Software Development Methodologies

Lecturer: Raman Ramsin

Lecture 15

Situational Method Engineering
Methodology Engineering

- Motivated by the prevalent belief that no one methodology fits all situations.

- First introduced by Kumar and Welke in 1992 as a discipline aimed at constructing methodologies to match given organizational settings or specific development projects.

- Later came to be known as *Method Engineering*, a term proposed by Brinkkemper in 1996;

  - Definition: "The engineering discipline to design, construct, and adapt methods, techniques and tools for the development of information systems".

- The most well-known subfield is *Situational* Method Engineering (SME): concerned with the construction/adaptation of a methodology specifically attuned to the project at hand.
Methodology Engineering: Alternative Approaches

- **Ad-hoc**: Concerned with constructing a new methodology from scratch.

- **Configuration-based**: Concerned with configuring methodologies and method components based on the situational context at hand.

- **Activity-diagram-based**: Concerned with application of UML Activity Diagrams for representing process-focused actions by Activities.

- **Deontic-matrices-based**: Concerned with using matrices of values for representing relationships of Activity-Task pairs chosen from OPF repository.

- **Paradigm-based**: Concerned with instantiating, abstracting or adapting an existing meta-model in order to produce the target methodology.

- **Extension-based**: Concerned with enhancing an existing methodology with new concepts and properties by using extension patterns.

- **Assembly-based**: Concerned with constructing the target methodology or enhancing an existing methodology by reusing parts of other methodologies.
Generic Model for Situational Method Engineering

[Henderson-Sellers et al. 2014]
Extension-Based SME

[Domain-driven strategy]

Select a meta pattern

[Pattern-based strategy]

Extend a method

[Pattern-matching strategy]

Start

Stop

Completeness strategy

[Henderson-Sellers et al. 2014]
Paradigm-Based SME

[Reference: Henderson-Sellers et al. 2014]
Assembly-Based SME

[Start]

- Requirements driven

Select a method part

- Decomposition
- Aggregation
- Refinement

Assemble method parts

- Completeness validation
- Integration
- Association

Stop

Completeness validation

[Henderson-Sellers et al. 2014]
Assembly-Based SME

[Mirbel and Ralyté 2006]
Method Chunk

[Mirbel and Ralyté 2006]
Assembly-Based ME In OPEN/OPF

[Framework/metamodel]

[Repository of process components]

[Example 1 of Concept A]
[Example 2 of Concept A]
[Example 1 of Concept B]
[Example 2 of Concept B]
[Example 1 of Concept C]
[Example 2 of Concept C]
[Example 1 Role 1]
[Example 2 Role 1]
[Example 1 Role 2]
[Example 2 Role 2]

[Henderson-Sellers 2003]
Assembly-Based ME In OPEN/OPF

[OPEN Process Framework]

**Method/Process Metamodel**

Repository of Predefined Method/Process Components

Construction Guidelines

Step 1. Method/Process Engineer selects Method/Process Components and constructs Methodology

Step 2. Project Manager creates Method/Process Instance by allocating specific resources

[Henderson-Sellers 2003]
OPF Repository

- Contains a range of predefined instances for each class and subclass in the OPF metamodel; e.g.:
  - 30 predefined instances of Activity
  - 160 instances of Task
  - 200 instances of Techniques
  - 76 instances of Role
## OPF: Task-Activity Matrix

<table>
<thead>
<tr>
<th>Task</th>
<th>Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Code</td>
<td></td>
</tr>
<tr>
<td>Construct the object model</td>
<td></td>
</tr>
<tr>
<td>Develop and implement resource allocation plan</td>
<td></td>
</tr>
<tr>
<td>- develop iteration plan</td>
<td></td>
</tr>
<tr>
<td>- develop timebox plan</td>
<td></td>
</tr>
<tr>
<td>- set up metrics collection program</td>
<td></td>
</tr>
<tr>
<td>- specify quality goals</td>
<td></td>
</tr>
<tr>
<td>Evaluate quality</td>
<td></td>
</tr>
<tr>
<td>Identify CIRTs (Class, Instance, Role, or Type)</td>
<td>X</td>
</tr>
<tr>
<td>Map roles onto classes</td>
<td></td>
</tr>
<tr>
<td>Test</td>
<td></td>
</tr>
<tr>
<td>Write manuals and other documentation</td>
<td></td>
</tr>
</tbody>
</table>

**Key**
1. Project planning
2. Modeling and implementation: OO analysis, design, programming
3. Verification and validation
4. User review
5. Consolidation
6. Evaluation

[Henderson-Sellers 2003]
# OPF: Technique-Task Matrix

<table>
<thead>
<tr>
<th>Techniques</th>
<th>Task 1</th>
<th>Task 2</th>
<th>Task 3</th>
<th>Task 4</th>
<th>Task 5</th>
<th>Task 6</th>
<th>Task 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract class identification</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Abstraction utilization</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Class internal design</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Class naming</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Collaborations analysis</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Complexity measurement</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Contract specification</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Coupling measurement</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>CRC card modeling</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Generalization and inheritance identification</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Implementation of services</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Implementation of structure</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Inspections</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Interaction modeling</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Package and subsystem testing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Prototyping</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Relationship modeling</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Responsibility identification</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Role modeling</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Service identification</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>State modeling</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Textual analysis</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Timeboxing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Unit testing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Walkthroughs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

**Key**

1. Code
2. Construct the object model
3. Develop and implement resource allocation plan
4. Evaluate quality
5. Identify CIRTs
6. Map roles onto classes
7. Test

[Henderson-Sellers 2003]
Hybrid Methodology Design

- Alternative Approaches:
  - *Instantiation approach*: instantiating an already available process metamodel
  - *Artefact-oriented approach*: devising a seamless complementary chain of artefacts and building the process around it
  - *Composition approach*: using one of the already available libraries of process patterns
  - *Integration approach*: integrating features, ideas and techniques from existing methodologies

- *Hybrid design approach*: using different alternatives from among the above-mentioned for different parts of the process and/or at different levels of abstraction
Hybrid Design Process

1. Identify Next Abstraction Level and Revise Requirements
2. Define and Apply Hybrid Design Methodology
3. Prioritize Requirements
4. Integrate Methodology Elements into Methodology
5. Refine and Revise Methodology
6. If Stabilized and Complete, Finalize Methodology

Methodology Elements

Prioritized Requirements

Methodology

Process-Centred Description of Methodologies

Analysis Results

[Ramsin 2006]
Hybrid Design: Emphasis on Approaches during Design

- Instantiation Approach
- Artefact-Oriented Approach
- Composition Approach
- Integration Approach

Design Start → Design Finish

[Diagram showing the duration of each approach over iterations]

[Ramsin 2006]
Hybrid Design: Sample Iterations

Analysis  Design  Implementation  Test  Transition
Hybrid Design: Sample Iterations

Iteration 1

- Preliminary Analysis
- Detailed Analysis
- Architectural Design
- Detailed Design
- Implementation and Test
- Transition

[Ramsin 2006]
Hybrid Design: Sample Iterations

Iteration 1

- Preliminary Analysis
- Detailed Analysis
- Architectural Design
- Detailed Design
- Implementation and Test
- Transition

Iteration 2

- Preliminary Analysis
- Detailed Analysis
- Architectural Design
- Detailed Design
- Implementation and Test
- Transition
- Executable Package

[Ramsin 2006]
Hybrid Design: Sample Iterations (Contd.)

Ramsin 2006
Hybrid Design: Sample Iterations (Contd.)
Hybrid Design: Sample Iterations (Contd.)

Iteration 2

[Diagram showing stages of Hybrid Design with arrows and boxes for Preliminary Analysis, Detailed Analysis, Architectural Design, Detailed Design, Implementation and Test, and Transition]

Iteration 3

[Diagram showing additional stages with arrows and boxes for Domain Modeling and Requirements Elicitation, System Specification, Software Design, and Executable Package]

Iteration 4

[Diagram showing further stages with arrows and boxes for Plan by Feature, Development Engine, Design by Feature, Build by Feature, and Executable Package]
References


