Enterprise Modeling for Digital transformation
Agenda

➡ Enterprise Architecture Challenges
➡ Architecture Repositories Ecosystem
➡ Architecture Model Taxonomies
➡ Model structures & building blocks
Initially focused on providing guidance for system design, Enterprise Architecture has gained acceptance as an approach to manage change and foster IT/business alignment by:

- (a) propagating strategy and process changes to the software and infrastructure level,
- (b) supporting consistent business transformation enabled by technology innovations, and by
- (c) decoupling business-oriented and technology-oriented architectures

Besides supporting strategy execution, a large number of other EA application scenarios exist, e.g. business continuity planning, security management, compliance management and sourcing management [Bu06; RB06].

EA is the primary tool for impact assessment and tradeoff analysis in these scenarios.
Architecture description challenges

- Enhancements of architecture descriptions
  - Be able to build architecture alternatives.
  - Be able to manage catalogs/libraries of reusable building blocks.
  - Be able to compare alternative architectures
  - Add planning dimension to static architecture descriptions

- Work in a collaborative manner.
Agenda

➡️ Enterprise Architecture Challenges
➡️ Architecture Repositories Ecosystem
➡️ Architecture Model Taxonomies
➡️ Model structures & building blocks
EA Repository requirements

In order to support EA functions, EA repositories must cover the following scope:

- Support EA descriptions (Content Model)
- Support EA activities (Management & Governance)
- Support relationships with operational enterprise data sources.
Enterprise Repositories Ecosystem

**External References**
- EA Frameworks, including EA Governance
  - TOGAF, ...
- Governance Frameworks
  - COSO: corporate, COBIT: IT...

**EA Repository**

**EA Function & Process Definition**
- EA Functions Definitions
- EA Skills

**EA Content Definition**
- Enterprise Meta-Model Definition
- Architecture Work Product Definitions & Templates
  - Diagrams, Matrix, Tables, ...

**EA Content**

**EA Governance Repository**
- EA Organization
- Decisions Log
- Workflows
- Assessments
- Architecture Projects
- Enterprise Concerns & Drivers
- EA Stakeholders

**Standards**
- Standard References
  - Reference implementation
  - Provides uses

**Architecture Work Products**
- Produced by 'regular' ADMs
  - Ability Models
    - Business Capabilities
  - Resource Capabilities
  - Operating Models
    - Business Operating Model
    - Solution Building Block

**Operational Repositories**
- Planning Information
- Portfolio of Initiatives
- Governance Definition
  - Enterprise Governance Repository
- Enterprise Requirement
- Digital OBB
  - Operation Building Block
  - CMDBs, BI, ...
- Non-digital OBB
  - Operation Building Block
  - person, machine, building...

**Operational Repositories**

**Definitions**
- defines the structure of
- realizes
Agenda

» Enterprise Architecture Challenges
» Architecture Repositories Ecosystem
» Architecture Model Taxonomies
» Model structures & building blocks
Architecture Models Taxonomy

Architecture meta-models are organized according to three main dimensions:

✔️ Models which describe finalities, purposes of enterprise & enabling systems: capabilities.

✔️ Models which describe how enterprises & enabling systems shall operate to fulfill expected capabilities.

✔️ Models which describe when things are expected (capabilities or systems): they are used for road-mappings.

PS: an enterprise is a system of systems.
 Abilities express expectation of desired Effects under a set of environmental constraints and expected measurable qualities.

 Performers (aka systems) conduct and participate to processes in order to produce deliverables made available through exposed services, thereby responding to expected abilities.
 Abilities express expectation of desired **Effects** under a set of environmental constraints and expected **measurable qualities**.

 Performers (aka systems) conduct and participate to **processes** in order to produce **deliverables** made available through exposed **services**, thereby responding to expected **abilities**.

---

### Model Kind

<table>
<thead>
<tr>
<th>Capability Models</th>
<th>Operating Models</th>
</tr>
</thead>
<tbody>
<tr>
<td>What is expected</td>
<td>What is produced</td>
</tr>
<tr>
<td>Ability</td>
<td>Effect</td>
</tr>
</tbody>
</table>

- Ability has desired effects
- Effect exposes
- Performer delivers
- Information exhibits
 Abilities express expectation of desired Effects under a set of environmental constraints and expected measurable qualities.

 Performers (aka systems) conduct and participate to processes in order to produce deliverables made available through exposed services, thereby responding to expected abilities.
 Abilities express expectation of desired Effects under a set of environmental constraints and expected measurable qualities.

 Performers (aka systems) conduct and participate to processes in order to produce deliverables made available through exposed services, thereby responding to expected abilities.
Enterprise Capabilities & Operating Models

<table>
<thead>
<tr>
<th>Model Kind</th>
<th>What is expected</th>
<th>What is produced</th>
<th>Who is in charge</th>
<th>What shall be remembered</th>
<th>How things are done</th>
<th>How things are exposed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ability</td>
<td>Effect</td>
<td>Performer</td>
<td>Information</td>
<td>Process</td>
<td>Services</td>
<td></td>
</tr>
</tbody>
</table>

- Ability has desired effects
- Performer exhibits
- Information exposes
- Process delivers
- Services activates
- Stores produces & consumes

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Business Layer</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Organization Layer</td>
<td>Skill</td>
<td>Content</td>
<td>Org-Unit</td>
<td>Data</td>
<td>Organizational Process</td>
<td>Exchange Contract</td>
</tr>
<tr>
<td>Application Layer</td>
<td>Functionality</td>
<td>Content</td>
<td>Application System Application</td>
<td>Data</td>
<td>System Process</td>
<td>Exchange Contract</td>
</tr>
<tr>
<td>Hardware layer</td>
<td>Functionality</td>
<td>Content</td>
<td>Artifact</td>
<td>Technical Data</td>
<td>System Process</td>
<td>Exchange Contract</td>
</tr>
<tr>
<td>Resource Configuration layer</td>
<td>Business Capability</td>
<td>Functionality</td>
<td>Content</td>
<td>Resource Architecture</td>
<td>Data &amp; Technical Data</td>
<td>System Process</td>
</tr>
</tbody>
</table>
Capability planning

- Capabilities are expected over period of time represented as desired phases of the enterprise (enterprise = undertaking).
- Each enterprise phase expresses a disposition to delivery of a set of capabilities, under particular quantified qualities.
Enterprise planning adds a time dimension to enterprise architecture in order to:

- Plan what is expected, why and how much.
- Specify accordingly which architecture should apply for each phase.
- Ensure that enterprise assets (deployed assets on the ground) are delivered or decommissioned accordingly.
Agenda

- Enterprise Architecture Challenges
- Architecture Repositories Ecosystem
- Architecture Model Taxonomies
- Model structures & building blocks

☑️ State of the art
☑️ d
Composition pattern: what is the problem?

- We need to address the following concerns for architecture descriptions:
  - Be able to build architecture alternatives.
  - Be able to manage catalogs/libraries of reusable building blocks.
  - Be able to compare alternative architectures.
  - Be prepared for enterprise transformation.

- Problem:
  - The two common modeling syntax used for architecture descriptions - hierarchies and flat models - prevent from creating effective scope for building blocks, thereby denying the notion of building block itself.
Problem 1: hierarchical models & interconnections

- Benefits of hierarchical models.
  - Follow the usual breakdown practice (Cartesian approach).
  - Provide scopes for building blocks. This sometimes represented by naming conventions, such as, “X.1.1” and “X.1.2” are in “X.1”. IDEF notations are a good example.

- Problem: hierarchical structures hardwire building blocks together:
  - Blocks can only be part of a single hierarchy: the single parent syndrome.
  - If multiple parentship are allowed, inter-connections become undefined: the (*,*) relationship syndrome (see next slide).
Let’s consider two application hierarchies: “APP 1..” (on the left below) and “APPX X..”. (on the right below)
When Sub-Application “APP XY” is composed of “APP 1.2” does this implies that:
- The “Msg flow 1” message flow becomes part of the “APP X” application tree?
- The “Msg flow 1” message flow becomes part of the definition of the “APP 1.2” application?

Similar issues occur for sequence between processes, flows between processes, etc. They prevent from having autonomous building blocks.
Problem 2: flat models

- Benefits of flat models:
  - Avoid the single parent syndrome of hierarchal models.
  - Enable a natural discovery of building blocks and general dependencies.

- Problem:
  - Architecture scopes have been lost (is eCommerce in HR system?): there is a single global graph.
  - Diagrams are used for pseudo scoping while they have no semantic value (environment diagrams) => back to Visio.
  - Adding a connector at one end of the graph has undefined effect on the rest of the graph, hence building blocks do not have autonomous definitions.

Questions:
- Is Recruitment changed?
- Is Training changed?
- Both?

Where is the change impact scope? Recruitment, HR System, ... the entire graph?
Archimate Flat Models

- "Car insurance" is composed of "Change conditions", "Policy" and "Submit Claim".

- Independently of Car Insurance", "Change Conditions", "Policy" et "Submit Claim" are connected by dependency relationships.
A second model "Car Insurance 2" involves a "Policy 2" object which replace the initial "Policy" object.

"Change conditions" and "Submit claim" keep their initial relationships with the initial "Policy" object.

These links are graphically hidden in the collapsed diagram.

We are back to Visio.
Decomposition/Composition principles

➡ Design principles

✔ Autonomy: decomposition is always designed at **two levels of depth only**.
  - The building block and its direct components.

✔ The complete block structure is obtained through recursive analysis of the following triple:
  - building block -> block use -> building block (process -> calling activity -> process).

➡ Benefits

✔ Reduce block structure complexity (the unresolvable question: how many levels of depth?)
✔ Homogeneity of building blocks.
✔ Block use enables better requirement design:
  - Requirements are set at the level of block usage: what is expected from a block in a specific context (usage).
  - Each block indicates what it is able to achieve independently of how it is used.

![Diagram showing good and bad use of building blocks](image)
Models are organized as elementary blocks, for instance Business Functions. Various assemblies enable the building of alternatives architectures in regard to common reference model (capability below).
The same approach applies at any layer, for instance at the physical layer, building blocks are Artifacts and Org-Units.

Various assemblies (resource architectures) enable the building of alternatives solutions for a common reference business architecture.
The same approach applies at any layer. For instance, at the application layer, building blocks are Application Systems and Applications.

Various assemblies (Application Systems) enable the building of alternatives solutions for a common reference logical application architecture.
The building block approach allows to detect what has been added and removed from one source structure to another target structure. However, this type of comparison doesn’t tell whether these adds and removals are replacements or effective adds and removals.
Composition & comparison: changes

⇒ Using a reference model as an invariant source for comparison enables to move from difference analysis to change analysis.
Composition & comparison & Time: transformation

⇒ Architecture Block assemblies are TIMELESS.

⇒ Adding a time layer, ON TOP OF, block assemblies, enable to address transformation.
Composite structure benefit summary

- Provide catalogs/libraries of autonomous architecture building blocks.
- Provide alternative assemblies of these building blocks.
- Provide comparison of alternative assemblies.
- Pave the way for enterprise transformation.
- Enable effective multi-layered approaches for enterprise modeling.

See Herbert Simon: Parable of the two watchmakers.