Jurnal Teknologi, 43(D) Dis. 2005: 125–142
© Universiti Teknologi Malaysia

ENACTING THE WATERFALL SOFTWARE DEVELOPMENT MODEL USING VRPML

KAMAL ZUHAIRI ZAMLI[1]* & NOR ASHIDI MAT-ISA[2]

Abstract. This paper describes the use of a new visual language called the Virtual Reality Process Modeling Language (VRPML), in order to specify a software process. In particular, this paper demonstrates the use of VRPML to model and enact (i.e. execute) the waterfall software development model. The main objective of this paper is to investigate whether VRPML provides a sufficiently rich notation to enable the modeling and enacting of software processes.

Keywords: Software process, software engineering, process modeling languages, VRPML

Abstrak. Artikel ini menggariskan penggunaan bahasa visual yang baru, Bahasa Permodelan Proses Realiti Maya (VRPML) untuk spesifikasi proses pembangunan perisian. Secara khususnya, artikel ini membincangkan penggunaan VRPML dalam proses permodelan dan larian model air terjun. Matlamat utama kertas kerja ini adalah untuk mengkaji sama ada VRPML mempunyai notasi yang mencukupi untuk tujuan permodelan dan larian proses pembangunan perisian.

Kata kunci: Proses pembangunan perisian, kejuruteraan perisian, bahasa permodelan Proses, VRPML

1.0 INTRODUCTION

A software process can be defined as sequences of steps that must be followed by software engineers to pursue the goals of software engineering. In order to allow a better control of a particular software process, a model of that process (termed a process model) can be created using a process modeling language (PML) making the process explicit and open to examination. Furthermore, through enactment (or execution) of the process model, automation, guidance, and enforcement of the policy embedded in a particular process model can be usefully achieved.

While there has been much fruitful research into PMLs (see [1] for a recent survey), their adoption by industry has not been widespread [2]. While the reasons for this lack of success may be many and varied, our research identified two areas in which PMLs may have been deficient: human dimension issues; and support for addressing

[1&2] School of Electrical and Electronic Engineering, Universiti Sains Malaysia, Engineering Campus, 14300 Nibong Tebal, Pulau Pinang, Malaysia. Tel: 604-5937788 (ext 6079), Fax: 604-5941023.
* Corresponding author: E-mail: {eekamal, ashidi}@eng.usm.my

management and resource issues that might arise dynamically when a PML is being enacted [3]. Furthermore, no single existing PML has emerged as the *de facto* standard for supporting the modeling and enacting of software processes. These reasons suggest that research into PMLs is still necessary.

This paper describes our assessment of a new visual PML, called the Virtual Reality Process Modeling Language (VRPML) [1, 3, 4-8] developed as part of our on-going research. The main design objectives for VRPML are:

(1) To develop an expressive, executable, and easy to use visual PML
(2) To address some of the perceived deficiencies in existing PMLs particularly in terms of the support for dynamic creation and assignment of tasks and resources, as well as the support for the awareness and visualization issues.

Although VRPML has been successfully employed to model and enact the standard benchmark problem in software engineering (i.e. the ISPW-6 problem) involving the software change request [9], the author felt that such experience may be insufficient to evaluate VRPML completely. One reason is that the ISPW-6 problem is perhaps too specific to the software change request process.

A more general case study process, particularly involving the software processes for a complete software development model is required. These processes must be explicit and well-defined in terms of their inputs and outputs. Arguably, if one could use an existing definition of a development model in which activities and their inputs and outputs have already been well-defined, more effort can be concentrated on the modeling and enacting issues and less on defining the stages (and activities). As the waterfall software development model seems to fit well into this category, it will be used here. The focus of this paper is, therefore, to explore the expressiveness of the VRPML notation for supporting the modeling and enacting of a complete software development model.

2.0 OVERVIEW OF VRPML

VRPML is a control-flow based visual PML for supporting the modeling and enacting of software processes. In VRPML, software processes are generically modeled. Resources (in terms of software engineers, artifacts and tools) can be dynamically assigned and customized for specific projects from a generic model.

Software processes are specified in VRPML as graphs, by interconnecting nodes from top to bottom using arcs that carry run-time control-flow signals. The complete description of the syntax and semantics of VRPML can be found in [6].

As an illustration, Figure 1 presents an excerpt of the VRPML solution to the ISPW-6 problem. Similar to JIL [10] and Little JIL [11], software processes in VRPML are described using process step abstractions, which represent the most atomic representation of a software process (i.e. the actual activity that software engineers

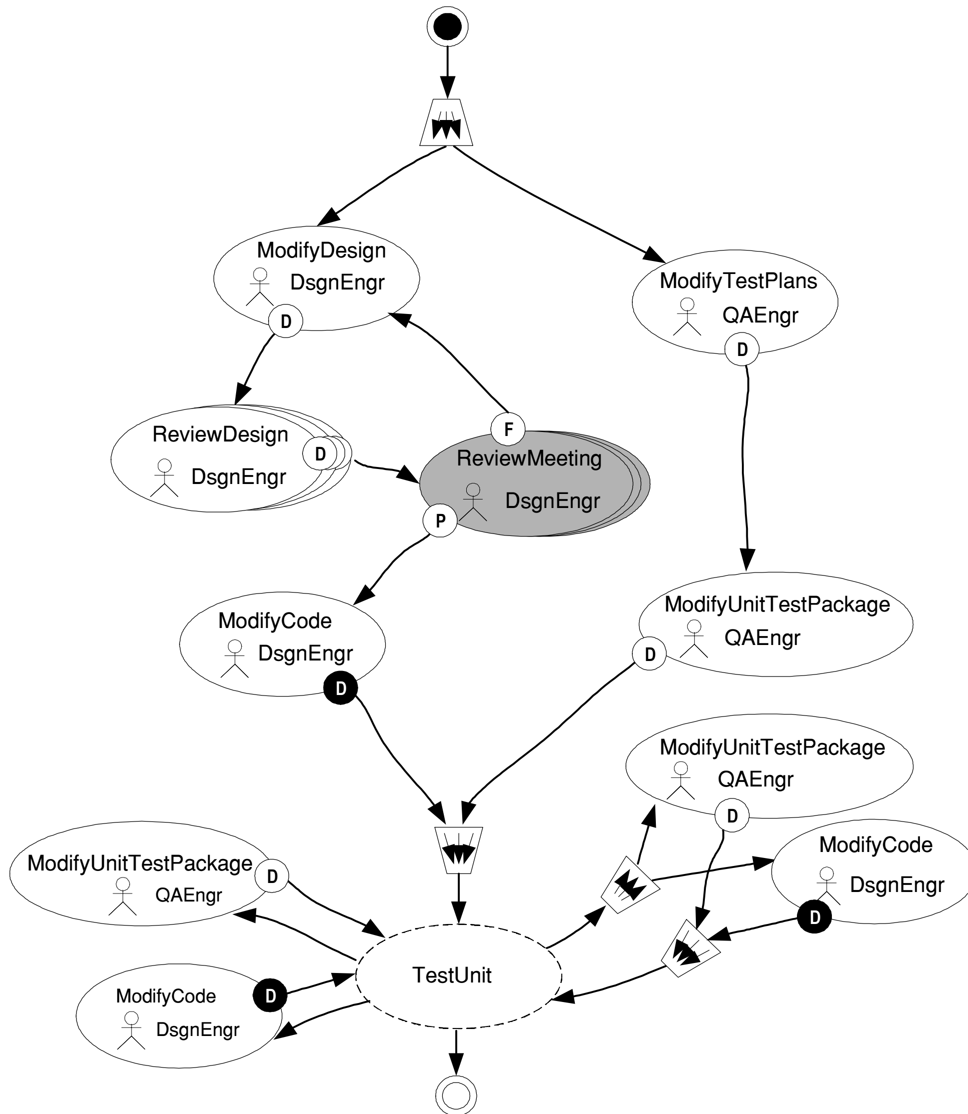

Figure 1 Excerpt of the VRPML graph

are expected to perform). These activities are represented as nodes, called activity nodes (shown as small ovals with stick figures).

As depicted in Figure 1, VRPML supports many different kinds of activity nodes. They include: general-purpose activity nodes (shown as individual small ovals with stick figures); multi-instance activity nodes (shown as overlapping small ovals with stick figures); and meeting activity node (shown as small and shaded overlapping ovals with stick figures). Both multi-instance activity nodes and meeting activity nodes have associated depths, indicating the actual number of engineers involved (and also the number of identical activities in the case of multi-instance activity).

The firing of activity nodes is controlled by the arrival of a control flow signal. In VRPML, an initial control flow signal is always generated from a start node (a white circle enclosing a small black circle). A stop node (a white circle enclosing another white circle) does not generate any control flow signals. Control flow signals may also be generated at the completion of a node, often from special completion events called transitions (shown as small white circles with a capital letter, attached to an activity node) or decomposable transitions (small black circles with a capital letter). Decomposable transitions enable automation scripts or sub-graphs to be specified (and executed if selected) as post-conditions before allowing transition to generate a control flow signal. The sub-graph associated with the decomposable transition representing Done (labeled D) for the activity node called Modify Code is given in Figure 2.

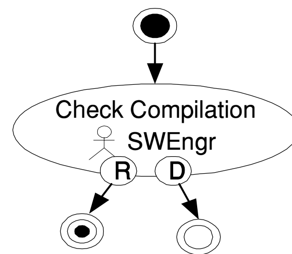

Figure 2 Sub-graph for decomposable transition labeled D in Modify Code

When Check Compilation fails, the assigned software engineer can select the transition R (for re-do). As a result, a control-flow signal will be generated to re-enact its parent node (i.e. Modify Code) through a re-enabled node (shown as two white circles enclosing black circle). Otherwise, if the compilation is successful, the assigned engineer can select the transition D (for Done). In this case, the control-flow signal will be generated and propagated back to the main graph to enable the subsequent connected node.

In VRPML, activity nodes can also be enacted in parallel using combinations of language elements called merger and replicator nodes (shown as trapezoidal boxes with arrows inside). To improve readability, a set of VRPML nodes can be grouped together and replaced by a macro node (shown as dotted line ovals), with the macro expansion appearing on a separate graph. For example, referring to Figure 1, Test Unit is a macro node. The macro expansion of Test Unit is given in Figure 3.

For every activity node, VRPML provides a separate workspace, the concept borrowed from ADELE-TEMPO [12], APEL [13] and MERLIN [14]. Figure 4 depicts the sample workspace for the activity node called Review Meeting in Figure 1. A workspace typically gives a work context of an activity as it hosts resources needed

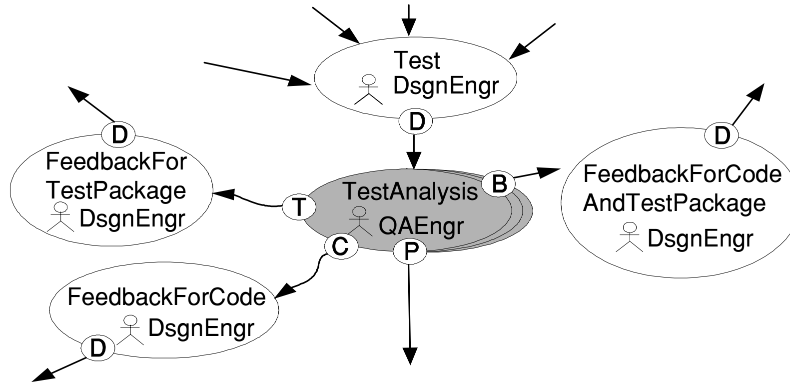

Figure 3 Macro expansion for Test Unit

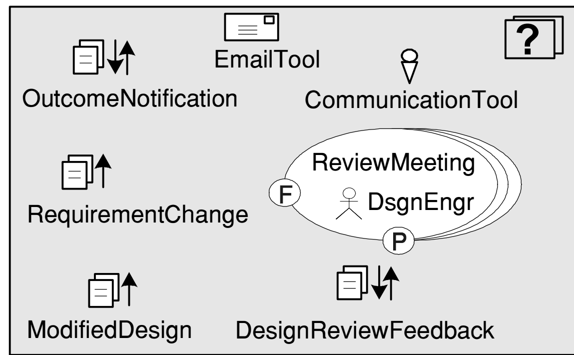

Figure 4 Sample workspace for activity node review meeting

for enacting the activity: transitions, artifacts (shown as overlapping two overlapping documents with arrows for depicting access rights), communication tools (shown as a microphone, and an envelope), and any task descriptions (shown as a question mark). Effectively, when an activity is undertaken, the workspace is mapped into a virtual room, transitions into buttons, and artifacts, communication tools (i.e. for synchronous and asynchronous forms of communications) and task description into objects which can be manipulated by software engineers to complete the particular task at hand.

As part of its enactment model, VRPML relies on its resource exception handling mechanism. In VRPML, resources include roles assignment, artifacts and tools (including communication tools) in a workspace as well as the depths of multi-instance activity nodes and meeting activity nodes. Depending on the needs of a particular software development project, these resources can either be allocated during graph instantiation or dynamically during graph enactment.

3.0 OVERVIEW OF THE WATERFALL DEVELOPMENT MODEL

The earliest form of the software processes based on the waterfall model was introduced by Royce [15]. Since then, many variants from the original waterfall model have been proposed. One of its variants [16] is shown in Figure 5.

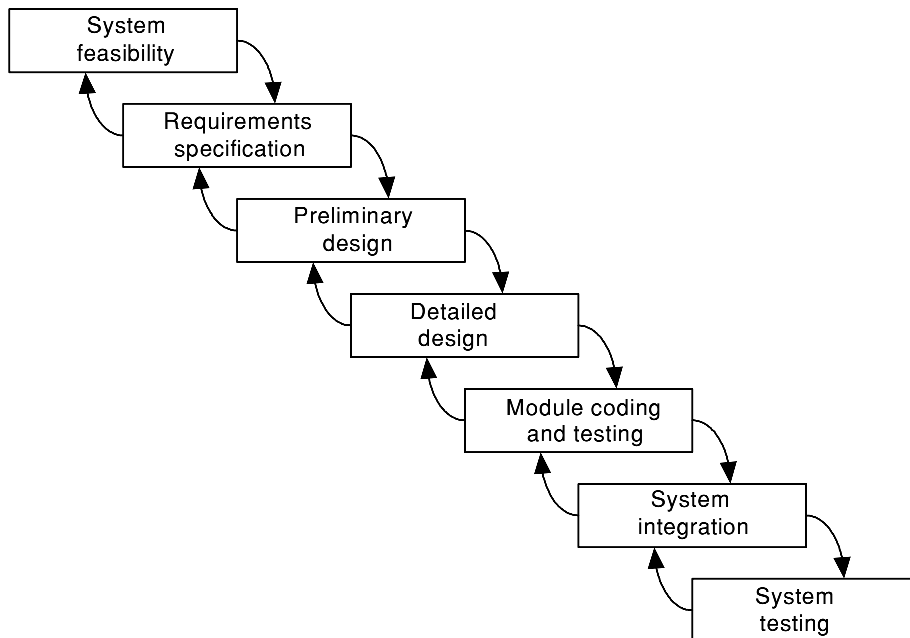

Figure 5 The waterfall model

Among the characteristics of the waterfall model are:

(1) The model is divided into a number of separate stages from system feasibility to maintenance.
(2) Each stage has a clearly delineated activity which is performed in a linear and sequential manner.
(3) Each stage is also independent that is, there is no overlap among stages.
(4) Feedback is usually provided to the preceding stage.
(5) The completion of a stage is determined by a review either formally or informally and conducted at the end of each stage so that development can proceed to the next stage. This is important because the output of the current stage often becomes the input of the next stage.

In order to support the modeling and enacting of software processes implementing the waterfall model, each stage of the model must be precisely defined in detail. Building on the work by Sommerville [17] and the waterfall model given earlier, Table 1 in the next page summarizes the possible activities along with their inputs

Table 1 Waterfall model activities, inputs, and outputs

Waterfall stage	Activities	Inputs	Outputs
System feasibility	Analyse and define requirements	Customer requirements	Draft feasibility study Draft requirement documents
	Review system feasibility	Draft feasibility study Draft requirement documents	Feasibility study Requirement documents
Requirements specification	Prepare functional specification	Requirement documents	Draft functional specification
	Prepare acceptance test plan Prepare draft user manual	Requirement documents Draft functional specification	Draft acceptance test plan Draft preliminary user manual
	Review specification	Draft functional specification Draft acceptance test plan Draft preliminary user manual	Functional specification Acceptance test plan Preliminary user manual
Preliminary design	Prepare architectural specification	Requirement documents Functional specification	Draft architectural specification
	Prepare system test plan	Requirement documents Functional specification	Draft system test plan
	Review preliminary design	Draft architectural Specification Draft system test plan	Architectural specification System test plan
Detailed design	Prepare interface specification	Functional specification Architectural specification Requirement documents	Draft interface specification
	Prepare integration test plan	Functional specification Architectural specification Requirement documents	Draft integration test plan
	Prepare design specification	Functional specification Architectural specification Draft interface specification Requirement documents	Draft design specification
	Prepare unit test plan	Functional specification Architectural specification Draft interface specification Requirement documents	Draft unit test plan
	Review detailed design	Draft interface specification Draft design specification Draft integration test plan Draft unit tast plan	Interface specification Design specification Integration test plan Unit test plan

(cont.)

Table 1 (continued)

Waterfall stage	Activities	Inputs	Outputs
Module coding and testing	Perform coding	Requirement document Design specification	Draft program code
	Perform unit and module testing	Unit test plan Draft program code	Draft unit test report
	Review coding and testing	Draft program code Draft unit test report	Program code Unit test report
System integration	Perform integration testing	Integration test plan Program code	Draft integration test report
	Prepare final user manual	Preliminary user manual Functional specification Program code	Draft user manual
	Review integration testing	Draft integration test report Draft user manual	Integration test report User manual
System testing	Perform system testing	System test plan Acceptance test plan Program code	Draft system test report
	Review system	Program code User manual Draft system test report	Final release

and outputs. It must be stressed that this is only one of the possible list of activities as there are a number of variations to the waterfall model.

Referring to Table 1, the summary of activities involved in each stage of the waterfall model raises a number of issues. Firstly, the roles associated with each defined activity have not been identified. In this case, it is assumed that all of the software engineers involved have the required skills to perform the activities assigned to them. Hence, the role for each activity will be simply software engineers.

Secondly, although the ordering of activities in each stage has not been defined, they can be indirectly inferred from their input dependencies. In fact, activities in a stage can also be enacted in parallel when they are independent, that is, they do not require any input from each other. This will be reflected in the VRPML solution given below.

Finally, in order to highlight only the key aspects of VRPML, only a partial solution of the waterfall model from Table 1 will be presented here. The complete solution can be found in Zamli [7].

4.0 VRPML SOLUTION OF THE WATERFALL DEVELOPMENT MODEL

The main graph of the VRPML solution for the software processes based on the waterfall model is given in Figure 6 consisting of 7 macros namely: System feasibility; Requirements specification; Preliminary design; Detailed design; Module coding and testing; System integration; and System testing.

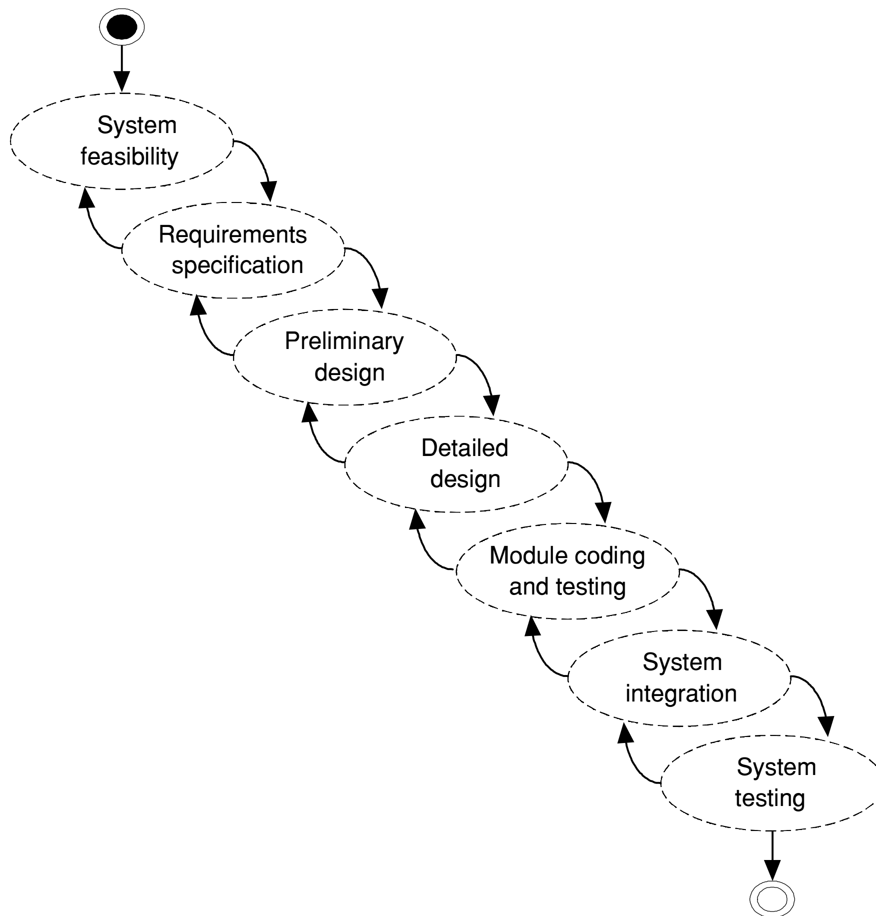

Figure 6 Main VRPML graph for the Waterfall Model

To further illustrate the VRPML notation, one of the macros, called Detailed Design, is shown in Figure 7. The presence of macros related to other stages in the Figure is merely to give focus and context to the expansion.

The macro expansion for Detailed Design consists of four multi-instance activity nodes (Prepare Interface Specification, Prepare Integration Test Plan, Prepare Design Specification, and Prepare Unit Test Plan) and one meeting node (Review Detailed Design). Since Prepare Interface Specification and Prepare Integration Test Plan are

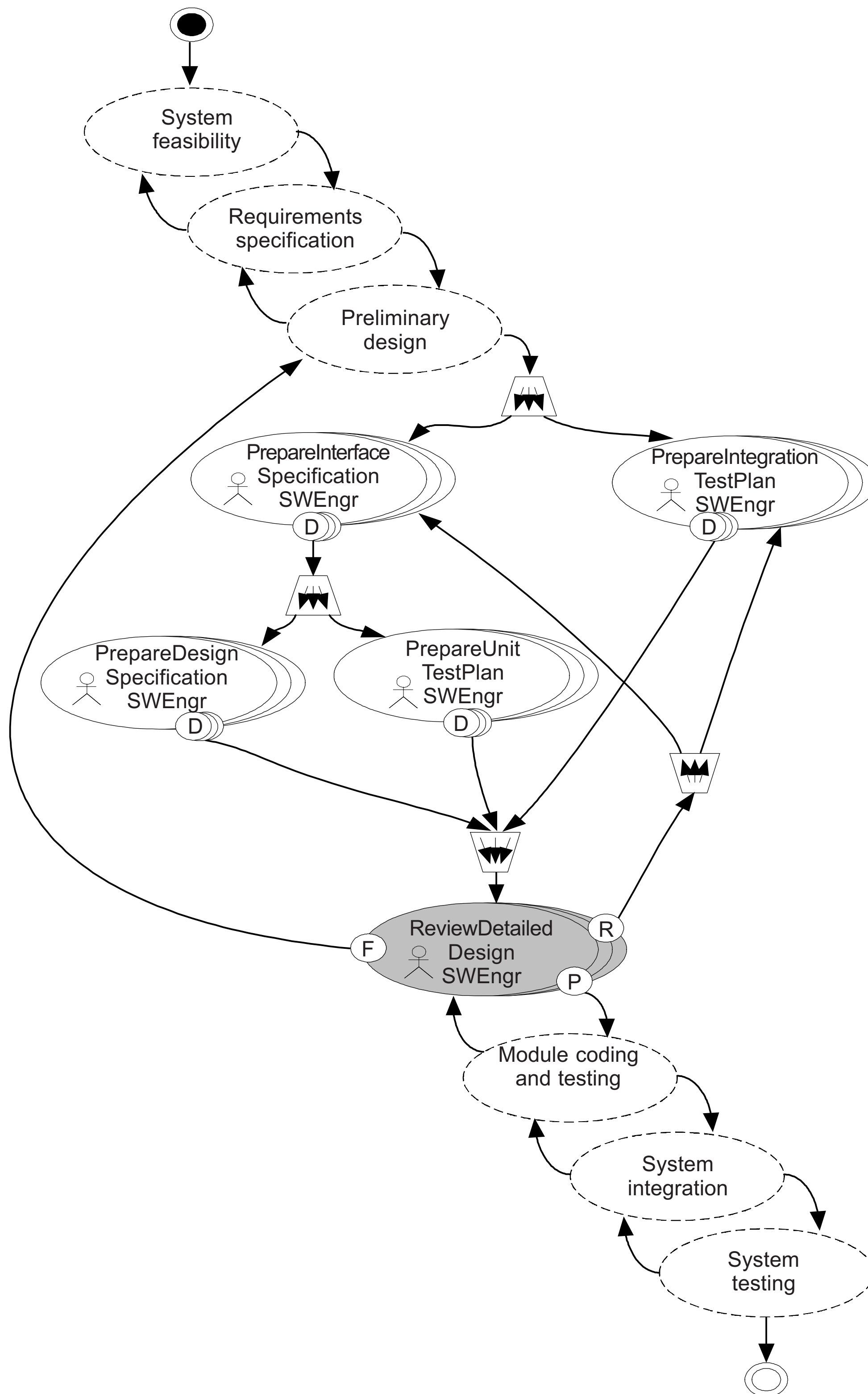

Figure 7 Macro expansion for Detailed Design

independent of each other, they can be enacted in parallel. However, Prepare Design Specification and Prepare Unit Test Plan can only be enacted after Prepare Interface Specification has been completed. This is because both Prepare Design Specification and Prepare Unit Test Plan require an artifact from Prepare Interface Specification, called the Interface Specification, as one of their inputs (see Table 1). Actually, once Prepare Interface Specification has been completed, both Prepare Design Specification and Prepare Unit Test Plan can be enacted in parallel. Lastly, Review Detailed Design is enacted when all the above activities have been completed.

In terms of transitions, Prepare Interface Specification, Prepare Integration Test Plan, Prepare Design Specification and Prepare Unit Test Plan each have only one defined transition for Done (labeled D) to allow their completion. However, Review Detailed Design has three defined transitions: Redo (labeled R) in order to allow loop back to the previous activities; Passed (labeled P) in order to move to the next stage; and Feedback (labeled F) in order to permit feedback to the previous stage.

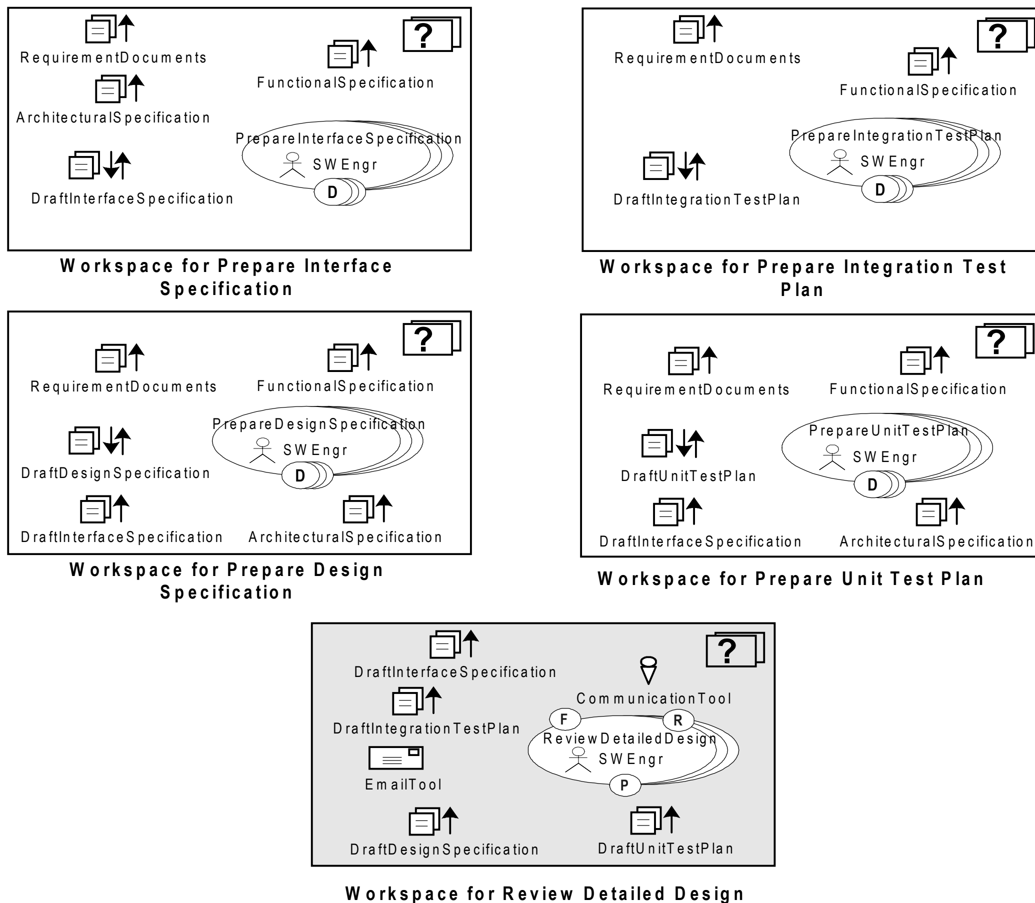

Figure 8 Workspaces for Detailed Design

In terms of workspaces, they can be straightforwardly defined by analyzing the inputs for each activity as described in Table 1. As an illustration, Figure 8 depicts all the respective workspaces for Detailed Design.

Although not shown in this paper (see [7]), all of the macros given in Figure 6 expand into a combination of multi-instance activity nodes and meeting activity nodes connected by arcs. The reason for using multi-instance activity nodes and meeting activity nodes is to demonstrate that VRPML supports the dynamic creation of tasks, that is, no prior assumption is made when constructing the model in terms of how many engineers have to be assigned to any of the activities represented by these nodes.

As far as enactment is concerned, the VRPML support system is responsible to allow the process model to be enacted. The overall structure of the VRPML support system is shown in Figure 9.

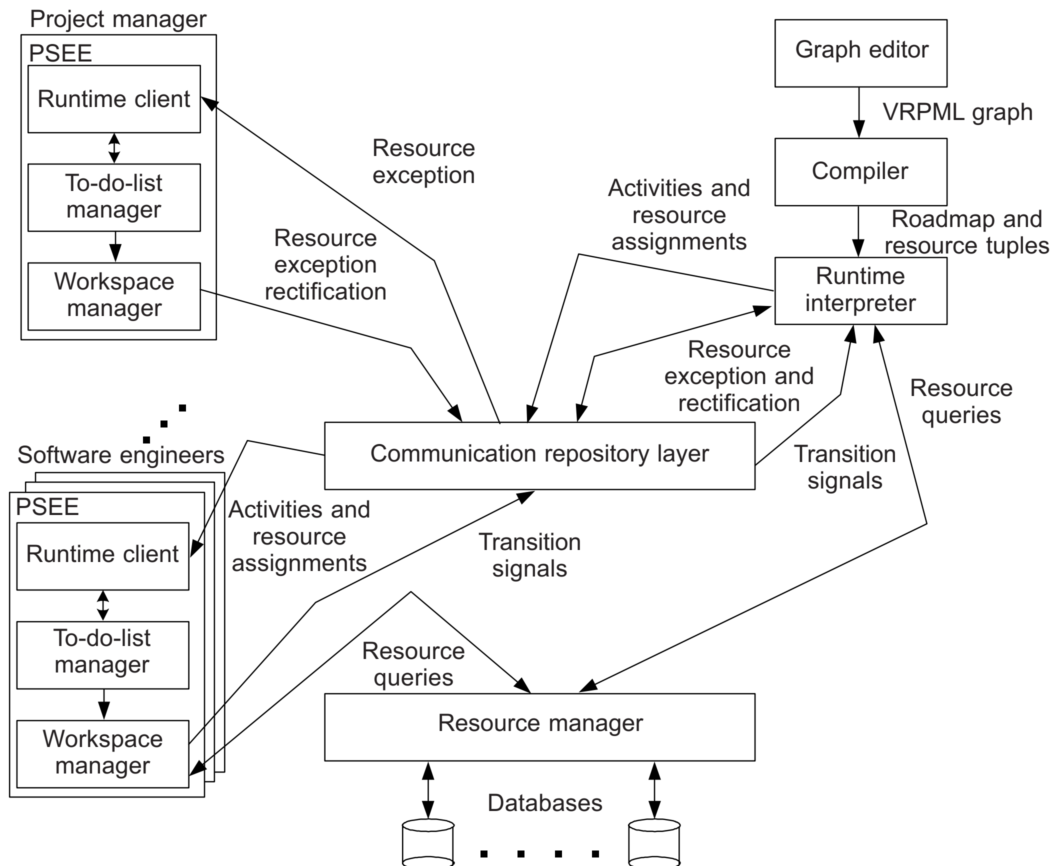

Figure 9 VRPML support system

The main components of the VRPML support system consist of:

(1) Graph editor – allows the VRPML graphs to be specified.
(2) Compiler – compiles the VRPML graphs into an immediate format for enactment.
(3) Runtime interpreter – interprets the compiled VRPML graph.
(4) Communication repository layer – allows communication between the runtime interpreter, runtime client, and workspace manager.
(5) Resource manager – queries the databases for artifacts.
(6) Process Centered Environment (PSEE) – encapsulates three main sub-components: the runtime client, the to-do-list manager, and the workspace manager. The runtime client retrieves activities and resource assignments from the communication repository layer. The to-do-list manager manages the activities assigned to a particular software engineer whist the workspace manager manages activity workspace in a virtual environment, manages activity transition, and forward queries to the resource manager.

The complete description of the VRPML support system, however, is beyond the scope of the paper. Interested readers are referred to Zamli [7].

To illustrate enactment, Figure 10 shows a sample snapshot of the to-do-list GUI for a software engineer name Kamal where the current activity in the to-do-list queue is Review Detailed Design whilst Figure 11 depicts the snapshot of resource allocation activity performed by the project manager.

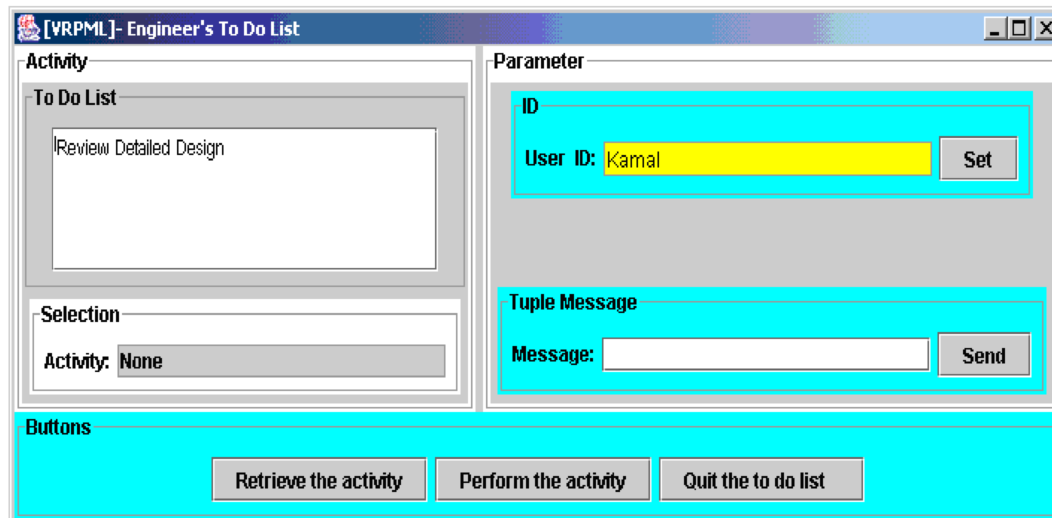

Figure 10 Snapshot of the engineer's to-do-list

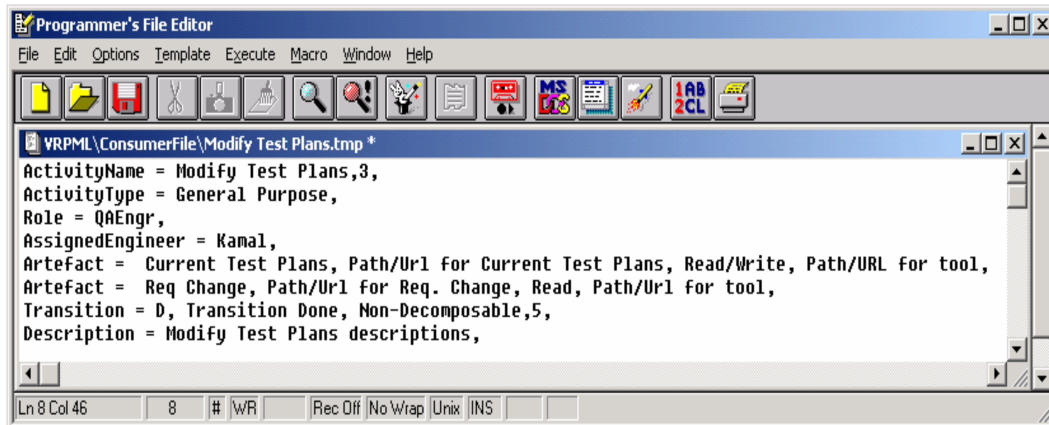

Figure 11 Snapshot of resource allocation activity

5.0 DISCUSSION

The fact that VRPML provides a sound solution to the waterfall development model as well as its enactment gives an encouraging indication of the expressiveness of VRPML. This can be further supported from the fact that the VRPML solution itself can be arranged like the waterfall development model. The ability of VRPML to support such arrangement may be useful to facilitate process understanding. In fact, similar arrangement may not be possible in other visual PMLs such as Slang [18], Promenade [19], and APEL [13].

Although the UML activity diagram [20] is non-enactable, it can be compared to VRPML in terms of its graph representation. Figure 12 depicts an example of a software process expressed using the UML activity diagram.

The UML activity diagram representation of a software process is simple and intuitive. Nonetheless, while the UML activity diagram can be used to express activities in a software process, it lacks features to express the individual role, resources, work contexts, and the completion of activities. Furthermore, UML activity diagrams do not have a well-defined executable semantics (i.e. as in VRPML). A known experience of using UML as a PML can be seen in the design of PROMENADE [19]. Here, the authors of PROMENADE dismiss the use of activity diagram as a PML, as PROMENADE mainly relies on class diagrams and object constraint language for supporting the modeling and enacting of software processes. Furthermore, in doing so, the authors of PROMENADE extensively extend the UML meta-models, hence, affecting the standardization of UML. For these reasons, we believe that UML is not particularly suitable as a PML.

Referring to the VRPML solution to the waterfall development model discussed in the previous section, multi-instance and meeting activity nodes were sufficient to construct that process model. Thus, at a glance, removing the general purpose activity

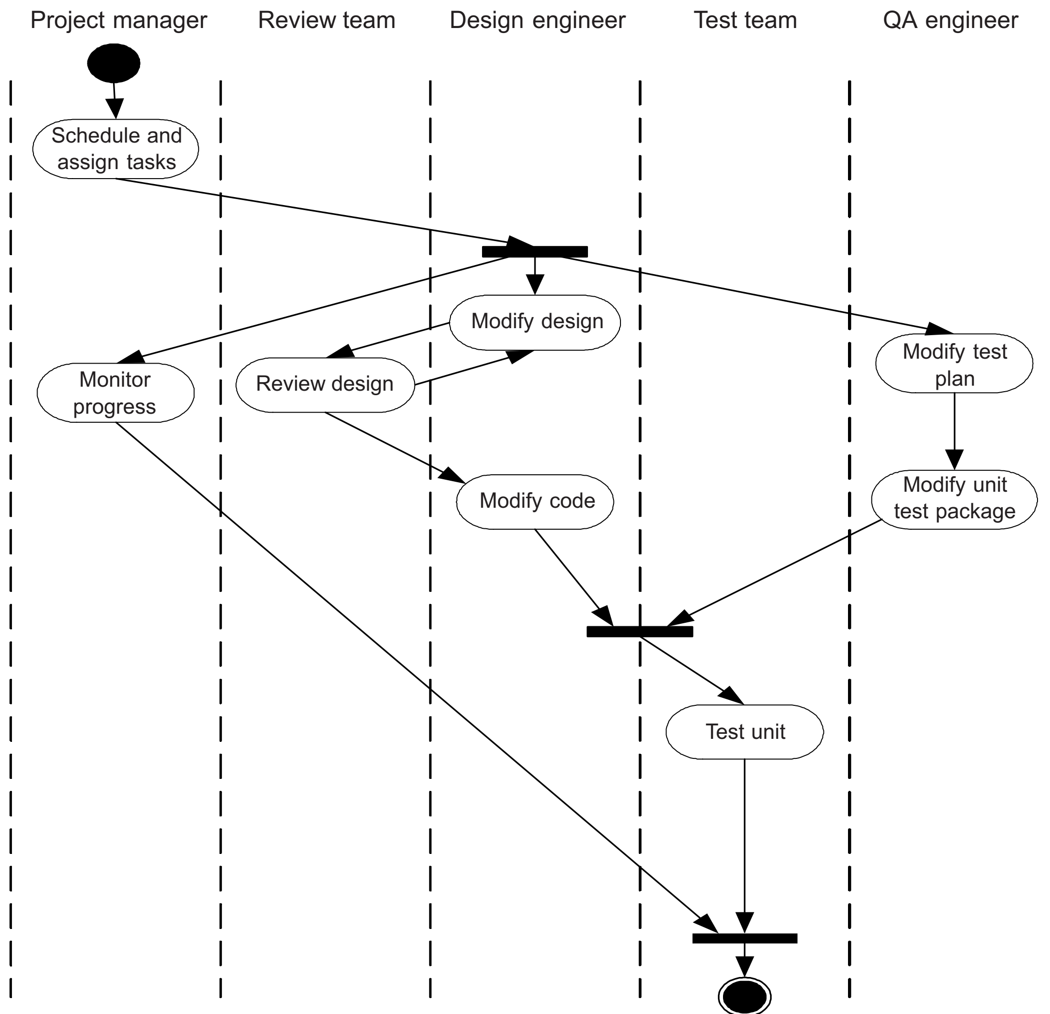

Figure 12 Example of the UML activity diagram

node from the VRPML notation seems beneficial to reduce the language complexity. Nevertheless, eliminating the general purpose activity node from the notation can be disadvantageous. As far as readability of a VRPML graph is concerned, it can be difficult to distinguish whether an activity will be solely performed by one person or collaboratively by more than one person [3]. Therefore, it is suggested that both general purpose activity nodes and multi-instance activity nodes are kept as part of the notation.

Concerning enactment, the fact that VRPML can produce an enactable model is helpful to facilitate coordination of activities involved in a particular development cycle. In addition, the support for enactment in VRPML can also be helpful for the following reasons:

(1) It provides guidance through the steps to be taken. Such guidance is particularly useful for junior software engineers.
(2) It can enforce strict procedures and policies. Enforcement of strict procedures is sometimes important in cases such as developing critical systems where human lives depend on a piece of software. An example of such a system would be a car auto-cruise control system. In this case, the software development team in charge of developing such a system may require its defined steps to be followed precisely. For example, evolution of the software in such a system must be strictly controlled. *Ad hoc* changes must not be permitted because such changes may introduce bugs which may not be tested and accounted for. Such bugs could be dangerous especially if they affect the mechanism to control the speed of the car in auto-cruise.
(3) It permits the automation of tasks. In software engineering, there are many tasks which can benefit from automation. For example, although tasks such as compiling and linking source codes look simple, they can be painstakingly dull especially if the source codes are very large and involves multiple modules. Such mundane tasks, if automated, can relieve software engineers from tedious routine work (and reduce potential human errors), and consequently, improve software engineer's productivity.

6.0 CONCLUSION

In conclusion, this paper has demonstrated the use of the VRPML for modeling and enacting of software processes. VRPML can be used to model a more realistic software process problem such as the spiral and the extreme programming model.

REFERENCES

[1] Zamli, K. Z., and N. A. Mat Isa. 2004. A Survey and Analysis of Process Modelling Languages. Malaysia Journal of Computer Science. 17(2): 68-89.
[2] Jaccheri, M. J., R. Conradi, and B. H. Drynes. 2000. Software Process Technology and Software Organisations. Proceedings of the 7th European Workshop on Software Process (EWSPT 2000). Kaprun, Austria. 96-108.
[3] Zamli, K. Z., and P. A. Lee. 2003. Modelling and Enacting Software Processes Using VRPML. Proceedings of the 10th IEEE Asia-Pacific Conference on Software Engineering. Chiang Mai, Thailand. IEEE CS Press. 243-252.
[4] Zamli, K. Z., and P. A. Lee. 2001. Taxonomy of Process Modelling Languages. Proceedings of the ACS/IEEE International Conference on Computer Systems and Applications. Beirut, Lebanon. IEEE 435-437.
[5] Zamli, K. Z. 2001. Process Modelling Languages: A Literature Review. *Malaysia Journal of Computer Science.* 14(2): 26-37.
[6] Zamli, K. Z., and P. A. Lee. 2002. Exploiting a Virtual Environment in a Visual PML. Proceedings of the 4th International Conference on Product Focused Software Process Improvements (PROFES02). In M. Oivo and S. Komi-Sirvio (Eds.). Lecture Notes in Computer Science Volume 2559. Rovaniemi, Finland. 49-62.
[7] Zamli, K. Z. 2003. Supporting Software Processes for Distributed Software Engineering Teams. PhD. Thesis. School of Computing Science, University of Newcastle upon Tyne, United Kingdom.

[8] Zamli, K. Z., and N. A. Mat Isa. 2005. The Computational Model for a Flow-based PML. Proceedings of the AIDIS International Conference on Applied Computing. Algarve, Portugal. 217-224.

[9] Kellner, M. I., P. H. Feiler, A. Finkelstein, T. Katayama, L. J. Osterweil, M. H. Penedo, and H. D. Rombach. 1990. Software Process Modelling Example Problem. Proceedings of the 6th International. Software Process Workshop, Hakodate, Japan.

[10] Sutton, S. Jr., and L. J. Osterweil. 1997. The Design of a Next-Generation Process Language. Proceedings of the Joint 6th European Software Engineering Conference and the 5th ACM SIGSOFT Symposium on the Foundation of Software Engineering, Lecture Notes in Computer Science Volume 1301. 142-158.

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

[12] Belkhatir, N., J. Estublier, and W. Melo. 1994. ADELE-TEMPO: An Environment to Support Process Modelling and Enaction. In A. Finkelstein, J. Kramer and B. Nuseibeh. (Eds.). *Software Process Modelling and Technology*. Taunton, England: Research Studies Press. 187-122.

[13] Dami, S., J. Estublier, and M. Amiour. 1998. APEL: A Graphical Yet Executable Formalism for Process Modelling. *Automated Software Engineering*. 5(1):61-96.

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

[15] Royce, W. W. 1970. Managing the Development of Large Software Systems. Proceedings of IEEE WESCON. 1-9.

[16] DeBellis, M., and C. Haapala. 1995. User-Centric Software Engineering. *IEEE Expert*. February 1995: 34-41.

[17] Sommerville, I. 2001. *Software Engineering (Sixth Edition)*. Addison Wesley.

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

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

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