

METHOD FRAGMENTS TO SUPPORT COLLABORATIVE TEAMWORK FOR SOFTWARE DEVELOPMENT PROJECTS

Numi Tran, Brian Henderson-Sellers, Igor Hawryszkiewicz
University of Technology

Abstract

Collaborative work, and especially CSCW, requires methodological support. One methodological approach is that of situational method engineering (SME). In the SME context, support for CSCW teams and teamwork has been identified as being deficient. Using a repository of method fragments constructed from a standard metamodel, we analyze the gaps and propose new method fragments that will allow a methodology focussed upon CSCW to be effectively constructed from the fragment repository.

Keywords: *Method engineering, CSCW, teams, method fragments*

1 INTRODUCTION: SUPPORTING COLLABORATIVE WORK THROUGH SITUATIONAL METHOD ENGINEERING

Situational method engineering, SME (Kumar and Welke, 1992; Brinkkemper, 1996; ter Hofstede and Verhoef, 1997) provides an approach to the creation of appropriate and highly focussed methodological approaches for software development e.g. Brinkkemper (1996), Henderson-Sellers (2003).

Computer supported collaborative work (CSCW), on the other hand, is a well-established subdiscipline of information technology necessarily leading to discussion of teams/teamwork.

Here, we combine these two ideas. In the context of creating, through SME, an appropriate method or methodology¹ for CSCW, we focus on the need for CSCW team and teamwork support within one existing repository of method fragments – that of the OPEN Process Framework or OPF (Firesmith and Henderson-Sellers, 2002).

In Section 2, we present an overview of situational method engineering (SME) and its incarnation in the OPEN Process Framework. Section 3 discusses how new fragments are generated from the defining metamodel followed by a full description of newly identified method fragment to support collaboration and collaborative teamwork in Section 4.

¹ For present purposes we will take method and methodology as synonyms.

2 SITUATIONAL METHOD ENGINEERING (SME) WITH THE OPF

2.1 SME

Since it is now widely recognized e.g. Avison and Wood-Harper (1991), Vessey and Glass (1994), Fayad *et al.* (1996), Fitzgerald *et al.* (2003) that a one-size-fits-all method is unattainable, situational method engineering (SME) provides a solution to the problem of the identification of the “most appropriate” methodology for an organization and/or its projects. SME suggests that the elements of one (or more) methodology can be modularized and encapsulated as “method fragments” (van Slooten and Hodes, 1996) and stored in a repository or methodbase e.g. Brinkkemper (1996), Ralyté and Rolland (2001). From the methodbase are then extracted only those fragments relevant to the current situation. These fragments are then connected together using “construction guidelines” to form the situational method(ology).

2.2 The OPEN Process Framework.

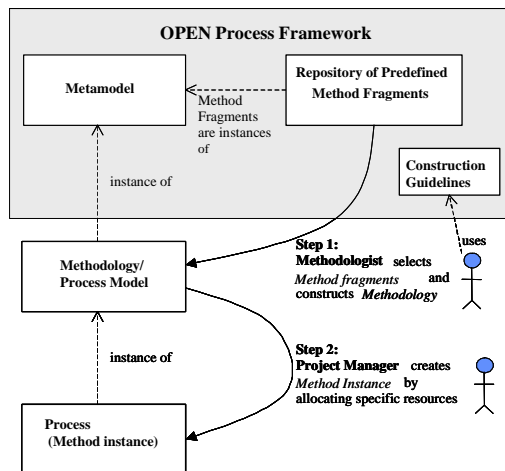


Figure 1 Method(ology) construction and enactment using the OPF approach to SME (after Henderson-Sellers and Hutchison, 2003)

One example of a method engineering approach that provides extensive, practical fragment support (over 1000 fragments are available) is the OPEN Process Framework or OPF (Firesmith and Henderson-Sellers, 2002; <http://www.opfro.org>), which consists of a repository of pre-defined method fragments underpinned by a standard metamodel (Figure 1). In the near future, the current, traditional, process-focussed metamodel of the OPF (as detailed in Appendix G of Firesmith and Henderson-Sellers, 2002) will be supplanted by the forthcoming ISO/IEC 24744 standard: Software Engineering Metamodel for Development Methodologies (discussed in more detail in Section 3). It is the predecessor of the ISO/IEC (2007) standard metamodel, the Australian Standard 4651 (Standards Australia, 2004), that will be used here as our standard generating “template” for the CSCW team fragments to be discussed in Section 4. In addition, it should be noted that the new metamodel supports

method enactment as well as method construction and brings together process-focussed components of a methodology and product-focussed components.

To create a situational method, various method fragments are chosen from the repository of the OPF and combined to describe the process model and its associated people and social issues, deliverables and so on. Using the tenets of SME outlined in Section 2.1, such a methodology/process model (Figure 1) can be specifically constructed and tailored towards a specific project or a specific organizational “standard” using the supplied construction guidelines together with a set of rules to link together the various method fragments – the OPF uses a set of deontic matrices, for example, which provide the ability to use fuzzy relationships between pairs of method fragment types thus giving a high degree of flexibility to the process engineer (Figure 1), perhaps assisted by an automated tool (Nguyen and Henderson-Sellers, 2003), who can allocate appropriate deontic values to any specific pair of process components depending upon the context i.e. the specific project, skills set of the development team etc.

Finally, as shown in Figure 1, the methodology/process model is enacted to create a value-specific instance applicable to the execution of a single project i.e. individuals’ names, actual deadlines and deliverables etc. are specified and progress monitored and managed in real (calendar) time for the duration of this project.

3 THE UNDERPINNING METAMODEL AND GENERATED FRAGMENTS

Although method engineering deals with the identification of method fragments and the construction of methodologies from those fragments, all at the “method” level (Figure 2), formal underpinning by means of a metamodel is well advised. A metamodel, as used by the OPF for instance, essentially captures the rules by which both fragments and constructed processes can be consistently and effectively used. These rules might say, for example, whether it is appropriate to sequence two activities, three techniques and then four roles (an occurrence that should be prevented since it is nonsensical). Using an object-oriented modelling approach, we describe the metamodel by a set of interacting classes, which we name metaclasses i.e. a metaclass is a class in a metamodel.

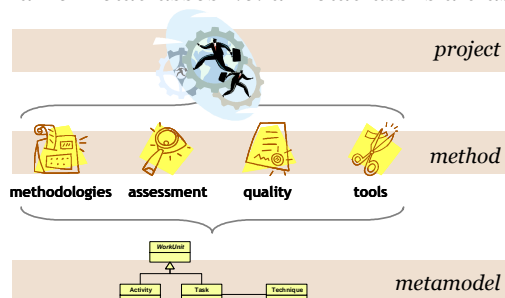


Figure 2 The three level architecture of AS4651 and ISO/IEC 24744

Following the original work that created the metamodel for the OPEN Process Framework e.g. Graham *et al.* (1997), Firesmith and Henderson-Sellers (2002), Standards Australia (the national standards body) published a metamodeling standard for methodologies (a.k.a. methods), the metamodel being named SMSDM – Standard Metamodel for Software Development Methodologies (Standards Australia, 2004). Although this standard (AS4651) is

the basis for an evolving ISO/IEC standard (24744), since this latter standard is, at the time of writing, not yet published, in this paper, we will use the earlier Australian Standard, while noting that all our conclusions will be equally valid in the context of ISO/IEC 24744, once published (ISO/IEC, 2007).

The ideas behind the SMSDM are documented in Henderson-Sellers and Gonzalez-Perez (2005). The architecture stands in some contrast with that of other process-focussed metamodels, such as the Object Management Group's SPEM (OMG, 2002), since the SMSDM uses the powertype pattern, as described by Odell (1994), throughout much of the model.

From the most abstract perspective, the SMSDM defines the classes MethodologyElement and ProjectElement to represent, respectively, elements in the methodology and the project layers (Figure 2). MethodologyElement is specialized into Resource and Template, corresponding to methodology elements that are used "as is" at the project level (i.e. resources) and methodology elements that are used by instantiation at the project level (i.e. templates) Gonzalez-Perez and Henderson-Sellers (2005). Since Template is the abstract type of all elements at the methodology level and ProjectElement is the abstract superclass of the same elements, these two classes form a powertype pattern in which Template is the powertype and ProjectElement is the partitioned type (Figure 3).

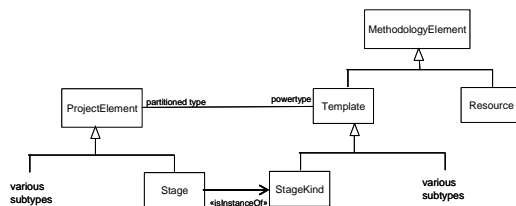


Figure 3 ProjectElement and Template form a powertype pattern since, for example, an instance of StageKind (subtype of Template) is Stage which is in turn a subtype of ProjectElement

The powertype pattern formed by Template and ProjectElement is refined into more specialised powertype patterns formed by subclasses of these two, namely

- StageKind and Stage (representing managed intervals of time, or points in time, within a project, i.e. the temporal aspect of the process),
- WorkUnitKind and WorkUnit (functionally cohesive operations performed during a project, i.e. the job aspect of the process),
- WorkProductKind and WorkProduct (any artifacts of value developed or used during a project),
- ActionKind and Action (representing the specific act or usage of a given work product by some types of work units),
- ProducerKind and Producer (finally responsible for actions, usually human),
- ModelUnitKind and ModelUnit (basic building blocks to be used in creating work products).

It should be noted that the suffix "Kind" is used to represent an element belonging to a *methodology*, which needs to be distinguished from an element that belongs to a particular *project*. For example, "Producers" refer to people involving in a particular systems engineering project, while "ProducerKind" refer to kinds of producers described by the

methodology used by that project. Note that project-level elements must be instances of some methodology-level elements. In this paper, we only define *Kind fragments, because our focus is on the methodology level, not the project level.

4 NEWLY IDENTIFIED FRAGMENTS TO SUPPORT COLLABORATION

For each of the various concepts expressed in the underpinning metamodel, fragments can be generated for storage in the repository (methodbase). Since the OPF methodbase already exists and contains a significant number of fragments, two possibilities arise: (i) the fragment necessary to support a specific, identified aspect of collaborative software development already exists or (ii) it does not exist and a new fragment must therefore be delineated.

We find no new fragments are necessary in the following categories (each category relating to a specific meta-element): TeamKind (subtype of ProducerKind, ModelKind (subtype of WorkProductKind), Language and Notation, ModelUnitKind, ModelUnitUsageKind, StageKind and ProcessKind (subtype of WorkflowKind). Note that if the reader wishes to check the prior existence of obvious CSCW kinds of fragments that may be brought to mind, unfortunately there is insufficient space in this paper to list them all and the original publications will need to be consulted.

In the following subsections, we outline the various kinds of new fragments that support collaborative work that are *not* already found in the repository i.e. the newly identified fragments that we must now identify - the full details of the fragments are found at http://www.open.org.au/fragments/collaboration_team_fragments.pdf. The meta-elements from which these are generated are only six: RoleKind, ToolKind, DocumentKind, TaskKind, TechniqueKind and ActivityKind

4.1 Role kinds

A Role is a collection of responsibilities that a producer can take. A RoleKind is a specific kind of role, characterized by the involved responsibilities. RoleKind is a subtype of ProducerKind, which is defined as an agent that executes work units i.e. a Producer Kind is a specific kind of producer, characterized by its area of expertise.

When we examine the existing OPF repository we find a number of fragments specific to teamwork. "Teamwork rolekinds" are kinds of roles that encapsulate responsibilities relating to the building, operation and management of *teams* and *teamwork*. They are to be distinguished from software-engineering roles (SE Roles), which encapsulate the responsibilities specific to the software engineering process. If a developer in a project belongs to a team, he/she should play *both* SE role(s) (e.g. programmer) and teamwork role(s) (e.g. team member).

To support teamwork in collaborative development we have identified four new teamwork-focussed RoleKinds: Team Leader, Team Member, Artefact Manager and Task Facilitator.

4.2 ToolKinds

A tool is an instrument that allows another producer to perform a work unit in an automated way. A tool kind is a specific kind of tool, characterized by its features.

No existing repository fragments describe teamwork tool support, only tools for software engineering activities. Consequently, we propose for addition to the repository a new fragment called Groupware ToolKind, which provides support for collaborative software development.

“Groupware Toolkind” is a kind of tools that “support and augments group work” (Greenberg, 1991). They are needed to support teams of developers working together. Their goal is to assist teams of developers in communicating, collaborating and coordinating their activities (Ellis *et al.*, 1991). Currently, OPEN’s repository only defines tools that support specific software engineering activities such as CASE tools and Documentation tools (e.g. Word processor). These tools do not aim at supporting developers’ teamwork during the software engineering process.

Potential sub-classes and instances of Groupware Toolkind (Terzis and Nixon, 1999) are Electronic Mail ToolKind, Conferencing ToolKind, Meeting Support ToolKind, Group Decision Support ToolKind and Workflow ToolKind.

4.3 DocumentKinds

A document is a durable depiction of a fragment of the observed reality. A document kind is a specific kind of document, characterized by its structure, type of content and purpose. In the metamodel, DocumentKind (and also ModelKind) are specializations of the class WorkProductKind.

OPEN also offers a wide range of SE-specific documents that can be consumed/produced by developer teams (e.g. Software Requirements Specification, CRC Cards or Software Architecture Document). Of these, there are various fragments derived from the metaclass DocumentKind that are relevant to teamwork, including Management Plan, Job Description (of each member or role in team), Organization Chart, Team Schedule, Project Schedule, Component Schedule, Statement of Work, Work Breakdown Structure and Status Report. However, there are other fragments that currently do not exist in the OPF repository. We have identified the following as necessary to support teamwork in a collaborative environment:

- “Team Structure Chart DocumentKind”
- “Team Policies DocumentKind”
- “Artifact Access Permissions DocumentKind”

4.4 TaskKinds

A task is a small-grained work unit that focuses on what must be done in order to achieve a given purpose. A task kind is a specific kind of task, characterized by its purpose within the project. It is a specialization (in the metamodel) of WorkUnitKind. A work unit is a job performed within a project. A work unit kind is a specific kind of work unit, characterized by its purpose within the project. It is specialized not only to TaskKind but also to TechniqueKind and WorkflowKind.

There are various pre-existing TaskKind fragments in the OPF repository that are relevant here. The following support team building and management of the development team:

- “Create an Appropriate Organizational Structure”: i.e. identify the Organizational Structure of the project in terms of Teams and major Roles.

- “Identify Project Roles and Responsibilities”: i.e. identify what Roles and Roles’ Responsibilities are required before considering which developers should play those Roles.
- “Project staffing”: involving interviewing and hiring prospective developers, and assigning them to Teams.
- “Choose project team”: involving the allocation of Roles to Teams and the allocation of team members to play these Roles.
- “Allocate Tasks”: i.e. allocate Tasks to Teams and Roles.
- “Choose Toolset”: involving choosing Teamwork tools
- “Review progress” (introduced in Dagher *et al.*, 2004)

Three new fragments are needed to fully support team management in collaborative situations. These are Specifying Team Structure TaskKind, Specifying Team Policies TaskKind and Managing Shared Artefacts TaskKind, the last of these having three subtasks:

- **“Identify Shared Artifacts”**: is the Task of identifying artifacts that can be accessed mutually by different teams or different members in a team.
- **“Allocate Shared Artifacts”**: is the Task of allocating these shared artifacts to Teams, Roles and/or Tasks.
- **“Specify Permissions to Shared Artifacts”**: is the Task of defining the permissions granted to each different Role/Team to access the artifact.

4.5 TechniqueKinds

A second specialization of WorkUnitKind is TechniqueKind. A technique is a small-grained work unit that focuses on how the given purpose may be achieved. A technique kind is a specific kind of technique, characterized by its purpose within the project.

OPEN already offers 3 techniques relevant to team building and management:

- “Team building”: Creation of teams should take into account the purpose of the team and the structure of team (OPEN offers a set of potential team structures). Team members should be selected for their wide-ranging skills and expertise (preferably across the whole lifecycle).
- “Role Assignment”: Locating team members to roles requires the identification of roles in team, an understanding of responsibilities of each role, and the assessment of skills of each member (both technical and inter-personal).
- “Group Problem Solving”, “Brainstorming” and “Workshops”: These are various techniques that a team can use to achieve a particular task.

Newly identified fragments are “Round robin”, “Role rotation”, “Conflict resolution”, “Team motivation” and “Team facilitation”. These, as with the other new fragments, are described in full detail at http://www.open.org.au/fragments/collaboration_team_fragments.pdf.

4.6 ActivityKind

An activity is a work flow that represents a continuous responsibility. An activity kind is a specific kind of activity, characterized by the area of expertise in which it occurs. ActivityKind (along with ProcessKind) are specializations of WorkflowKind, where a work flow is defined as a large-grained work unit that operates within a given area of expertise. A work flow kind is a specific kind of work flow, characterized by the area of expertise in which it occurs.

OPEN already offers “Project Management” ActivityKind, which contains a “Human Resource Management” sub-ActivityKind dealing with organizational structuring, staffing and task allocation. Dagher *et al.* (2004) introduced a new ActivityKind called “Team Building”. However, this does not deal with team management - a new fragment must therefore be identified and defined:

“Team Management” ActivityKind: ensures that every team in the project functions in an effective manner. The objectives of the Team Management ActivityKind include:

- To ensure that each team member understands their team role, responsibilities and position.
- To ensure that team members collaborate smoothly, with no intra-team or inter-teams unsolvable conflicts
- To ensure that the goals of each team are achieved or exceeded.

The Team Management ActivityKind involves Team Leaders (and/or Team Members) performing the following Tasks: “Choose project teams”, “Specify team structure”, “Specify team policies”, “Identify project roles and responsibilities”, “Manage shared artifacts” and “Review progress”.

5 SUMMARY AND CONCLUSIONS

In the context of (situational) method engineering, we have examined the support for teams and teamworks in one of the more extensive method fragment repositories – that of the OPEN Process Framework. Several gaps were identified in the context of supporting CSCW and software development collaborations. For each gap, a new method fragment has been proposed and fully defined and described. These new method fragments will allow a methodology focussed upon CSCW to be effectively constructed from the enhanced OPF fragment repository

6 ACKNOWLEDGMENTS

We wish to thank the Australian Research Council for funding this project.

References

- Avison, D.E. and Wood-Harper, A.T. 1991. ‘Information systems development research: an exploration of ideas in practice’. *The Computer Journal*, 34(2): 98-112.
- Brinkkemper, S. 1996. ‘Method engineering: engineering of information systems development methods and tools’. *Inf. Software Technol.*, 38(4): 275-280.
- Dagher, L., Henderson-Sellers, B. and Serour, M. 2004. ‘Roles and teams in a method engineering framework’. *CD-ROM Procs. EMCIS2004*, (eds. Z. Irani, S. Alshawi and O.D. Sarikas).
- Ellis, C.A., Gibbs, S.J. and G.L. Rein. 1991. ‘Groupware: Some issues and experiences’. *Comm. ACM*, 34(1): 39-58.
- Fayad, M.E., Tsai, W.T. and Fulghum, M.L. 1996. ‘Transition to object-oriented software development’. *Comm. ACM*, 39(2): 108-121.
- Firesmith, D.G. and Henderson-Sellers, B. 2002. *The OPEN Process Framework: An Introduction*. Addison-Wesley.
- Fitzgerald, B., Russo, N.L. and O’Kane, T. 2003. ‘Software development method tailoring at Motorola’. *Comm. ACM*, 46(4): 65-70.
- Gonzalez-Perez, C. and Henderson-Sellers, B. 2005. ‘Templates and resources in software development methodologies’. *J. Object Technol.*, 4(4): 173-190.
- Graham, I., Henderson-Sellers, B. and Younessi, H., 1997. *The OPEN Process Specification*. Addison-Wesley.

- Greenberg, S. 1991. *Computer-supported co-operative work and groupware*. Academic Press Ltd
- Henderson-Sellers, B. 2003. 'Method engineering for OO system development'. *Comm. ACM*, 46(10): 73-78.
- Henderson-Sellers, B. and Gonzalez-Perez, C. 2005. 'A comparison of four process metamodels and the creation of a new generic "standard"'. *Inf. Software Technol.*, 47: 49-65.
- Henderson-Sellers, B. and Hutchison, J. 2003. 'Usage-Centered Design (UCD) and the OPEN Process Framework (OPF)'. *Performance by Design. Procs. for USE2003, Second International Conference on Usage-Centered Design* (ed. L.L. Constantine), Ampersand Press, Rowley, MA, USA: 171-196.
- ISO/IEC. 2007. *Software Engineering – Metamodel for Development Methodologies*. ISO 24744, Geneva, Switzerland.
- Kumar, K. and Welke, R.J. 1992. 'Methodology engineering: a proposal for situation-specific methodology construction'. in *Challenges and Strategies for Research in Systems Development* (eds. W.W. Cotterman and J.A. Senn), J. Wiley: 257-269.
- Nguyen, V.P. and Henderson-Sellers, B. 2003. 'OPENPC: a tool to automate aspects of method engineering'. *Procs. ICSSSEA 2003*. Paris, France, Volume 5, 7pp.
- Odell, J.J. 1994. 'Power types'. *Journal of Object-Oriented Programming*, 7(2): 8-12.
- OMG. 2002. *Software Process Engineering Metamodel Specification*. Version 1.0, Object Management Group, formal/02-11-14, Nov 2002.
- Ralyté, J. and Rolland, C. 2001. 'An assembly process model for method engineering'. *Advanced Information Systems Engineering*, LNCS2068, Springer-Verlag, Berlin: 267-283.
- Standards Australia. 2004. *Australian Standard 4651-2004. Standard metamodel for software development methodologies*. ISBN 0 7337 6195 X, 72pp.
- ter Hofstede, A.H.M. and Verhoef, T.F. 1997. 'On the feasibility of situational method engineering'. *Information Systems*, 22(6/7): 401-422.
- Terzis, S. and Nixon, P. 1999. 'Building the next generation groupware: A survey of groupware and its impact on the virtual enterprise'. Technical report TCD-CS-1999-08, Computer Science Department, Trinity College, Dublin, Ireland.
- van Slooten, K. and Hodes, B. 1996. 'Characterizing IS development projects'. in *Procs. IFIP TC8 Working Conf. on Method Engineering: Principles of method construction and tool support* (eds. S. Brinkkemper, K. Lyytinen, R. Welke), Chapman&Hall, Great Britain: 29-44.
- Vessey, I. and Glass, R.L. 1994. 'Application-based methodologies: development by application domain'. *Information Systems Management*, Fall 1994.