Towards an Agile Foundation for the Creation and Enactment of Software Engineering Methods: The SEMAT Approach

Brian Elvesæter¹, Michael Striewe², Ashley McNeile³ and Arne-Jørgen Berre¹

¹ SINTEF ICT, P. O. Box 124 Blindern, N-0314 Oslo, Norway
   brian.elvesater@sintef.no, arne.j.berre@sintef.no
² University of Duisburg-Essen, Gerlingstrasse 16, D-45127 Essen, Germany
   michael.striewe@s3.uni-due.de
³ Metamaxim, 48 Brunswick Gardens, London W8 4AN, UK
   ashley.mcneile@metamaxim.com
Outline

- Background and Motivation
- Overview of the SEMAT Approach
- Language Requirements
- Language Metamodel
  - Specification of Kernel elements
  - Specification of Practices (Scrum example)
  - Enactment (Scrum example)
- Conclusions and Future Work
Background and Motivation

• Software development is driven by fashions and fad
  – huge number of methods and method variants
• SPEM 2.0 software engineering process framework
  – viewed by development teams as being too heavyweight and inflexible
  – requires separate process or method engineers
• SEMAT (Software Engineering Method and Theory)
• OMG Request for Proposal (RFP)
  – "A Foundation for the Agile Creation and Enactment of Software Engineering Methods”
  – June 2011
• Solicits submissions for
  – a Kernel model of widely agreed elements
  – a minimal domain-specific Language to describe and enact software engineering methods
• Targets
  – practitioners rather than method engineers
  – focuses on usage and enactment
Overview of the SEMAT Approach

<table>
<thead>
<tr>
<th>Model</th>
<th>Kernel (essentials)</th>
<th>Practice (specific guidelines)</th>
<th>Method (composition and enactment)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><img src="image" alt="Kernel" /></td>
<td><img src="image" alt="Practice" /></td>
<td><img src="image" alt="Method" /></td>
</tr>
</tbody>
</table>

- **Kernel (essentials)**: 
  - ![Alpha](image)
  - Activity Space

- **Practice (specific guidelines)**: 
  - Use Case
  - User Story
  - Scrum
  - Scrum of Scrums
  - Test Driven Development

- **Method (composition and enactment)**: 
  - User Story
  - ![Kernel](image)
  - Scrum
  - Scrum of Scrums

**Language constructs**

- ![State](image)
- Activity
- ![Checkpoint Pattern](image)
- Work Product
- Composition
- Dynamic Semantics

*Image credit: SINTEF ICT*
## Language Definition (1.x) and Language Features (2.x) Requirements

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>MOF metamodel</td>
<td>Abstract syntax defined in MOF.</td>
</tr>
<tr>
<td>1.2</td>
<td>Static and operational semantics</td>
<td>Static and operational semantics defined in terms of the abstract syntax.</td>
</tr>
<tr>
<td>1.3</td>
<td>Graphical syntax</td>
<td>Graphical syntax that maps to the abstract syntax.</td>
</tr>
<tr>
<td>1.4</td>
<td>Textual syntax</td>
<td>Textual syntax that maps to the abstract syntax.</td>
</tr>
<tr>
<td>1.5</td>
<td>SPEM 2.0 metamodel reuse</td>
<td>Reuse SPEM 2.0 metamodel where appropriate.</td>
</tr>
<tr>
<td>2.1</td>
<td>Ease of use</td>
<td>Easy to use by practitioners.</td>
</tr>
<tr>
<td>2.2</td>
<td>Separation of views</td>
<td>Separation of two different views of a method for practitioners and method engineers.</td>
</tr>
<tr>
<td>2.3</td>
<td>Specification of kernel elements</td>
<td>Description, relationships, states, instantiation and metrics.</td>
</tr>
<tr>
<td>2.4</td>
<td>Specification of practices</td>
<td>Cross-cutting concern, element instantiation, work products, work progress, verification.</td>
</tr>
<tr>
<td>2.5</td>
<td>Composition of practices</td>
<td>Overall concerns, merging elements, separating elements, practice substitution.</td>
</tr>
<tr>
<td>2.6</td>
<td>Enactment of methods</td>
<td>Tailoring, communication, managing, coordinating, monitoring, tooling.</td>
</tr>
</tbody>
</table>
Specification of Kernel elements

- Requirements
- Software System
- Team
- Work

- Performs and plans scope and constrain
- Produces
- < fulfils
- Updates and changes>
- < performs and plans

- Prepare to do the Work
- Coordinate the Work
- Support the Team
- Track Progress
- Stop the Work
# States of the Work alpha

<table>
<thead>
<tr>
<th>States (State Graph)</th>
<th>Description (State)</th>
<th>Checklist items (Checkpoints)</th>
</tr>
</thead>
</table>
| Initiated            | The work has been requested. | • The result required of the work being initiated is clear.  
                               • Any constraints on the work’s performance are clearly identified. 
                               • … |
| Prepared             | All pre-conditions for starting the work have been met. | • Commitment is made.  
                               • Cost and effort of the work are estimated.  
                               • … |
| Started              | The work is proceeding. | • Development work has been started.  
                               • Work progress is monitored.  
                               • … |
| Under Control        | The work is going well, risks are under control, and productivity levels are sufficient to achieve a satisfactory result. | • Work items are being completed.  
                               • Unplanned work is under control.  
                               • … |
| Concluded            | The work to produce the results has been concluded. | • All outstanding work items are administrative housekeeping or related to preparing the next piece of work.  
                               • Work results are being achieved.  
                               • … |
| Closed               | All remaining housekeeping tasks have been completed and the work has been officially closed. | • Lessons learned have been itemized, recorded and discussed.  
                               • Metrics have been made available.  
                               • … |
Specification of Practices (Scrum example)

- Supporting the practitioner
  - How to represent Scrum in the language?
  - How to ensure that software developers follow and apply the principles of Scrum in a correct way?
  - How to support enactment in the daily working environment (the tools) that software developers are using?

- Scrum team (roles)
  - Product Owner
  - Development Team (of developers)
  - Scrum Master

- Scrum artifacts
  - Product Backlog
  - Sprint Backlog
  - Increment

- Scrum events
  - The Sprint
  - Sprint Planning Meeting
  - Daily Scrum
  - Sprint Review
  - Sprint Retrospective
Metamodel instantiation

M2: Metamodel

Alpha  Activity  WorkProduct  Class

<<instanceOf>>  <<instanceOf>>  <<instanceOf>>  <<instanceOf>>

M1: Kernel

Work

M1: Static

ProductBacklog

Sprint

SprintPlanning

M1: Dynamic

my_Alpha

+instanceName : String
+currentState : State
+myWorkProductInstances : my_WorkProduct [*]
+mySubAlphaInstances : my_Alpha [*]

my_WorkProduct

+instanceName : String

my_Alpha.Ext

start_time : Time
+endTime : Time
+duration : Time

my_Activity

+instanceName : String
+myAlphaInstances : my_Alpha [*]
+myWorkProductInstances : my_WorkProduct [*]
# Scrum example: Details of Sprint

<table>
<thead>
<tr>
<th>Sprint (Alpha)</th>
<th>States of Sprint</th>
<th>Checklist items (Checkpoints)</th>
</tr>
</thead>
</table>
|               | Planned          | • Sprint Planning Meeting is held.  
|               |                  | • Product Owner presents ordered Product Backlog items to the Development Team.  
|               |                  | • Development Team decides how it will build this functionality into a "Done" product Increment during the Sprint.  
|               |                  | • Scrum Team crafts a Sprint Goal.  
|               |                  | • Development Team defines a Sprint Backlog.  
|               | Started          | • Team is taking their work items from the Sprint Backlog. |
|               | Under Control    | • Daily Scrum optimizes the probability that the Development Team will meet the Sprint Goal.  
|               |                  | • Every day, the Development Team should be able to explain to the Product Owner and Scrum Master how it intends to work together as a self-organizing team to accomplish the goal and create the anticipated increment in the remainder of the Sprint.  
|               | Concluded        | • During the Sprint Review, the Scrum Team and stakeholders collaborate about what was done in the Sprint.  
|               | Closed           | • A Sprint Review Meeting is held at the end of the Sprint.  
|               |                  | • The Sprint Retrospective occurs after the Sprint Review and prior to the next Sprint Planning Meeting.  

## Diagram:

- **Work**
  - **Sprint (Alpha)**
  - **States of Sprint**
    - Planned
    - Started
    - Under Control
    - Concluded
    - Closed
  - **Sprint Backlog**

- **1..***
  - **1**
Understanding Enactment

- 1. How does the level 0 model look like?
- 2. How do elements enter or leave the level 0 model?
- 3. How are the properties of elements in the level 0 model updated?
- 4. What statements (e.g., progress health, what to do next, ...) can be derived from my level 0 model?
- 5. How are the elements in the level 0 model related to elements in the level 1 models (i.e. in the Kernel or a Practice)?
Scrum example: Model instantiation

M1: Kernel
- Work
- ProductBacklog
- Sprint
- SprintPlanning

M1: Static
- M1: Dynamic
  - my_Alpha
    - +instanceName: String
    - +currentState: State
    - +myWorkProductInstances: my_WorkProduct[*]
    - +mySubAlphaInstances: my_Alpha[*]
  - my_Alpha_Ex
    - +startTime: Time
    - +endTime: Time
    - +duration: Time
  - my_Activity
    - +instanceName: String
    - +myAlphaInstances: my_Alpha[*]
    - +myWorkProductInstances: my_WorkProduct[*]

M0: Endeavour
- [ProductBacklogID1]: ProductBacklog
  - instanceName = "Project Product Backlog"
- [WorkID1]: Work
  - currentstate = "Under Control"
  - instanceName = [ProjectWork]
  - mySubAlphaInstances = [SprintID1], [SprintID2]
  - myWorkProductInstances = [ProductBacklogID1]
- [SprintPlanningID1]: SprintPlanning
  - instanceName = "Planning Game for Sprint #2"
  - myAlphaInstances = [SprintID2]
  - myWorkProductInstances = [ProductBacklogID1]
- [SprintID1]: Sprint
  - currentState = "Closed"
  - duration = "2 weeks"
  - endTime = "15-Apr-2012"
  - instanceName = "Sprint #1"
  - myWorkProductInstances = [ProductBacklogID1]
  - startTime = "01-Apr-2012"
- [SprintID2]: Sprint
  - currentState = "Planned"
  - duration = "n/a"
  - endTime = "n/a"
  - instanceName = "Sprint #2"
  - myWorkProductInstances = [ProductBacklogID1]
  - startTime = "16-Apr-2012"
Scrum example: Enacting a Sprint

- Existing collaborative environments
  - JIRA, Trac, FusionForge, Redmine
- Rich set of features
  - issue tracking
  - calendar, time tracking
  - wiki, forums
  - source code versioning
- Typically lack support for software engineering methods
  - introduce progress health dashboards (alphas)
  - introduce alpha state cards
Conclusions and Future Work

- The SEMAT approach
  - A minimal language with a few key language constructs to cover the essentials
    - **Static semantics defined at M2**: Kernel, Alpha, ActivitySpace, Practice, WorkProduct, Activity, State, Checkpoints, …
    - **Dynamic semantics defined at M1**: my_Alpha, my_Activity, my_WorkProduct, …
  - A baseline domain model, the Kernel (M1 layer), of reusable elements that we extend when defining our Practices and Methods.
    - **Alphas**: Requirements, Software System, Work, Team, …
    - **Activity Spaces**: Prepare to do the Work, Coordinate the Work, …
- Currently preparing a revised submission to the OMG RFP (August 2012)
  - Introducing a set of standard sub-alphas as part of the Kernel
    - Task (Work), Requirements Item (Requirements), Member (Team), …
  - Revised metamodel, including the dynamic semantics part.
    - Defining additional endeavour-level properties for the sub-alphas
    - Better support for extensions
  - Revised graphical notation
Thanks for your attention!

- Questions?

- SEMAT website:
  - http://www.semat.org/

- NEFFICS website:
  - http://www.neffics.eu/

- REMICS website:
  - http://www.remics.eu/

- SiSaS website:
  - http://sisas.modelbased.net/