliminary design only, to be discarded after coding
  - Merely drawings or documentation?
  - An accurate, evolving reflection of the design?
  - Something analyzable
  - Something executable

- Perceptions have consequences and risks
- Choose wisely
**Architectural Needs**

- Do your architectural values address your problems?
- Do you know you *have* problems?
- Is there an architectural value gap between you and your architectural provider?

- Architectural values are stressed when:
  - Managing complex, evolving architectural relationships
  - Architectural modeling practices are unclear
  - Unexpected concerns arise
Architectural Artifact Relationships

Use Cases

Unified Modeling Language (UML)

Class Diagrams

Sequence Diagrams

State Diagrams

Code

SW ICD

Test Procedure

Unit Test

Requirement Specification

Course of action

COA step

COA step

Architectural Artifacts

Architecture artifacts are life cycle concerns
Complexity of Requirement Evolution

Course of action dependency

Course of action steps

16 months later

Problematic references
Architectural Modeling Practices
(What we think we want)

System Engineering Activities
- Define assumptions
- Identify simplifications, tradeoffs
- Identify limitations
- Define system constraints
- Identify preferences
- Identify, analyze and manage requirements, specifications

Software Engineering Activities
- Evaluate design tradeoffs
- Define concrete classes, interactions, relationships, states, activities
- Identify infrastructure
- Develop prototypes, deployment views
- Identify product constraints, timing
- Ensure traceability to requirements
- Integrate and Test

Architectural Design (system in context)
- (styles, views, patterns, relationships)
- Design Engineering
- (classes, data structures)
- Component Layering

Analysis Model
- What
- System Modeling (abstracted world view use cases)
  - Informational models (data flow)
  - Functional models (activity flow)
  - Behavioral models (events, sequences)
  - Abstract
  - Problem domain oriented

Design Model
- How
- Coherent, thorough, well planned representations
- Traceable to requirements,
- Completeness, correctness
- Concrete realizations
- Solution domain oriented
Architectural Modeling Practices
(What tends to happen)

System Engineering Activities
- Define assumptions
- Identify simplifications
- Identify limitations
- Define constraints
- Identify preferences
- Identify, analyze and manage requirements, specifications

Software Engineering Activities
- Evaluate design tradeoffs
- Define concrete classes, interactions, relationships, states, activities
- Identify infrastructure
- Develop prototypes
- Identify preferences
- Ensure traceability to requirements
- Integrate and Test

Analysis Model
Architectural Design (system in context)
(styles, views, patterns, relationships)
Design Engineering
(classes, data structures)
Component Layering

Design Model

What
How
Architectural Modeling Practices
(Blurring of models)
Architectural Modeling Practices
(What we usually get)

Architecture Model

Huh?
Level of Architectural Detail

- Architectural models tend to be a mix of conceptual and implementation based information

![Diagram showing the level of architectural detail for different projects]

Impact Assessment of Crosscutting Concerns

Course of action
COA step
COA step

Unified Modeling Language (UML)
Class Diagrams
Sequence Diagrams
State Diagrams

Test Procedure
Code

Concern

ECP

Point of interest

Good Intentions

- Even the “best” of architectural values can be overwhelmed by the complexity of our problems
Software Architecture Realities

- We don’t know everything
  - What we know may be correct, but irrelevant
  - What we do know, may not be represented clearly
  - Scalability of manual techniques
- We can’t predict everything
  - Things change unexpectedly
  - Unplanned feature interactions
- We tend to ignore the hard stuff
  - Non-functional requirements
  - Conflicting operational concepts/goals
  - Domain-specific details within commercial tools
  - Subtle software-hardware real-time dependencies

Space architectures are handicapped by their complexity
Strategies for Managing Handicapped Architectures

- **Ignore It:**
  - It’s the contractor’s problem.

- **Contract It**
  - Just put the required level of detail on contract

- **Enforce It**
  - Have contractor develop the appropriate level of architectural granularity

- **Evangelize It**
  - Motivate best practices through conferences, tutorials

- **Analyze It**
  - Stakeholder collaboration for effective automated analysis
Strategic Advice

- Develop **analyzable architectures**

- An **aspect-oriented strategy can help** with assessing evolving, handicapped architectures that might not be natively analyzable.

- Reduce architectural value gaps through **aspect-oriented augmentation**
Aspect-Oriented Assessment

Periodic Deliveries of Evolving Artifacts

UML Model
Use Cases
Reqts Allocations
Sizing, Timing, Constraints
Test Procedures
Code
Other

As-designed UML (multi-viewpoint Metamodel)
Use case info
Allocation info
Constraint info
Test Procedure info

As-built UML Model

Augmentation Support
Re-augmentation Aspects
Concern Management/Extraction Aspects

XML
Profiles

COTS translation
Project-specific and reusable translations

Architectural handles (native)
Architectural handles (augmented)


GSAW 2005 18
## Comparison of Scenario-based and Aspect-Oriented SW Assessments

<table>
<thead>
<tr>
<th>Scenario-based Strategy</th>
<th>Aspect Oriented Strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Quality attributes</strong> prior to development interpreted in context of pre-planned provided scenarios. Feature management in terms of planned change.</td>
<td>Planned or unplanned concerns described as aspects of interest over an architecture. Crosscutting concerns support unplanned changes</td>
</tr>
<tr>
<td><strong>Predictive</strong> strategy. E.g. quality attributes depend on some pre-defined scenario context.</td>
<td><strong>Corrective</strong> strategy. Some quality attributes do not have a scenario context that can be predicted. Concern evaluation assumes an architectural representation.</td>
</tr>
<tr>
<td><strong>Questioning technique</strong> with reliance on human consultation for completeness of scenario interactions; qualitative metric formulation. Good for conceptual overview of whole system</td>
<td>Managing complex interactions require measurement and evaluation. <strong>Quantitative</strong> emphasis on completeness. Good for investigating issues raised from questioning techniques</td>
</tr>
<tr>
<td>Use of taxonomic hierarchies to define quality attributes</td>
<td>Uses <strong>architectural profiles</strong> to support augmentation, derivation, monitoring, and evaluation of architectural aspects of interest.</td>
</tr>
<tr>
<td>Phased application</td>
<td><strong>Periodic</strong> application during life cycle</td>
</tr>
</tbody>
</table>
Applying an Aspect-Oriented Strategy

- Periodic assessment more costly
- Concern management more general than feature management
  - Not everything is preplanned

- Aspect mining helpful in investigating potential problems
  - Multiple inter-related representation spaces
  - A code-centric viewpoint was inadequate for addressing unexpected change impacts, subsystem test dependencies, schedule/resource dependencies

- UML2 profiles/XML schemas as a framework for architectural augmentation
  - Requirements evolution
  - Constraint enforcement
  - Parameterizing real-time embedded systems for analysis
References

- **Aspect-Oriented Architectural Assessment**

- **Related UML Approaches**

- **Scenario-Based Approaches**
Backup Charts
REACT Implementation Elements

- **Eclipse**
  - **UML 2.0 Model**
    - **UML 2.0 Reflexive Editor**
  - **UML Profiles**
    - **RTPS Profile**
      - OMG-based: Schedulability, performance, timing, resources, etc.
    - **REACT Profile**
      - Environment, Dynamic Assessment, Requirements, Other Augmentation
    - **Project-Specific Profiles**
      - Domain-Specific
  - **Model Generator**
  - **REACT Profile Augmentation Aspects**
  - **Sequence-State Diagram Support Aspects**
  - **Model Reader/Analyst**

- **Analysis Results**

- **REACT Plug-ins**
  - **Model Extractor**
  - **Derivation/Analysis Aspects**
  - **Project-Specific Aspects**
  - **REACT Profile Augmentation Aspects**
  - **Project-Specific Profile Augmentation Aspects**
  - **Sequence Diagram Support Aspects**
  - **Model Generator**
  - **Requirement Visualization**

- **UML 1.x Models**
  - **Config Files**
  - **Model Execution and Dynamic Assessment Tools**

- **REACT Representation**
  - **Sequence Diagram**
  - **State Diagrams**
  - **Other Schemas**

- **Contractor-provided**
- **Eclipse Environment**
- **Eclipse Support**
- **REACT Data**
- **REACT-Implementation**
- **REACT-generated**