Method Construction by Goal Analysis

C. Gonzalez-Perez[1], P. Giorgini[2] and B. Henderson-Sellers[3]

[1] University of Technology, Sydney, Department of Software Engineering, cesargon@verdewek.com
[2] University of Trento, Department Information and communication Technology, paolo.giorgini@unitn.it
[3] University of Technology, Sydney, Department of Software Engineering, brian@it.uts.edu.au

Abstract. Method engineering proposes the construction of methodologies by selecting method fragments from a repository and assembling then in an appropriate way. However, the rules by which the "optimal" method fragments are chosen are not clear, and such chores are usually done manually by an expert. This paper presents a goal analysis technique for the selection of the optimal method fragments from a repository, using backward reasoning to obtain the set of fragments that satisfy the desired goals with minimum effort. By using this technique, a methodologist can determine the goals that the organisation wants the methodology to satisfy, and then, preferably, rely on automated tools for the selection of the optimal solution.

1 Introduction

It is well accepted that no single software development methodology (or method; we will consider them here as synonyms) serves all purposes (Cockburn 2000). Different project, product and organisational characteristics call for different methodologies, which are often further tweaked or customised to fit the particular idiosyncrasies of its users (Bajec, Vavpotič, and Krisper 2007). One quick way to obtain a customised methodology is to adopt an existing one and change it as necessary. However, this entails significant risks since the methodologists making the changes are not necessarily aware of the interconnections and dependencies between different components of the methodology. The situational method engineering (SME) paradigm (Brinkkemper 1996; Henderson-Sellers, Serour, McBride, Gonzalez-Perez, and Dagher 2004b) offers a solution to this problem: instead of adopting an existing methodology and changing it as necessary, a custom methodology is created by selecting the appropriate method fragments from an existing repository and combining them appropriately. This approach is used in methodological frameworks such as OPF (Firesmith and Henderson-Sellers 2002), OOSPICE (Henderson-Sellers, Stallinger, and Lefever 2002) and FIPA (Cossentino, Gaglio, Garro, and Seidita 2007),

and is advocated in the recent ISO/IEC (2007) 24744 International Standard "Software Engineering Metamodel for Development Methodologies".

Despite an increasing and broadening interest in situational method engineering, some areas are still to be fully explored. For example, how are the method fragments to be selected from the repository? A complete methodology is likely to be composed of hundreds of method fragments, and each of these must be carefully chosen to (a) fit the purpose of the methodology being constructed and (b) be compatible with other method fragments. Usually, this task is performed by a methodologist, who uses his/her expert judgement to handcraft an "optimal" solution. This approach has a number of drawbacks. First of all, it can be extremely time consuming. Secondly, there is no way to demonstrate that the chosen collection of method fragments is best, i.e. no guarantee can be given on the quality of the result (other than that given by the trust on the methodologist's expertise). Typically, an organisation willing to adopt the method engineering paradigm will need to recruit a methodologist or hire a consultant to compose a methodology each time.

This paper presents a solution to these drawbacks in which a project manager will be able to create a profile of the methodology to be constructed in terms of the *goals that it must achieve*, and then use a goal analysis technique, ideally implemented by a tool, to extract the optimal combination of method fragments from the repository that fulfils the goals at minimum effort.

In the rest of this paper, Section 2 introduces some important concepts of method engineering; Section 3 explains the basic concepts of goal analysis and Section 4 its application to methodology construction.

2 Background for Situational Method Engineering

As explained above, the SME approach needs the existence of a method fragment repository. This repository is usually a database that contains method fragments of different kinds. Method fragments are self-contained, relatively independent specifications of some aspect of a methodology, such as a task to be performed, a technique that may be employed, a product that can be generated or a team that can be formed. Different kinds of method fragments have different properties: for example, task specifications have a purpose (that declares what the task intends to achieve) and a description (that specifies the steps that may be followed in order to achieve it); work product specifications, on the other hand, have a name (such as "Requirements Specification Document" or "Class Diagram") and a description. In turn, different kinds of method fragments are related to each other: for example, task specifications may be linked to the work products that they generate when executed.

The structure of the repository, i.e. the kinds of method fragments, their properties and the relationships between them, is usually given by a metamodel. A *metamodel* is a formal description of the concepts that can be used to construct a methodology and the relationships amongst them. Here, we will adopt the International Standard ISO/IEC (2007) 24744 "Software Engineering Metamodel for Develop-

ment Methodologies" (SEMDM). SEMDM defines 68 concrete classes, instances of which can potentially be stored in a method fragment repository. Not all the method fragment classes are relevant for this paper; we will concentrate on the following:

- PhaseKind. Specification of a managed timeframe within a project for which the objective is the transition between levels of abstraction. Phase kinds specify the "when" of a methodology, i.e. its temporal ordering and organisation.
- ProcessKind. Specification of a discrete, large-grained job performed within a project that operates within a given area of expertise. Process kinds specify the "what and why" of a methodology at an abstract level, i.e. the methodology's job structure.
- TaskKind. Specification of a small-grained job performed within a project that focuses on what must be done in order to achieve a given purpose. Task kinds specify the "what and why" of a methodology at a detailed level.
- TechniqueKind. Specification of a small-grained job performed within a project that focuses on how the given purpose may be achieved. Technique kinds specify the "how" of a methodology, i.e. the specific means of achieving the associated task.
- WorkProductKind. Specification of an artefact of interest for the project. Work product kinds specify what is created and consumed during a project.
- ActionKind. Specification of how a given task kind acts upon a particular work product kind.

These classes are interrelated in the following way (Fig. 1) i.e. each phase kind is composed of process kinds, which give "content" to it. The phase kind specifies when something must be done, while the associated process kinds define what to do. In turn, each process kind contains a number of task kinds, which flesh out and refine the process' purpose. In turn, each task kind may be associated to a number of technique kinds, since there is often a choice from several techniques, each of which can be used to achieve the goals of the same task, and different tasks can use the same technique. Finally, each task kind may be mapped to a number of work product kinds via action kinds. These mappings involve different action types: a task can *create*, *modify*, *delete* or *read* a work product. Typically, each task kind will read work products of some kinds and perhaps create a new work product of a different kind.

2.1 Sample Method Fragment Repository

Consider the following (simplified) example. Two phase kinds are defined in a repository: "System Definition" and "System Construction". The first is intended to be performed at the beginning of a project and defines the system to be built. The second is meant to be executed at the end of a project in order to construct the system previously defined. A number of process kinds are also defined: "Requirements Engineering", "Coding", "Acceptance Testing", "Quality Assurance" and "Process Improvement". Each of these process kinds specifies, from an abstract point of view, what can be done at some point in the project. Some of these process kinds are associated to the "System Definition" phase kind, some to "System Construction", and

some to both (Table 1). Then, some task kinds can be introduced, such as “Elicit requirements”, “Analyse requirements”, “Validate requirements”, “Develop service models” and “Determine work product defects” (Table 2). These task kinds, together with many more, would be associated to different process kinds. A number of technique kinds can also be introduced, such as “Prototyping”, “Peer reviewing” and “Threat modelling” (Table 3). These technique kinds would be mapped to task kinds in a many-to-many fashion. Finally, some work product kinds can be introduced into the repository, such as “Requirements Specification Document”, “Service Diagram”, “Source Program” and “Report” (Table 4). Each of these work product kinds would be associated to a number of task kinds with a particular action type; for example “Requirements Specification Document” can be mapped to “Document requirements” via an action kind with a “create” type and to “Develop service models” via a different action kind with a “read” type. In turn, “Service Diagram” can be mapped to “Develop service models” via an action kind with a “create” type.

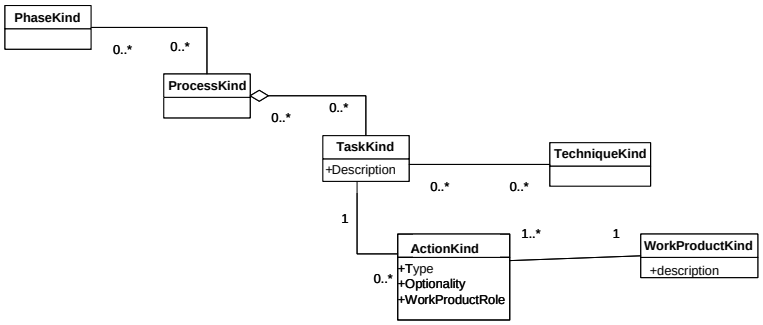

Fig. 1. Metamodel fragment (a subset of ISO/IEC 24744). Only relevant classes, attributes and associations are depicted. Here the diamond indicates a generic whole-part relationship.

Relevant life cycle models are created by instantiating the class TimeCycleKind from ISO/IEC 24744. This is a subtype of StageWithDurationKind (also the supertype of PhaseKind). Selection of the lifecycle is a stylistic decision much akin to the choice of architectural style for an software application. While it is possible that we can represent this selection process in terms of a soft goal, it is more likely that the choice will be made based on other, external factors and influences. (This topic of life cycle selection is a topic for future research - not discussed further here.)

Table 1 Sample process kinds.

Name	Mapped to phase kinds
Requirements Engineering	System Definition
Coding	System Construction
Acceptance Testing	System Construction
Quality Assurance	System Definition, System Construction
Process Improvement	System Definition, System Construction

Table 2 Sample task kinds.

Name	Mapped to process kinds
Elicit requirements	Requirements Engineering
Analyse requirements	Requirements Engineering
Validate requirements	Requirements Engineering
Document requirements	Requirements Engineering
Develop class models	High-Level Modelling
Develop service models	High-Level Modelling
Sketch user interface	High-Level Modelling
Develop interaction models	Detailed Modelling
Write code	Coding
Unit test class	Coding
Demonstrate the system	Acceptance Testing
Obtain stakeholder feedback	Acceptance Testing, Quality Assurance
Determine work product defects	Quality Assurance
Prepare defect report	Quality Assurance
Test build system	Quality Assurance

Table 3 Sample technique kinds.

Name	Mapped to task kinds
Prototyping	Develop service models, Sketch user interface
Text analysis	Analyse requirements, Develop class models
CRC cards	Develop class models
Peer reviewing	Validate requirements, Determine work product defects
Test-first development	Unit test class
In-house customer	Demonstrate the system, Obtain stakeholder feedback
Automated builds	Test build system
Threat modelling	Analyse requirements

Table 4 Sample work product kinds with action types.

Name	Mapped to task kinds	Action type
Stakeholders Statement	Elicit requirements	create
	Analyse requirements, Validate requirements	modify
	Document requirements	read
Requirements Specification Document	Document requirements	create
	Develop class models, Develop service models, Sketch user interface	read
Service Diagram	Develop service models	create
	Develop interaction models	modify
	Sketch user interface	read
User Interface Sketch	Sketch user interface	create
	Develop service models	modify
	Write code	read
Source Program	Write code	create
	Unit test class	modify
	Test build system	read
Report	Test build system, Determine work product defects, Prepare defect report	create

From the sample method fragments in Tables 1-4, it can be seen that the dependency network that can arise from the method fragments in a repository can be extremely intricate. For example, selecting the "High-Level Modelling" process kind would usually imply bringing along the "Develop service models" task kind, which "creates" a "Service Diagram" work product and "reads" a "Requirements Specification Document" work product. In order to provide the necessary input (i.e. a requirements specification document), we need to select a task kind that creates it, namely "Document Requirements". This task kind, in turn, may bring along the whole "Requirements Engineering" process kind together with additional task kinds.

Technique selection is usually more flexible, since a number of technique kinds are often available for each individual task kind. Which is selected depends only on the characteristics of the project (e.g. time or budget constraints), the product context (e.g. safe-criticality) and the organisation (e.g. culture and skills). Although we can assume that any of the technique kinds mapped to a given task kind is appropriate to achieve the task's purpose, the particular technique kinds that are chosen will likely influence overall project properties such as time consumed or defect injection rate as well as providing a different level of risk and associated costs. From this perspective, we can say that some techniques are better than others for some particular purposes.

2.2 Requirements for Method Construction

The design and construction of a methodology can be seen as any other engineering activity: some requirements are given and a suitable artefact that satisfies them must be developed. Therefore, we can assume that some requirements exist when methodologists face the task of constructing a methodology from a method fragment repository. These requirements can be described in terms of the capabilities and qualities of the intended outcome of the engineering effort, namely, the future methodology. In turn, method capabilities and qualities may refer to the kind of products that the method can construct, the type of projects used to tackle such activities and the characteristics of the organisations where these projects may take place. If we can characterise products, projects and organisations with measurable attributes, we will have a solid starting point on which requirements for method construction can be defined. These can be seen as defining the requirements *for the construction of the methodology* (as opposed to the requirements for the construction of the software application, which is the target of the software development project) (Ralyté 2002). Factors that influence these requirements are many, including organizational maturity level, skills set of development team members, type of domain (e.g. information systems, real-time control, e-business), project size, team size, level of criticality, interface style, level of resources allocated to project and whether or not the system is to be a distributed application (Nguyen and Henderson-Sellers 2003)

Table 5 shows a list of the attributes that we have identified for the purpose of illustration in this paper. We have only included attributes that may be directly affected by the choice of method fragments when constructing a methodology. We are aware that many other attributes (such as product correctness or readability) are also

of interest to software engineering, but they have been left out from this experiment since they are not likely to be directly affected by the choice of method fragments.

Table 5 Product, project and organisation attributes for method construction.

Area	Attribute	Description
Product	Reliability	The product must offer high reliability, i.e. its users will depend on it for critical operations.
	Changeability	The product will need to be changed, so it will need to offer the appropriate mechanisms to achieve this with ease.
	Usability	The product must be easy to use.
Project	Cost constraints	The project has cost constraints, so it must be completed at the lowest cost possible.
	Time constraints	The project has time constraints, so it must be completed in the shortest time possible.
	Staffing constraints	The project has staffing constraints, so it must be completed with the lowest possible number of staff.
	Visibility	The project needs high visibility, so all the work must be properly documented.
Organisation	Formal culture	The development team's culture promotes formal, high-ceremony work.
	Agile culture	The development team's culture promotes agile-style, low-ceremony work.
	Experience	The development team has got extensive experience in the kind of project and product to be developed.

3 Goal Analysis Concepts

In goal analysis, the final goal of each process step is considered from the point of view of a specific actor. There are three relevant reasoning techniques that are useful: means-end analysis, contributions analysis and AND/OR decomposition (Bresciani, Giorgini, Giunchiglia, and Mylopolous 2004). In means-end analysis, the following are performed iteratively until an acceptable solution is reached: **"**Describe the current state, the desired state (the goal) and the difference between the two; Select a promising procedure for enabling this change of state by using this identified difference between present and desired states; Apply the selected procedure and update the current state." (Henderson-Sellers, Giorgini, and Bresciani 2004a)

Contributions analysis helps to identify goals that may contribute towards the partial fulfilment of the final goal and is sometimes used as an alternative to means-end analysis, particularly useful for softgoals. Positive or negative influences towards attainment of the goal are identified and quantified on a (usually 5 point) Likert scale. In particular, contribution analysis has been shown to be very effective for soft goals used for eliciting non-functional (quality) requirements.

Finally, AND/OR decomposition changes a root goal into a finer goal structure i.e. a set of subgoals - either alternatives (OR decomposition) or additive (AND decomposition).

Goal analysis has been used in a number of ways to support software development e.g. in the design of systems, especially for documenting early requirements, as in the Tropos methodology (Bresciani *et al.,* 2004); in business process reengineering (Grau, Franch, and Maiden 2005); and in the support of ISO/IEC15504 assessments (Rifaut, 2005). Here, we present the first application of goal analysis to method construction in the context of method engineering.

4 Applying Goal Analysis to Method Construction

In order to use goal analysis for method construction, we need to determine how each of the method fragments in the sample repository affects each of the above listed attributes. For example, we can say that performing the Quality Assurance process (see Table 1) enhances product reliability. For each method fragment plus attribute pair, one of five possible values has been determined: strongly enhances, enhances, neutral, deteriorates and strongly deteriorates.

Table 6 shows these (non-neutral) mappings between method fragments and attributes. Please note that we are not claiming that these mappings are optimal or even correct; these are a sample collection of reasonable mappings for the purpose of this paper. A separate study would be necessary in order to determine how each method fragment in a production repository affects each attribute of interest.

Suppose we have two options for a Software Engineering Process (SEP) and each has several Tasks, each implemented by a Technique chosen from a list. The two options are shown graphically in Figure 2.

Table 6 Mappings between attributes and method fragments. For each mapping, a value is included indicating how the choice of the method fragment affects the attribute.

Attribute		Method Fragment		Value
Area	Name	Class	Name	
Product	Reliability	Process kind	Quality Assurance	strongly enhances
		Task kind	Unit test class	enhances
		Technique kind	Test-first development	enhances
			In-house customer	enhances
			Threat modelling	strongly enhances
	Changeability	Process kind	Configuration Management	enhances
		Task kind	Document requirements	enhances
	Usability	Process kind	Acceptance Testing	strongly enhances

Attribute		Method Fragment		Value
		Task kind	Demonstrate the system	enhances
			Obtain stakeholder feedback	strongly enhances
Project	Cost constraints	Phase kind	System Definition	deteriorates
		Process kind	Quality Assurance	deteriorates
			Process Improvement	deteriorates
	Time constraints	Phase kind	System Definition	deteriorates
		Process kind	Process Improvement	deteriorates
		Task kind	Unit test class	deteriorates
		Technique kind	Prototyping	deteriorates
			Automated builds	enhances
	Staffing constraints	Process kind	Quality Assurance	deteriorates
		Technique kind	Peer reviewing	deteriorates
			Pair programming	deteriorates
	Visibility	Task kind	Prepare defect report	enhances
			Prepare process quality report	enhances
Organisation	Formal culture	Phase kind	System Definition	strongly enhances
		Task kind	Measure process quality	enhances
	Agile culture	Phase kind	System Definition	strongly deteriorates
		Process kind	Process Improvement	deteriorates
		Task kind	Document requirements	deteriorates
			Elicit requirements	enhances
		Technique kind	In-house customer	enhances
			Test-first development	enhances
	Experience	Phase kind	System Definition	strongly enhances
		Process kind	Requirements Engineering	enhances
			Acceptance Testing	enhances
		Task kind	Elicit requirements	enhances
		Technique kind	Focus groups	strongly enhances
			Prototyping	strongly enhances
			Walkthroughs	enhances
			In-house customer	enhances

Looking at the Techniques we have a table (akin to Table 6 above) that links Techniques to impact factors (-ilities). The Techniques are labelled as X1-X6 where X1 = Test first; X2 = In house customer; X3 = Prototyping; X4 = Automated builds; X5 = Threat modelling; and X6 = Peer reviewing. Then the two processes can be described in terms of these terminal Techniques as:

SEP1 is (X1 or X2); (X3 or X4)
SEP2 is (X1 or X2); (X1 or X2 or X5 or X6)

We consider just two examples. The impact on the Reliability and of Agility factors:

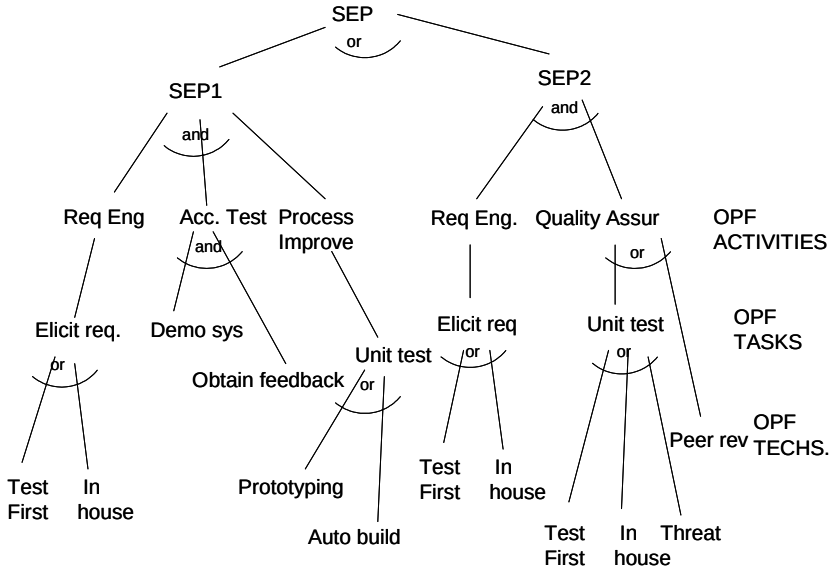

Fig. 2 Hierarchical tree depicting Activities, Tasks and Techniques for two hypothetical SEPs

1) reliability	
Test-first development (X1)	enhances (+)
In-house customer (X2)	enhances (+)
Prototyping (X3)	neutral (o)
Automated builds (X4)	deteriorates (-)
Threat modelling(X5)	strongly enhances (++)
Peer reviewing (X6)	strongly enhances (++)
2) agility	
Test-first development (X1)	enhances (+)
In-house customer (X2)	enhances (+)
Prototyping (X3)	deteriorates (-)
Automated builds (X4)	strongly deteriorates (--)
Threat modelling(X5)	strongly deteriorates (--)
Peer reviewing (X6	strongly enhances (++)

Then the impact is as follows:

OPTION	Reliability	Agility
SEP1 option 1 is X1; X3	+ / 0	+ / -

SEP1 option 2 is X1; X4	+ / -	+ / - -
SEP1 option 3 is X2; X3	+ / 0	+ / -
SEP1 option 4 is X2; X4	+ / -	+ / - -
SEP2 option 1 is X1; X1	+ / +	+ / +
SEP2 option 2 is X1; X2	+ / +	+ / +
SEP2 option 3 is X1; X5	+ / + +	+ / - -
SEP2 option 4 is X1; X6	+ / + +	+ / + +
SEP2 option 5 is X2; X1	+ / +	+ / +
SEP2 option 6 is X2; X2	+ / +	+ / +
SEP2 option 7 is X2; X5	+ / + +	+ / - -
SEP2 option 8 is X2; X6	+ / + +	+ / + +

We conclude that from a reliability viewpoint, the best choice would be SEP2, options 3, 4, 7 or 8. On the other hand, from an agility perspective, the best choice would be SEP2, option 4 or 6.

The above analysis is fully supported and automated in Tropos (Giorgini, Mylopoulous, and Sebastiani 2005). In particular, backward reasoning allows the analyst to search for possible method fragments from the repository that satisfy the desired goal. Moreover, by assigning a cost to each fragment, backward reasoning also produces the solution with the minimum cost.

5 Conclusions and Future Work

With the aim of creating a high quality software development methodology from those method fragments selected from an existing repository, we have examined a new idea based on goal analysis. Rather than select the elements of the methodology "top-down" by considering what seems reasonable in a particular situation using what might be termed "intuition" (the current approach in SME), we suggest that a more objective process can be created in which the main focus becomes the goal rather than the means of achieving that goal (the process element). The goal analysis approach proposed here permits the creation of an optimized methodology; importantly, one that is optimized for a particular characteristic such as reliability or agility. An hierarchical tree is constructed (Figure 2) and, for each element, we identify whether there is a positive or negative impact *for the chosen optimization characteristic*. We have demonstrated this approach with a simple example of a small tree in which fragments for activities, tasks and techniques from the OPF repository have been selected, considering the impacts on two different software engineering processes, SEP1 and SEP2 (Figure 2). That these processes have different optima under different evaluation criteria (here agility and reliability) suggests that this approach is worthy of further investigation including practical trials in industry and the development of a prototype support tool. We are currently planning such industry trials within the Italian ministry funded project MEnSA (http://www.mensa-project.org) project and anticipate building appropriate support tools in due course.

Acknowledgments

We wish to thank for financial support both the Australian Research Council and the Italian ministry for research through its PRIN-MEnSA project.

References

Bajec, M., Vavpotič, D. and Krisper, M. (2007) Practice-driven approach for creating project-specific software development methods. Inf. Software Technol. 49, 345-365.

Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F. and Mylopolous, J. (2004) Tropos: an agent-oriented software development methodology. Autonomous Agents and Multi-Agent Systems 8(3), 203-236.

Brinkkemper, S. (1996) Method engineering: engineering of information systems development methods and tools. Inf. Software Technol. 38(4), 275-280.

Cockburn, A.S. (2000) Selecting a project's methodology. IEEE Software 17(4), 64-71.

Cossentino, M., Gaglio, S., Garro, A. and Seidita, V. (2007) Method fragments for agent design methodologies: from standardization to research. Int. J. Agent-Oriented Software Eng. 1(1), 91-121

Firesmith, D.G. and Henderson-Sellers, B. (2002) *The OPEN Process Framework.* Addison-Wesley, London.

Giorgini P., Mylopoulous J. and Sebastiani R. (2005). Goal-oriented requirements analysis and reasoning in the Tropos methodology. Eng. Appl. Artific. Intell. 18(2), 159-171.

Grau, G., Franch, X. and Maiden, N.A.M. (2005) A goal-based round-trip method for system development. In: *Procs. 11th International Conference on Requirements Engineering: Foundations for Software Quality (REFSQ'05)*, pp. 67-82.

Henderson-Sellers, B., Stallinger, F. and Lefever, B. (2002) Bridging the gap from process modelling to process assessment: the OOSPICE process specification for component-based software engineering. In: *Procs. 28th EUROMICRO Conference. Dortmund, Germany, 4-6 September 2002.* IEEE Computer Society: Los Alamos, CA, USA, pp. 324-331.

Henderson-Sellers, B., Giorgini, P. and Bresciani, P. (2004a) Enhancing Agent OPEN with concepts used in the Tropos methodology. In: A. Omicini, P. Pettra and J. Pitt (Eds.), *Engineering Societies in the Agents World IV. 4th International Workshop, ESAW 2003*. LNAI 3071, Springer-Verlag, Berlin, pp. 328-345.

Henderson-Sellers, B., Serour, M. McBride, T. Gonzalez-Perez, C. and Dagher, L. (2004b) Process construction and customization. J. Universal Computer Science. 10(4), 326-358.

ISO/IEC (2007). *Software Engineering Metamodel for Development Methodologies*. ISO/IEC 24744, International Organization for Standardization, Geneva.

Nguyen, V.P. and Henderson-Sellers, B. (2003) Towards automated support for method engineering with the OPEN Process Framework. In: M.H Hamza (Ed.), *Procs. Seventh IASTED International Conference on Software Engineering and Applications.* ACTA Press, Anaheim, CA, USA, pp. 691-696.

Ralyté, J. (2002) Requirements definition for the situational method engineering. In: C. Rolland, S. Brinkkemper and M. Saeki (Eds.), *Engineering Information Systems in the Internet Context.* Kluwer Academic Publishers, Boston, USA, pp. 127-152.

Rifaut, A. (2005) Goal-driven requirements engineering for supporting the ISO 15504 assessment process. In: I. Richardson, P. Abrahamsson and R. Messnarz (Eds.), *Software Process Improvement. 12th European Conf., EuroSPI 2005*, LNCS 3792, Springer, pp. 151-162