Requirements to models: goals and methods

The domain of architecting

The “what”
- Architecture Qualities
- Architecture Representation

The “why”
- System Features
- S/W Requirements

The “how”
- Technology

The “who”
- Architect
- Stakeholders
- Skills
- Organization
- Process

- Defines role
- Defines
- Follows
- Constrains
- Satisfies
- Produces

Enterprise Architectures
Architectural view

• An architectural view is a simplified description (an abstraction) of a system from a particular perspective or vantage point, covering particular concerns, and omitting entities that are not relevant to this perspective
Representing System Architecture

- **Logical View**
  - End-user Functionality

- **Implementation View**
  - Programmers
  - Software management

- **Process View**
  - System integrators
  - Performance
  - Scalability
  - Throughput

- **Deployment View**
  - System engineering
    - System topology
    - Delivery, installation
    - Communication

Conceptual

Physical

Georgia State University

Professor Truex

Enterprise Architectures
Architecturally significant elements

• Not all design is architecture
• Main “business” classes
• Important mechanisms
• Processors and processes
• Layers and subsystems
• Architectural views = slices through models
Reconciling Software Requirement & Architectures

• Grünbacher et al provide insight into the challenge and guidance into solutions
  – The problem is…?
  – The challenge is…?
  – High level solution…?
    • A mid level mapping

• Borrowing from the literature it assembles a composite approach and method to achieve six identified goals of a good ‘requirements ==> Architecture’ mapping method

• Makes an intermediate model that “looks’ like requirements and “sounds” like an architecture
Requirement vs Architecture

- **Requirement** *(IEEE -610-1990)*
  - “condition or capability needed by a user to solve a problem or achieve an objective”
    - Describe aspect of the problem to be solved and constraints on the solution

- **Architectures**
  - High-level abstraction or Model of a solution to a problem
    - High-level abstractions representing structure, behavior and key properties of the system
    - Components; interactions between them as *Connectors* or *Busses* and the properties needed
The “Twin Peaks” conundrum

Level of Detail

Technology Dependence

Low

High

Architecture(s): Enterprise, IT, Infrastructure & Software

Descriptions of the ‘Real’ Work System
Garlan says…

Figure 1: Software Architecture as a ‘Bridge’
But how do we get from …

Figure 1: Software Architecture as a ‘Bridge’
The “Twin Peaks” conundrum

- Level of Detail
  - The ‘real’ work system
  - A large gap
  - Requirements for that ‘real’ system

- Technology Dependence
  - A larger gap
  - Architecture(s) to realize the ‘real’ system in code
The “Twin Peaks” conundrum

- The ‘Real’ work system
- Work System Models and taxonomy
- Requirements for the ‘real’ system
- Mid-range Architectural models
- Architecture(s) to realize the ‘real’ system in code

Level of Detail

Technology Dependence
The “Twin Peaks” conundrum

The ‘Real’ work system

Work System Models and taxonomy

Requirements for the ‘real’ system

Mid-range Architectural models

Architecture(s) to realize the ‘real’ system in code

Level of Detail

Technology Dependence

Low

High

Low

High

Professor Truex

Enterprise Architectures
Six ‘essential’ components to a translation method

1. identification of system attributes
2. creating requirements from attributes
3. prioritizing and analyzing requirements
4. applying architectural style to requirements
5. validating requirements in the resulting system and;
6. managing architectural standards.
Architectural style

• An architecture style defines a family of systems in terms of a pattern of structural organization.

• An architectural style defines
  – a vocabulary of components and connector types
  – a set of constraints on how they can be combined
  – one or more semantic models that specify how a system’s overall properties can be determined from the properties of its parts
Professor Truex
Enterprise Architectures

Characteristics of a Good Architecture

• Resilient
• Simple
• Approachable
• Clear separation of concerns
• Balanced distribution of responsibilities
• Balances economic and technology constraints
Antecedent #1: Quality Attributes & Tactics
(Bass et al 2003)

• Deals with non-functional requirements
• Attributes include
  – availability, modifiability, performance, security, testability and usability
  – Tactics or design decisions are applied to each attribute and a
  – collection of tactics leads to an Architectural Strategy.
• Uses ‘Use case’ diagram
• suggests standard architectural strategies or patterns that might be utilized
• Good for brainstorming
Antecedent #2: Component-Bus-System-Property (CBSP)

- A structured method for
  - identifying and classifying key architectural elements and the dependencies among those elements based on system requirements.
- Assists in identifying patterns and styles
- Privileges reuse
Antecedent #3:
Blitz Quality Functional Deployment (QFD)
(Zultner 2000)

- From the software design domain
- Scaled-down QFD model from manufacturing area
  - facilitates the formulation and prioritization of requirements from stakeholders
  - Uses matrices to map requirements and functionality
  - Helps to provide direct requirement tracibility
Antecedent #1: Architecture Standards Model (Boh et al, 2002)

- Another high-level model
- Divides architectural standards into two categories
  - Infrastructural architectures
    - Typically IT professional driven
    - Rather narrow and it focused
  - Integration architectures
    - User driven and business driven
- Recommends approaches to
  - requirements identification, standards setting, standards communication and standards conformance
What would a complete method include?

Ways to:

1. extract wants from stakeholders and clearly translating those wants into system attributes (see QFD)
2. decompose system attributes into requirements (see Quality Attributes & Tactics)
3. prioritize and identify mismatch between requirements (see CBSP)
4. apply an architectural style to a set of requirements (see Quality Attributes & Tactics or CBSP)
5. validate the accomplishment of requirements in a resulting system architecture (see QFD or Quality Attributes & Tactics)
6. create and manage resulting architectural standards (see Infrastructure/Integration Architecture Standards)
CBSP meta model

Figure 2: CBSP Meta Model
CBSP Process

Figure 3: CBSP process
Terminology

• CBSP
  – Component
  – Bus
  – System
  – Property

• ETVX
  – Entry
  – Task
  – Verity/validate
  – exit
Overview of approach

1. Identify requirements for next iteration (ETVX)
   - Users and architects; but user driven
   - Decide the essential requirements by (1) project relevance and value  (2) technical feasibility

2. Classify requirements
   ETV and vote on the degree of import and fit

3. Identify mismatches

4. Architects refine requirements
   1. With properties
   2. Without properties

5. Trade off and fine ‘styles’ to reuse
   Goal reuse and minimal new design
# Styles mapping matrix

## Table 7: CBSP to Style Mapping

<table>
<thead>
<tr>
<th>CBSP Dimensions</th>
<th>Properties</th>
<th>Client-Server</th>
<th>C2</th>
<th>Event-Based</th>
<th>Layered</th>
<th>Pipe-and-Filter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Component</td>
<td>aggregated</td>
<td>++</td>
<td>++</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>persistent</td>
<td>+</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>streamed</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>cached</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Processing</td>
<td>service provide / consume only</td>
<td>+</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Component</td>
<td>has N interfaces</td>
<td>+</td>
<td>+</td>
<td>++</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>stateful</td>
<td>+</td>
<td>++</td>
<td>++</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Loose coupling</td>
<td>+</td>
<td>+</td>
<td>++</td>
<td>-</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>can be migrated</td>
<td>+</td>
<td>++</td>
<td>++</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Connector/bus</td>
<td>synchronous</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>++</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>asynchronous</td>
<td>-</td>
<td>+</td>
<td>++</td>
<td>-</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>local</td>
<td>-</td>
<td>++</td>
<td>0</td>
<td>++</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>distributed</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>secure</td>
<td>+</td>
<td>0</td>
<td>0</td>
<td>+</td>
<td>0</td>
</tr>
<tr>
<td>(sub)System</td>
<td>efficient</td>
<td>0</td>
<td>+</td>
<td>+</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>scalable</td>
<td>+</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>evolvable</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>-</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>portable</td>
<td>0</td>
<td>+</td>
<td>0</td>
<td>++</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>reliable</td>
<td>0</td>
<td>0</td>
<td>-</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>dynamically reconfigurable</td>
<td>+</td>
<td>++</td>
<td>++</td>
<td>-</td>
<td>++</td>
</tr>
</tbody>
</table>

**Legend:** ++ extensive support   + some support   o neutral   - no support
Example of relationship requirement to CBSP model
Example of property mapping to architecture-relevant requirements

**Architecture-Relevant Requirements**

- R46: The system should allow the addition and removal of warehouses and vehicles
- R09: Support Cargo Arrival and Vehicle Availability Estimation
- R10: Automatic or Manual Routing of Vehicles
- R56: In case of network downtime, the system should maintain the latest known facts about warehouses and vehicles
- R42: The system should be reliable in the face of network failures (message loss)
- R59: Vehicles are equipped with location emitters but generally cannot receive commands during driving (outside warehouses)
- R26: The system must support both Metric and Imperial measurement systems in its displays (but not necessarily simultaneously)

**CBSP Properties**

- R42_S: Cargo Router (global)
  - SP: reliable
- R10_B: Comm-Link to Vehicle
  - BP: distribute
- R10_Cp: Route
  - CP: persistent
- R01_1c: Cargo (weight, shape)
  - CP: cached
- R10_Cp: Warehouse
  - CP: cached
- R09_Cp: Vehicle
  - CP: cached
  - CP: streaming
- R29_Cp: User Interface for the capture of vehicle, warehouse, and route information
  - loose coupling: yes

**Figure 6: Identifying properties in architecture-relevant requirements**