Malaysian Journal of Computer Science, Vol. 14 No. 2, December 2001, pp. 26-37

PROCESS MODELING LANGUAGES: A LITERATURE REVIEW

Kamal Zuhairi Zamli
School of Electrical and Electronics Engineering
Universiti Sains Malaysia
14300 Nibong Tebal
Pulau Pinang
email: kamal_zamli@hotmail.com

ABSTRACT

A software process is defined as a sequence of steps that must be carried out by the human agents to pursue the goals of software engineering. In order to achieve a precise specification of what these steps actually are, a software process can be represented using a process modeling language (PML). A representation of the software process in a PML is called a process model. Through a process enactment mechanism, which allows execution of the process model, a software process can automate, guide, and enforce software engineering practices and policies. These technologies are often collected together into what are referred to as process-centred environments or process centred software engineering environments (PSEE). Over the past 12 years, there have been many PSEEs (and PMLs) developed. However, the use of PSEEs and PMLs are not widespread. We envisage that these technologies provide a vital support for software engineering in the future. This article surveys the current state of the art of the PMLs including the second generation PMLs, which have not been included in other surveys in the literature, and discusses the possible research agenda for future work in the area.

Keywords: ***Software Process, Process Modeling Languages, Software Engineering Support***

1.0 INTRODUCTION

A software process is defined as a sequence of steps that must be carried out by the human agents to pursue the goals of software engineering. In order to achieve a precise specification of what these steps actually are, a software process can be represented using a process modeling language (PML). A representation of the software process in a PML is called a process model.

A process model must constitute four important process elements namely activity, products, roles, and tools [15]. Activity consists of one or more process steps [1], which may run in parallel with other process steps. Products are artifacts which are normally under configuration control. Roles describe the responsibility and rights of the human who performs the process steps or who is in charge of the human agents. It should be noted that a person may play more than one role and a role may be associated with several people. Tools covers any tools used in the production of the software including compilers, debuggers, editors and even CASE tools.

Through a process enactment mechanism which allows execution of the process model, a software process can automate, guide, and enforce software engineering practices and policies. These technologies are often collected together into what are referred to as process-centred environments or process centred software engineering environments (PSEE).

Over the past 12 years, there have been many PSEEs (and PMLs) developed. However, the use of PSEEs and PMLs are not widespread. We envisage that these technologies provide a vital support for software engineering in the future. This article surveys the current state of the art of the PMLs including the second generation [2] PMLs, which have not been included in other surveys in the literature (such as [12], [20] and [23]), and discusses possible research

[1] Process step is an atomic action of a process that has no externally visible substructure (i.e. the smallest possible representation in a software process).

[2] Generally speaking, a second generation PML, the distinction first noted by Sutton and Osterweil [38], is one published after 1996.

agenda for future work in the area.

2.0 CLASSIFICATION OF PMLs

The best-known classification is probably that of Ambriola *et al* [1]. Ambriola *et al* classifies PMLs into three categories namely Process Specification Languages (PSL), Process Design Languages (PDL) and Process Implementation Languages (PIL). Process specification languages, typically formal languages, are languages that are in use in the specification phase of the process lifecycle. Process design languages are languages that are in use in the design phase of the process lifecycle. Process implementation languages are languages that are in use in the implementation phase of the process lifecycle.

Clearly, the classification according to the process lifecycle, helps to distinguish the lifecycle each PML supports. Arguably, it is difficult to classify existing PMLs into well-defined groups since some PMLs, for instance, being process specification languages yet support process enactment. In some cases, some researchers are using more than one PMLs in different phases of the process lifecycle in the modeling of software process.

In our previous work [41], we proposed a simpler classification based on process enactment support. While Huff [23] has also proposed a similar classification, her work suffers from the inclusion of implementation details of each PML, which makes it difficult to classify the second generation PMLs. In summary, our classification recognized the PML as being:

- Non-enactable
- Simulated
- Enactable

Non-enactable PMLs support only process modeling and understanding and not process enactment. Simulated PMLs enable a high-level simulation of the process model, which normally aids in the design of new software processes, but does not provide fine-grained guidance or control of the software process. Enactable PMLs permit the process model specified using that PML to be enacted to actively guide or even to control a software process.

3.0 AREAS OF SUPPORT FOR NON-ENACTABLE, ENACTABLE AND SIMULATED PMLs

There are five important areas that a PML must support namely, modeling support, enactment support, evaluation support, evolution support and human dimension support [41]. Modeling support refers to the ability of the PMLs to express the precise specification of steps involved in the software process. Enactment support refers to whether the syntax of the PML has an underlying executable semantics, allowing it to be executed. Evaluation support refers to whether the PML has some support for evaluation of the process model either quantitatively or qualitatively. Evolution support relates to the ability of the process model through its PML syntactic support to recover from its current state of enactment after some changes to the process model. Human dimension support is a new area of support for PML, which considers augmentation of some human dimension issues in the design of a PML [41].

Table 1: Classification of PMLs and their areas of support

Support Areas	**Process Modeling Languages**		
	Non-Enactable	Simulated	Enactable
Modeling Support	√	√	√
Enactment Support	x	x	√
Evaluation Support	x	√	√
Evolution Support	x	x	√
Human Dimension Support	x	x	√

Table 1 tabulates the comparison in terms of support areas among the three classifications of PMLs. It can be seen that enactable PMLs cover a wider spectrum of support areas. Simulated PMLs tend to cover modeling support and evaluation support. Non-enactable PMLs, on the other hand, mainly concentrate on modeling aspects of software process. The general common support area of the PMLs is mainly the modeling support.

Having established the common ground of non-enactable, enactable and simulated PMLs, we would proceed to survey the current state of the art of PMLs and relate them in terms of their classification.

4.0 SURVEY OF EXISTING PMLs

This section uses the classification derived from our previous work, which is discussed in Section 2.0 and 3.0, to survey the state of the art of PMLs.

4.1 Non-Enactable PMLs

Table 2 lists PMLs in the non-enactable group in chronological order of major publications. Entry Condition, Tasks, Verification activities and Exit criteria (ETVX) existed long before interest to support software process started. ETVX was started by IBM in the early 1980s to write the procedure and quality manuals. A software process model is expressed as a set of interdependent activities, each of which has four attributes: entry (E) criteria, task to be accomplished (T), task to be validated (V), and exit (X) criteria. The software process model indicates the relationships and flow among the four attributes of an activity and between activities. Early publication such as that of Humprey [25] uses an ETVX variation to model software process.

Table 2: Non-enactable PMLs

Year	PML	PSEE
1989	ETVX	-
1994	E3	-
1994	Limbo	Oikos
1994	Base Model	PADM
1999	UML	-
2000	UPM	-

E3 [3] is an object-oriented language designed to aid the understanding of software processes. It provides four views of a software process model namely: inheritance view, task view, functional decomposition view and informational perspective view. A software process is modeled in each view using pre-defined classes and relations that are denoted by graphical symbols.

Limbo is a specification language, which has been created for reactive systems. It is the PML used during the specification and design phase of the process lifecycle of OIKOS [31]. OIKOS is derived from an ancient Greek word to mean the environment for software development. A software process model consists of a hierarchy of entity structure that includes process, office, environment, desk, role, service instance and coordinator. Entities can interact and influence each other by message passing controlled by coordinator agents. Limbo is actually a compatible specification language for Pate, an enactable PML used during the implementation phase of the process lifecycle. Pate is discussed in an enactable PML group.

Base Model (BM) is a specification language for the Process Analysis and Design Methodology (PADM) [7], formerly known as Integrated Process Support Environment (IPSE). It is used during the specification and design phases of the process lifecycle of PADM. BM employs temporal logic to allow formal representation of processes. It supports gradual refinement from high-level specification to detailed design. Actually, BM is a specification language for Process Management Language (also called Process Wise Integrator PML), the implementation language for PADM. Process Wise Integrator PML is described in an enactable PML group.

Unified Modeling Language (UML) [35] is a language that is used for object-oriented software design. Jager [27] reported the use of UML class diagrams with modified stereotype constructs to aid the construction of dynamic task net (DYNAMITE) model. In this case, UML is used as the process design language to support construction of software process model for DYNAMITE. DYNAMITE uses a separate PML called Programmed Graph Rewriting System (PROGRESS), which is discussed in the enactable PML group.

In other works, Franch and Ribo [17] used the extended meta-model of UML to model the static part of software processes, essentially a conceptual model that defines the elements participating in a software process model. This work is performed as a basis of further work in designing Process Oriented Modeling and Enactment of Software Developments (PROMENADE), which is discussed in the enactable PML group.

Unified Process Modeling Language (UPM) [32] is a PML that shares some of its constructs with UML. It is developed by the Object Management Group (OMG) specifically to aid understanding of software processes. The modeling of software processes is done almost exclusively using the UML-like activity diagram and some extended notations.

4.2 Simulated PMLs

Table 3 lists PMLs in the simulated group in chronological order of major publications. The Statemate system is based on the work of David Harel on state-charts. State-charts are enhancements to traditional state transition diagrams. Statemate system was originally developed to support the construction of complex reactive systems. Statemate provides three graphical languages to represent structural, functional, and behavioural views of complex reactive systems. Humprey and Kellner [26] reported a successful attempt of representing behavioural modeling perspectives of software process using state-charts and simulated through Statemate tools.

Table 3: Simulated PMLs

Year	PML	PSEE
1989	Statemate	-
1994	Socca	-
2000	SDL	-

Specification of Coordinated and Cooperative Activities (Socca) [16] is a specification language based on PARADIGM, which is the specification language for specification of coordinated parallel processes. Socca describes a software process model in three different perspectives, namely data perspective, behaviour perspective and process perspective. It describes its data perspective using class diagram. Behaviour perspective is described using internal and external state transition diagram. The communication between each internal and external state transition diagram, which constitutes process perspective, is expressed using PARADIGM.

Specification and Design Language (SDL) is a formal language, with both graphical and textual representations, for specifying and describing real-time communication systems consisting of concurrent process. The top abstraction level of a system is hierarchically structured as a set of interrelated diagrams mainly consisting of blocks whose behaviours are controlled by the system specification. Typically, blocks contain one or more SDL processes, an instance of SDL process types, which are allowed to communicate among themselves and the system environment. In the work by Podnar *et al* [33], a software maintenance process for a large telecommunication company is modeled using SDL. In the work, software process elements consisting of role, artifact and tools are modeled as SDL process type. An activity is modeled with a particular role in SDL process type. Through SDL Development Tool (SDT), the role, artifact, tools, and activity are instantiated to process instances, which allow simulation and verification of the software maintenance process produced.

4.3 Enactable PMLs

Table 4 lists all PML in the enactable group in chronological order of major publications. Marvel Strategy Language (MSL) is the PML for Marvel [29]. The software process model in Marvel is an extensible collection of rules for the process steps with preconditions and post conditions, stored in an object-oriented database. Marvel interprets its rules using forward and backward chaining. Forward chaining lets Marvel perform opportunistic execution of process steps as soon as their pre-condition are satisfied as a result of prior steps performed. Backward chaining helps Marvel find the process steps whose post-conditions satisfy the pre-condition of other process steps that have been activated.

Table 4: Enactable PMLs

Year	PML	PSEE
1988	MSL	Marvel
1988	Grapple	Grapple
1989	HFSP	-
1990	Melmac	Melmac
1994	Slang	Spade
1994	Pate	Oikos
1994	PWI PML	PADM
1994	Merlin/PML	Merlin
1994	Spell	Epos
1994	MASP/DL	Alf
1994	Peace/PDL	Peace
1994	Adele -Tempo	Adele -Tempo
1995	APPL/A	Arcadia
1996	PROGRESS	Dynamite
1997	JIL	Julia
1997	CSPL	CSPL
1998	Little JIL	Juliette
1998	EVPL	Serendipity
1998	APEL	-
2000	PROMENADE	-

Grapple [24] uses the artificial intelligent planning paradigm, an artificial intelligent approach to a theory of action to model the software process. Processes are formally defined in a hierarchy using plan operators with multiple levels of abstraction. Each operator has some precondition defining the state that must hold in order for action to be legal, and a set of effects that defines the state changes that result from performing the action. A plan places stress more on goals rather than activities. Grapple combines plan generation with plan recognition. Plan generation automatically executes process steps to achieve a goal while plan recognition attaches steps executed by the process performer to the current set of plans.

Hierarchical and Functional Software Process (HFSP) [30] is based on attribute grammars and functional programming. In HFSP, a software process is described as a collection of activities, which are characterised by their input and output relationship and defined as mathematical functions. Complex relationships can be decomposed into sub-activities together with the definition of their input and output. The enactment mechanism in HFSP provides activity scheduling, activity execution management, tool invocation, access to input and output of the software process model, and user interaction. Activity scheduling allows concurrent activities to execute when their input becomes available.

Melmac [14] is the pioneering system that uses Petri Nets extension called FUNSOFT nets as its base. FUNSOFT nets represent the general structure of the software process. From within the net, Melmac denoted a number of views including:

- Object Type and Activity View
- Process View
- Project Management View

The Object Type and Activity View describe the activity, types and tools involved. The Process View defines the data flow constraints between the software development activities. The Project Management View specifies the roles and temporal constraint for each activity.

Slang is a PML for SPADE [4]. Like Melmac, Slang also uses a Petri Net extension, called ER nets, as its base. Nevertheless, unlike Melmac, Slang process model can be hierarchically structured as a set of activities, each is described by a net that may include invocation of other (sub) activities. An activity is the Slang modularisation

facility. Since Slang is based on high-level Petri Nets, process data are represented as tokens. Slang is integrated with object-oriented database O2, which serves as repository for process models.

Pate is an implementation language for OIKOS [31]. The bulk of modeling software process is actually done by its specification language Limbo, discussed as non-enactable PML. Step-wise refinement of the Limbo specification produces Pate executable code.

Process Management Language (PWI PML) is the class based implementation language for PADM [7] that allows its user to write process program. PADM provides three views of the software process models:

- The User Model
- The Environment Model
- The Application Model.

The User Model describes the interaction between roles, tasks and artifacts. The Environment Model shows the infrastructure support for process enactment. The Application Model depicts the actual executable software process model. PWI PML is used in the Application Model where software process is modeled as a set of interacting "system" roles, which may be user role or user tasks. Each system role communicates with each other using message passing.

Merlin PML is a Prolog-like language to support process enactment for Merlin [28]. Process modeling in Merlin is done in 2 levels: process design and process enactment. Process design is supported using a graphical notation called Entity Relationship Models and State-charts Combined for Advanced process Engineering (ESCAPE). State-charts, which are enhanced state transition diagrams, are used to specify the behaviour of the process models. ESCAPE design is automatically mapped to Merlin PML to support process enactment.

Spell is the PML for Expert System for Program and System Development (EPOS) [10]. Software process is modeled as a typed network of chained and decomposed tasks, an instantiation of task type. These are linked to other tasks, products, tools and roles. Tasks interact with each other and with tools and humans. A task type in Spell defines input output parameter for the task, decomposition that defines task breakdown, static and dynamic pre-conditions and post-conditions around a script code to be executed using textual Prolog-like notation. Spell is integrated with object-oriented database called EPOSDB.

Model for Assisted Software Process Description Language (MASP/DL) is the PML for Alf [8]. MASP/DL describes a generic MASP software process model. A generic MASP software process model is composed of software process fragments consisting of:

- An Entity Relationship Attribute
- A set of operator types
- A set of rules of type event-condition-action
- A set of ordering constraints
- Characteristics

An Entity Relationship Attribute describes all the data used in the fragments. Operator types allow abstraction of tools and pre/post-conditions. Rules of type event-condition-action express how to react to some predefined events. Ordering constraints express how operator invocation can occur with respect of precedence rules, simultaneity constraints between operators and condition on object instances. Characteristics are expressions used as invariants and/or as objectives. A generic MASP model can be tailored and instantiated for a specific project or specific organisation.

Process Centered Enactable and Adaptable Computer Aided Environment (Peace) [2] adopted a goal-oriented approach, considering goal rather than activities as the fundamental concepts of modeling. A Peace software process model is a set of Process Model Fragments (PMF) similar to Alf. Peace/PDL is a PML, which describes the generic Peace PMF software process model. PMF specification is described in terms of object model using data definition language called PCTE Object Management System (OMS) and an operator model. The operator model formally describes the process steps associated with a PMF in terms of its name, input and output, its intrinsic role,

pre/post-conditions and its incoming and outgoing events. An improvement of Peace called Peace+ extends enactment support for distributed process model and support for process evolution.

Adele-Tempo [6] provides two enactable PMLs. The first language, Adele, is an object-oriented language for manipulating database contents in Adele database. Data, in Adele database, is organised as objects with relationships. Adele PML provides support for modeling the entity relationship data model with trigger support mechanism, which automatically responses to prescribed database operation events. A trigger program is actually executed each time the corresponding event is true. Artifacts and workspace support is described at this level with sets of events, actions, conditions and coupling associated with database operation. As Adele PML was found to be difficult to understand, language and trigger execution is difficult to control, the second PML called Tempo was developed on top of Adele PML. Tempo defines a process model based on the concepts of role and connection. A role, if granted permission, can dynamically redefine the static and behavioural property of an object. A connection expresses how the process collaborates.

APPL/A [37] is one of the pioneering PML. It runs under a PSEE called Arcadia. APPL/A extends the ADA programming language with shared persistence relations, concurrent and reactive triggers, optionally enforceable predicates and five transactions-like composite statements. As with ADA, APPL/A abstractions and modularisations are obtained via procedures and packages.

In Dynamic Task Nets (DYNAMITE) [22], a software process is modeled via hierarchies of tasks that are connected to various kinds of relationships (control flow, data flow and feedback). State transition diagrams are used to model the dynamic behaviour of a task. As the name suggests, tasks can be dynamically created. Dynamic Task Nets are formally defined in an executable specification language called Programmed Graph Rewriting System (PROGRESS), which is based on graph rewriting systems.

JIL [38], the APPL/A successor, which runs on a PSEE called Julia, emphasises two elements:

- Process Steps
- Control Paradigms and Exception Handling

A JIL step represents a step in a software process. In JIL, a step in the software process provides abstraction for individual process steps. JIL control paradigms, recognise proactive and reactive control, integration of pre/post-condition and ability to loosely organise an incomplete process model. Proactive control allows one or more process steps to be imperatively programmed in a JIL step. Reactive control reacts on stimuli or events by executing one or more process steps. JIL reactive control recognizes four types of events related to product state, process state, resource state and exceptions.

Little JIL [40] is a scaled down version of JIL. However, unlike JIL, Little JIL is a visual language. Some features from JIL, such as events related to product states and data type model, are dropped in Little JIL. A software process model is a tree of steps whose leaves represent the smallest specified unit of work and whose structure represents the way in which this work will be coordinated. Little JIL provides four kinds of non-leaf step kinds called sequential, try, parallel and choice, to capture process steps ordering.

Concurrent Software Process Language (CSPL) [9] is a PML that shares most of its syntax with object-oriented ADA95. CSPL adopts a unique approach by integrating object-oriented ADA95 for its modeling support with UNIX shell scripts for its enactment support. Besides the usual ADA95 syntax, CSPL adds special language constructs to model software processes consisting of:

- Work assignment statement
- Communication-related statements
- Role unit
- Tool unit
- Relation unit

The work assignment statement allows assignment of work to multiple developers. Communication-related statements allow synchronisation of tasks with other tasks. The role unit defines the mapping of a role to the

developers. The tool unit specifies the tools needed to complete the tasks. The relation unit allows modeling of dependency between artifacts.

Extended Visual Planning Language (EVPL) [21] is the PML derived from the Swenson's Visual Planning Language (VPL) [37]. VPL is originally developed to model the process and plan work for multiple collaborating users in the CSCW community. EVPL extends VPL with new visual notation such as identifiers for process stages, representation of role, artifact and tools, usage connection between process stages and roles, and various usage annotations to indicate work context. Software processes are modeled in EVPL as work plans with a retrievable history of work. A work plan consists of various stages with indication of roles, artifacts and tools used. EVPL's work plan captures the work context for the tasks in the form of tools and artifacts used in each stage, and communication and coordination needed between stage roles.

Abstract Process Engine Language (APEL) [13] is a graphical high-level language designed to model a software process and can be enacted by two commercial process engines, ADELE [6] and Process Weaver [18]. In APEL, software process is described using Object Management Techniques-like diagrams, data flows, control flows, workspaces and cooperation and roles, and state transition diagrams. APEL provides support for Goal Question Metrics (GQM) model from the Quality Improvement Paradigm [5]. GQM is an approach for goal-oriented measurement in software projects which support measurement of products and processes for further process improvements. The GQM plan actually consists of a goal, questions related to the process model achieving the goal, and metrics to quantify the questions.

Process Oriented Modeling and Enactment of Software Developments (PROMENADE) [34] concepts and techniques have been influenced by UML. In PROMENADE, UML meta-model is extended to allow modeling of both the static and the dynamic aspects of software processes. The static aspect of software processes is given by a conceptual model that defines the elements that participate in a software process model. The dynamic aspect of software processes consists of the way in which a model is enacted, such as the ordering of tasks. The process of building a software process model in PROMENADE mainly consists of creating instances of the extended UML meta-model class. In addition, PROMENADE introduces both proactive control-flow (enactment of some actions according to a pre-establish plan) and reactive control-flow (enactment of some actions in response to events) mechanisms as means to model the dynamic behaviour of the software processes.

5.0 DISCUSSION

The survey in Section 4.0 raises an old issue of whether to have a single large PML or more than one compatible PMLs for different phases of the process lifecycle [11]. The vast majority of the work has opted for a single PML supporting the design and implementation phase of the process lifecycle. As the survey illustrates, there has also been work that uses more than one PMLs in different phases of the process lifecycle. It seems that PROMENADE is the only language that attempts to cover all the phases of the process lifecycle in a single language. The question remains unanswered since no single PML has claimed dominance for the modeling of software process.

While APEL has already integrated a form of measurement of the software process through GQM, which is an informal experienced-based measurement, PMLs are still lacking in support for empirical evaluation such as software metrics. Software metrics would allow effective quantitative comparison, for instance, of two software processes, which are performing the same tasks. In fact, this finding is also consistent with the views of Fuggetta [19] and Zamli and Lee [41].

Besides, the work of EVPL, PMLs have not fully exploited experience from other similar research areas such as Computer Supported Cooperative Work (CSCW) and Workflow Management Systems (WFMS). In fact, we are currently investigating an enactable PML, called VRPML, which exploits research in virtual environment particularly Collaborative Virtual Environment (CVE), a subset of CSCW. Our work mainly concentrates on improving the human dimension issues at the PML level.

While outsourcing of software is now common in the industry, it can be seen that PMLs are still lacking in support for modeling and enactment of distributed software process across organisational boundary. This raises issues such as security and rights to access shared artifacts, which could be explored.

Application of the ideas in software process to open source development [36], such as Linux-like development, is another exciting possible area of research. One reason is that open source development seems to need some notion of guidance such as that of conventional software development process.

6.0 CONCLUSION

In summary, this article presents a survey of PMLs which highlights the current state of the art of the technology and discusses the possible areas of research in the area of software process. The survey has also incorporated the second generation PMLs, which previously have not been included in other surveys in the literature.

The contribution of second generation PMLs particularly PROMENADE, APEL, EVPL suggest areas of research worth exploring such as the use of one or many PMLs, the application of software metrics and possible integration with research in computer supported cooperative work (CSCW) and Workflow Management Systems (WFMS). Other possible research areas also include open source and cross-organisational boundary developments.

REFERENCES

[1] V. Ambriola, R. Conradi and A. Fuggetta, "Assessing Process-Centered Software Engineering Environments". *ACM Transactions on Software Engineering and Methodology*, Vol. 6, No. 3, July 1997, pp. 283-328.

[2] S. Arbaoui and F. Oquendo, "PEACE: Goal-Oriented Logic-Based Formalism for Process Modeling", in A. Finkelstein, J. Kramer and B. Nuseibeh, eds., *Software Process Modelling and Technology*. Research Studies Press, Taunton, England, 1994, pp. 249-278.

[3] M. Baldi, S. Gai, M. L. Jaccheri and P. Lago, "Object-Oriented Software Process Model Design in E3", in A. Finkelstein, J. Kramer and B. Nuseibeh, eds., *Software Process Modelling and Technology.* Research Studies Press, Taunton, England, 1994, pp. 279-290.

[4] S. Bandinelli, A. Fuggetta, C. Ghezzi and L. Lavazza, "SPADE: An Environment for Software Process Analysis, Design, and Enactment", in A. Finkelstein, J. Kramer and B. Nuseibeh, eds., *Software Process Modelling and Technology*. Research Studies Press, Taunton, England, 1994, pp. 223-247.

[5] V. R. Basili and H. D. Rombach, "The TAME Approach: Towards Improvement-Oriented Software Environments". *IEEE Transactions on Software Engineering*, Vol. 14, No. 6, June 1988, pp. 758-773.

[6] N. Belkhatir, J. Estublier and W. Melo, "ADELE-TEMPO: An Environment to Support Process Modelling and Enaction", in A.Finkelstein, J. Kramer and B. Nuseibeh, eds., *Software Process Modelling and Technology.* Research Studies Press, Taunton, England, 1994, pp. 187-222.

[7] R. F. Bruynooghe, R. M. Greenwood, I. Robertson, J. Sa and B. C. Warboys, "PADM: Towards a Total Process Modelling System", in A. Finkelstein, J. Kramer and B. Nuseibeh, eds., *Software Process Modelling and Technology*. Research Studies Press, Taunton, England, 1994, pp. 293-334.

[8] G. Canals, N. Boudjlida, J. C. Derniame, C. Godart and J. Lonchamp, "ALF: A Framework for Building Process-Centered Software Engineering Environments", in A. Finkelstein, J. Kramer and B. Nuseibeh, eds., *Software Process Modelling and Technology*. Research Studies Press, Taunton, England, 1994, pp. 153-185.

[9] J. J. Chen, "CSPL: An Ada95-Like, Unix-Based Process Environment". *IEEE Transactions on Software Engineering*, Vol. 23, No. 3, March 1997, pp. 171-184.

[10] R. Conradi, M. Hagaseth, J. O. Larsen, M. N. Nguyen, B. P. Munch, P. H. Westby, W. Zhu, M. L. Jaccheri and C. Liu, "EPOS: Object Oriented Cooperative Process Modeling", in A. Finkelstein, J. Kramer and B. Nuseibeh, eds., *Software Process Modelling and Technology.* Research Studies Press, Taunton, England, 1994, pp. 33-64.

[11] R. Conradi and C. Liu, "Process Modelling Languages: One or Many?", in W. Schafer, ed., *Proceedings of the 4th European Workshop on Software Process Technology (EWSPT-4)*, Noordwijkerhout, The Netherlands. Lecture Notes in Computer Science, Vol. 913, Springer, April 1995.

[12] R. Conradi and M. J. Jaccheri, "Process Modelling Languages", in J. C. Derniame, B. A. Kaba and D. Wastell, eds., *Software Process: Principles, Methodology and Technology.* Lecture Notes in Computer Science, Vol. 1500, Springer, 1999, pp. 27-51.

[13] S. Dami, J. Estublier and M. Amiour, "APEL: A Graphical Yet Executable Formalism for Process Modeling". *Automated Software Engineering*, Vol. 5, No. 1, January 1998, pp. 61-96.

[14] W. Deiters and V. Gruhn, "Managing Software Processes in the Environment MELMAC", in *Proceedings of the 4th ACM SIGSOFT Symposium on Software Development Environment*, 1990, pp. 193-205.

[15] J. C. Derniame, B. A. Kaba and D. Wastell, eds. *Software Process:Principles, Methodology and Technology*. Lecture Notes in Computer Science, Vol. 1500, Springer, 1999.

[16] G. Engels and L. Greonewegen, "SOCCA: Specification of Coordinated and Cooperative Activities", in A. Finkelstein, J. Kramer and B. Nuseibeh, eds. *Software Process Modelling and Technology*. Research Studies Press, Taunton, England, 1994, pp. 71-102.

[17] X. Franch and J.M. Ribo, "Using UML for Modelling the Static Part of a Software Process", in *Proceedings of 2nd Unified Language Conference (UML'99)*. Lecture Notes in Computer Science, Vol. 1723, Fort Collins, Colorado, USA, Springer, October 1999, pp. 292-307.

[18] C. Fernstrom, "Process Weaver: Adding Process Support to Unix", in *Proceedings of 2nd International Conference on the Software Process*, 1993, pp. 12-26,

[19] A. Fuggetta, "Software Process: A Roadmap", in A. Finkelstein, ed., *The Future of Software Engineering (FOSE 2000) in Conjunction with the Proceedings of the 22nd International Conference on Software Engineering (ICSE 2000)*, Limerick, Ireland. ACM Press, June 2000.

[20] P. Garg and M. Jazayeri, "Process Centered Software Engineering Environments: A Grand Tour", in A. Fuggetta and A. Wolf, eds. *Trends in Software Process*. John Wiley & Sons, 1996, pp. 25-49.

[21] J. C. Grundy and J. G. Hosking, "Serendipity: Integrated Environment Support for Process Modelling, Enactment and Work Coordination". *Automated Software Engineering*, Vol. 5, No. 1, January 1998, pp. 27-60.

[22] P. Heiman, G. Joeris, C. A. Krapp and B. Westfechtel, "DYNAMITE: Dynamic Task Nets for Software Process Management", in *Proceedings of the 18th International Conference on Software Engineering*, Berlin, Germany. IEEE Computer Press, March 1996, pp. 331-341.

[23] K. E. Huff, "Software Process Modeling", in A. Fuggetta and A. Wolf, eds. *Trends in Software Process*, John Wiley & Sons, 1996, pp. 1-24.

[24] K. E. Huff and V. Lesser, "A Plan-Based Intelligent Assistant that Supports the Software Development Process", in *Proceedings of the 3rd ACM Symposium on Practical Software Development Environments*. ACM Press 1988, pp. 97-106.

[25] W. S. Humprey, *Managing Software Process.* Addison Wesley, Reading, Mass, 1989.

[26] W. S. Humprey and M. I. Kellner, "Software Process Modeling: Principles of Entity Process Models", in *Proceedings of the 11th International Conference on Software Engineering*. IEEE Computer Press 1989, pp. 331-342.

[27] D. Jager, A. Schleicher and B. Westfechtel, "Using UML for Software Process Modeling", in *Proceedings of the Joint 7th European Software Engineering and Foundation of Software Engineering ESEC/FSE99*, Toulouse, France, 1999, pp. 91-108.

[28] G. Junkermann, B. Peuschel, W. Schafer and S. Wolf, "MERLIN: Supporting Cooperation in Software Development Through a Knowledge-Based Environment", in A. Finkelstein, J. Kramer and B. Nuseibeh, eds. *Software Process Modelling and Technology*. Research Studies Press, Taunton, England, 1994, pp. 103-129.

[29] G. Kaiser, P. H. Feiler and S. S. Popovich, "Intelligent Assistance for Software Development and Maintenance". *IEEE Software*, No. 5, May 1988, pp. 40-49.

[30] T. Katayama, "A Hierarchical and Functional Software Process Description and its Enaction", in *Proceedings of the 11th International Conference on Software Engineering*, Pittsburgh, Pensylvania, USA. IEEE Computer Press, March 1989, pp. 343-352.

[31] C. Montangero and V. Ambriola, "OIKOS: Constructing Process-Centred SDEs", in A. Finkelstein, J. Kramer and B. Nuseibeh, eds. *Software Process Modelling and Technology.* Research Studies Press, Taunton, England, 1994, pp. 335-353.

[32] Software Process Engineering Management, "The Unified Process Model (UPM)", Document Number ad/2000-05-05, May 12, 2000.
http://www.omg.org

[33] I. Podnar, B. Mikac and A. Caric, "SDL Based Approach to Software Process Modeling", in R. Conradi, ed., *Proceedings of 7th European Workshop on Software Process Technology (EWSPT 2000)*, Kaprun, Austria, Springer, February 2000, pp. 190-202.

[34] J. M. Ribo and X. Franch, "PROMENADE: A PML Intended to Enhance Standarization, Expressiveness and Modularity in Software Process Modelling". *Research Report LSI-00-34-R*, Llenguatges I Sistemes Informatics, Politechnical of Catalonia, 2000.

[35] J. Rumbaugh, I. Jacobson and G. Booch, *The UML Reference Manual.* Addison Wesley, 1999.

[36] S. Mc Connel, "Open Source Methodology". *IEEE Software*, Vol. 4, July/August 1999, pp. 6-8.

[37] S. Sutton Jr., D. Heimbigner and L. J. Osterweil, "APPL/A: A Language for Software Process Programming". *ACM Transaction on Software Engineering Methodology*, Vol. 4, No. 3, July 1995, pp. 221-286.

[38] S. Sutton Jr. and L. J. Osterweil, "The Design of a Next-Generation Process Language", in *Proceedings of the Joint 6th European Software Engineering Conference and the 5th ACM SIGSOFT Symposium on the Foundation of Software Engineering ESEC/FSE'97.* Lecture Notes in Computer Science, Vol. 1301, Springer, 1997, pp. 142-158.

[39] K. D. Swenson, "A Visual Language to Describe Collaborative Work", in *Proceedings of the 1993 Symposium on Visual Languages*, Bergen, Norway. IEEE Computer Science Press, 1993, pp. 298-303.

[40] A. Wise, "Little JIL 1.0 Language Report". *Technical Report 98-24*, Department of Computer Science, University of Massachusetts at Amherst, April 1998.

[41] K. Z. Zamli and P. A. Lee, "Taxonomy of Process Modeling Languages", in *Proceedings of ACS/IEEE International Conference on Computer Systems and Applications AICCSA 2001*, Beirut, Lebanon. IEEE Computer Science Press, June 2001.

BIOGRAPHY

Kamal Zuhairi Zamli obtained his BSc in Electrical Engineering from Worcester Polytechnic Institute, USA in 1992, and his MSc in Real Time Software Engineering from the Centre For Advanced Software Engineering, Universiti Teknologi Malaysia in 1999. He is currently a PhD candidate at the University of Newcastle upon Tyne, UK. His research areas include Software Process, Software Engineering and Object-Oriented Analysis and Design. He has published a number of papers related to these areas.