Software Maintenance Process: Tasks and Methods

Khaled Md. Khan[1]
Bruce W. N. Lo[2]
Torbjorn Skramstad[3]
Sikander M. Khan[4]

[1]School of Computing and Information Technology
University of Western Sydney
PO Box 10 Kingswood
NSW 2747, Australia
Email: k.khan@uws.edu.au

[2]School of Information Technology and Multi Media
Southern Cross University
P.O. Box 157 Lismore
NSW 2480 Australia
Email: blo@scu.edu.au

[3]Department of Informatics and Computer Science
Norwegian University of Sciences and Technology
N- 7491 Trondheim, Norway.
Email: torbjorn.skramstad@idi.ntnu.no

[4]Department of Information and Library Sciences
University of Dhaka
Ramna 1000 Dhaka, Bangladesh

Abstract

The paper proposes a framework of various tasks involved in the software maintenance process. The work reported in this paper disassembles the complex process of software maintenance into tasks as to aid in the allocation of resources, acquisition of appropriate tools, and distributing responsibilities of the software maintenance process. The associated toolsets, methods, input-output sources, and communication protocols between tasks are addressed in this paper. This work is intended to conveying a high-level understanding of the software maintenance process and its dimensions. Software maintenance can be viewed in various ways depending on its purposes, nature and characteristics. We also show that software maintenance and development are two separate processes, but they are highly interrelated and interdependent. We attempt to find the intersection of activities between the software development and maintenance processes in the final part of this paper, and the software maintenance process is integrated with the development process into a high level software life cycle model.

Keywords

Software maintenance, software development, software life cycle.

INTRODUCTION

Information systems always tend to change and evolve, some are more frequent, some are less frequent. The evolution of an information system is unavoidable, and it tends to degrade desirable properties of the system over time (Notkin 1993). The evolutionary nature of the information system is formally defined and characterised by Lehman (1980). Lehman has proposed the essential laws of the structure of evolving software. Software evolutionary laws in fact leave us with two options: either to maintain the products with their continual changing nature, or to accept the degrading performance of the product. To maintain a software product properly over time, we need a well-defined software maintenance process model.

Some work have already been done towards software process modelling (Humphrey 1989a), but comparatively a less significant progress has been made on the area of software maintenance process modelling. The research tracks on software maintenance reported particularly in the last ten years suggest that the software engineering community continues to work for an widely accepted solution for the complex task of software maintenance. It is apparent that a widely accepted software maintenance paradigm is difficult to materialise. This is partly because the software maintenance process involves multi-dimensional activities comprising varieties of application domains with diverse sets of design rationales, and highly subjective programming styles. A universally accepted process is difficult to define to solve a wide range of maintenance problems.

The procedures followed in software maintenance by practitioners are normally not well defined. It is reported that very few organisations adopt a separate process for maintenance because they cannot make a distinction between software maintenance and software development. This leads to some fundamental questions like, 'Is software maintenance a separate process?', 'Can one process satisfy all types of software maintenance?', and 'Which factors or activities make software maintenance distinct from, or similar to the development process?'. No research work on software maintenance could avoid these three issues.

There are reasons to believe that a better formalised maintenance process would significantly improve the efficiency of software maintenance (Haziza *et al.* 1992). A more defined formalism describing various tasks, tools and methods is required to enable a clear understanding of the process. In this direction, modelling the software maintenance process would be the initial requirement. We have defined a framework of various tasks involved in the software maintenance process in this paper to meet this initial requirement. The purpose of this work is to disassemble the complex process of software maintenance into related tasks to aid in the allocation of resources, acquisition of appropriate toolsets, and distributing responsibilities.

The paper is intended to convey a high-level understanding of the software maintenance process. First we define software maintenance as a collection of well-defined tasks aimed at maintaining an existing application software product. The paper also highlights the relationship between the software maintenance and the software development processes in an integrated life cycle model. There are plenty of reasons to argue that software maintenance and development are essentially two different processes, but intersection between activities of these two processes can easily be seen in a well defined software life cycle model. We attempt to define an intersection between the software development and maintenance processes later in this paper. Finally, we briefly compare our work with other models.

BACKGROUND OF THE WORK

The motivation of the work reported in this paper was actually generated from a maintenance project of a PC based application software. Our candidate system was a small Inter Bank Reconciliation System (IBRS) used in a developing country in Asia. Our candidate system was relatively small in terms of lines of code (LOC), and was written in a fourth generation language (4GL). The system was organised hierarchically, and included more than 100 modules and almost 35 physical data files in various format. The system was capable of providing at least 40 various types of services to the user. The company developing the system, later wondered whether it was possible to transform the system into a more portable and efficient programming language platform, like C, keeping the entire functionality of the system intact. One of the authors of this paper was assigned the responsibility to lead the project. He realised that it was a reengineering task. This maintenance experience and our previous work reported in Khan *et al.* (1996) prompted us to propose a software maintenance process framework as reported in this paper.

FRAMEWORK OF SOFTWARE MAINTENANCE PROCESS

After presenting our generic software maintenance model in (Khan *et al.* 1996), we realised later that the model lacks some detail information associated with each task. From this observation, we refined our work further, coupling methods, tools, source and destination of input and output associated with each task. This work is an extension of our earlier work in (Khan *et al.* 1996).

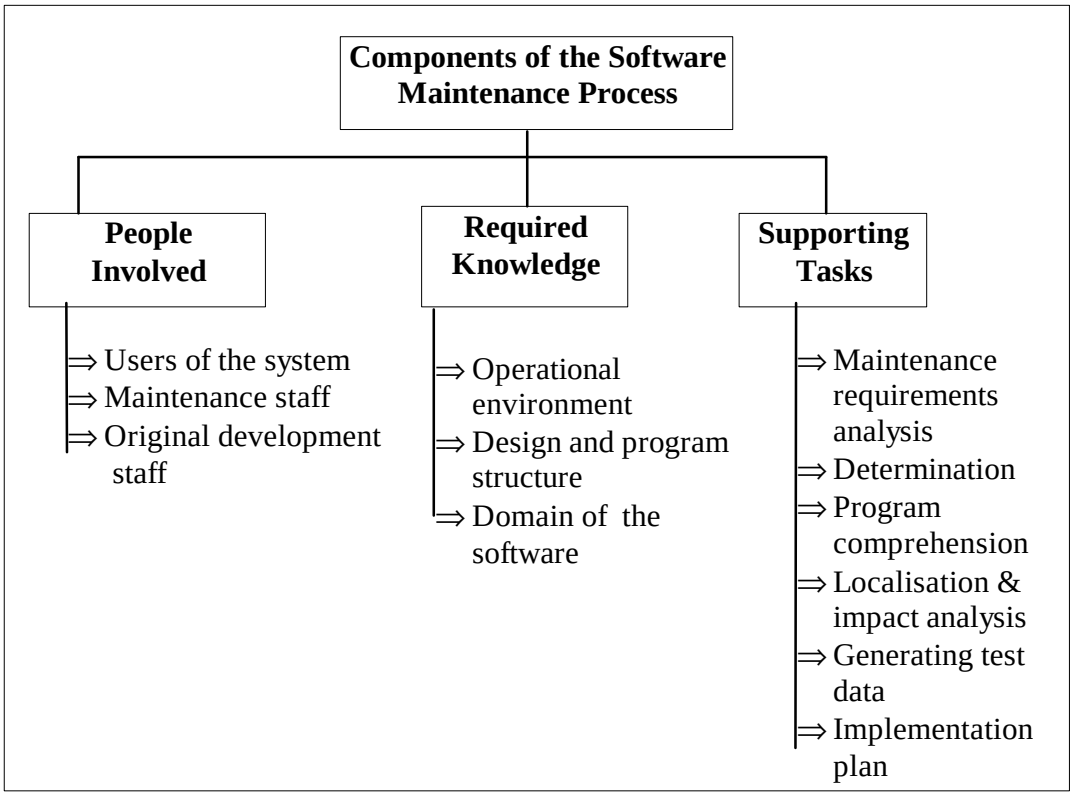


Figure 1: Components of software product maintenance process

In software maintenance three ingredients are most important:

- *people* who are involved,
- supporting *tasks* which are well defined, and
- *knowledge* (structural, semantics) about the software product

These three are inter-related and interdependent to each other as shown in Figure 1. We will mainly focus our attention on the supporting *tasks* involved in the software maintenance process in this paper. These tasks are believed to be fundamental for all types of maintenance work, but the degree of importance of each task varies from project to project. Supporting tasks are augmented with the people involved, the readily available knowledge of the software product, and the nature of the operational environments. Our presentation in this paper is not tied to any specific process modelling notations or process program language. The framework defines a collection of interrelated tasks describing the properties of the software product maintenance process. The framework of the maintenance process is based on a view of a sequence of tasks performed by agents. The agents can be toolsets, tasks, people, or, most often, a combination of these three. A task itself may also play the role of an agent. The tasks depict what is 'really going on' in software maintenance activity. Each task has seven major components or properties as follows.

- *objectives*,
- *input* information, and the source of input to feed the tasks,
- *output* information that the task produces,
- *methods*, the way a task performs its activities,
- *toolsets* that supports the objectives and methods of the task,
- *communication* that establishes relationships among the tasks, and
- *status* of the maintenance process.

The high level software maintenance process shown in Table 1 focuses on how various tasks in a chain are to be performed in the context of the maintenance. Table 1 is a two dimensional matrix in which the first column shows all components required for a given task. Each of the remaining columns is allocated for individual task and their required components. We now briefly describe the components required for each of the tasks defined in our process.

Objectives

Each task has one or more predefined objective. A task performs certain function, and supports the activities of another task. This model consists of a sequence of defined tasks that must be performed in order to achieve the designated objectives. Each task could be further enlarged into a complete model if the project is complex enough.

Input and output information

Each task is committed to understand the information it receives, and it performs its acts on this information. Each task has two classes of information. One is the information required for the task to perform its service, called sources of input information. The other category is the information generated within the task, and used by the task itself called output information. One of the neglected elements in maintenance models is the source of input information. Quite often it is not mentioned how the various tasks will get their input data.

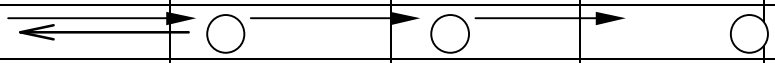
	SOFTWARE MAINTENANCE TASKS				
Components of the Tasks	**Maintenance Requirements analysis**	**Determination**	**Program comprehension**	**Localisation and impact analysis**	**Generating test cases**
Objectives	• Trigger of the process enactment	• Examines the technical and economical feasibility	• Understanding semantics and architecture of the software	• Identifying program location and ripple effects	• Tests cases defined for proposed changes
Sources of input	• Program execution at the operational site • Real users of the system	• Requirements specification, • knowledge on software and its nature and characteristics, • organisational policy, • status of tools and staff availability	• Source code, • Information from original designer and programmers readable from program documents	• Requirements specification, • source code, • class hierarchy, • function call sequences, • data structures, data file format	• Req.spec. • source code • function names, • variables used
Output	• Refined maintenance requirement specifications	• Requesting to filter req. spec. • termination message, • filtered requirements, • primary knowledge about the software	• Recovered system design artefacts, • Program domain	• Function names, • variables declarations	• Test data, • program path spec.
Methods	• Interviews, • Prototyping	• Verify requirements	• Program walk through, • Program slicing, • Execution of program	• Program walk through, • program slicing, • execution of program	• Regression testing, • quality control
Tools	• Not specific	• Cost estimation software	• Reverse engineering, • Design recovery, • Debugging, • Static analyser	• Code analyser, • Design recovery, • Reverse engineering tools	• Test tools
Communication					
Status		 • Premature termination			• Impl. plan • development process

Table 1: Tasks of Software Maintenance Process Infrastructure

Lack of input information to the tasks may lead to an unsuccessful project termination or premature termination.

Toolsets and methods

Each task is implemented by defined methods, in turn, the methods are supported by automatic tools and human interactions with the process. Tools must support the underlying methods of a task to accomplish the objective of the task. Automated tools, in some cases may not support methods; these should be supported by human intervention. Tools must be able to use the available data format in the task, and produce data which in turn will be used by the subsequent tasks. Many available tools can be used for software maintenance, but they may not be aimed at the maintenance process specifically. Automated toolsets, manual procedures, or a combination

of both can support tasks. However, most tool environments in software engineering have deficiencies in terms of integration and coordination of the tools with projects (Sharon *et al.* 1997).

The associated tools must be applicable to a wide range of projects. It is certainly desirable to extend the adaptability of the tools to keep the process operative for a longer time. Toolset must fit into the cultural context of the maintainers. It must support methods and techniques used by the programmer. Several tool classification schemes exist, but very few of them are intended to maintenance tools. More work on software maintenance tools can be found in (Durant 1989, Holbrook e*t al.* 1987, Khan *et al.* 1997).

Communications

The communications between tasks are done through the feed-back and feed-front loops to express the cycle of the model. The link arrow from one task to another indicates that the first one must follow the other. The feed-back loop is expressed by the ⟵ arrow, and the feed-front loop is shown by the ⟶ arrow. A line without any arrow shows the supporting component of the tasks.

Status

Status of the entire maintenance process can be a premature termination of the process, or a transition to a separate process. The process termination, or transition to the development process is expressed by the ◆ arrow.

SUPPORTING TASKS

We now briefly describe each of the tasks defined in our maintenance process as shown in Table 1.

Modification Requirements Analysis

This task is considered as the trigger of the process. The process enactment occurs by activating this task. In this task, requests from users or from the system itself are received. These may include adding new functions, improving the performance of existing functions, migrating the system to other operational platforms, modifying the existing function, or correcting faults in the system. These requests are analysed in detail so the entire modification requirements can be well understood both by the user and the staff involved in the maintenance project. Maintenance requirements certainly are different from that of a new development, therefore, it is important to analyse the maintenance requirements from a different perspective.

Determination (management decision)

Whether the software can absorb the changes successfully or not must be tested before implementation of the changes. This is a constraint mechanism in the maintenance process. This task examines the technical and economical feasibility of the project based on the requirement specification and the characteristics of the candidate software product. In this task various management issues are analysed such as how much effort will be required to implement the maintenance requirements. The answers to this fundamental management issue will determine whether the maintenance project should go ahead or not. Thus, cost to benefit ratio and the merits

of technical aspect required for the project are considered as the determinant of the maintenance process. The user requirements are refined and filtered at this stage. It actually includes estimating cost, availability of resources and tools. A *feedback* loop exists between this task and the previous task for a finer granularity of the requirements. If the project is found feasible, the *feed-front* loop will be activated. If the project fails to meet certain criteria set by this task, the entire process will terminate by activating the *exit loop*.

Program comprehension

If the design documents are missing or unreliable, or the original designers are not available for consultation, the entire program architecture must be understood by the maintainer programmers. Automatic or semi-automatic program comprehension tools are required to aid the task. Program understanding is not only important for the task, but also subsequent tasks require the full understanding of the source code and the domain knowledge of the system. In most cases, there is no historical track of how the product was actually developed and why certain types of design were crafted (Curtis 1992). Comprehension of the source code requires code reading, program execution, and the use of existing design documents. In such a situation, the assistance of automatic or semi-automatic program understanding tools like reverse engineering or re-engineering can be sought.

Localisation and impact analysis

If the maintenance project involves modernisation or corrections, then the exact location in the source code where the proposed modifications are to be made is identified. The most difficult task in introducing a change to software is to detect the ripple effects of the proposed change to other parts of the system. An unforeseen side effect on other parts of the code may occur when a change is introduced. This can be augmented either through variables or values, or through parameter passing. The project members must understand the impact of every change they are introducing.

Software maintenance in most cases suffers with ripple effects due to code modifications. It is often reported that corrections in the software may contribute to create more errors in the code (Humphrey 1989b). This is an important and difficult phase in software maintenance. Without proper attention to and mastery of the candidate system design, it would be hard to find out how the intended modification can cause side effects in the system, and where these would occur (Bohner *et al.* 1996).

Generating test cases

The test cases are designed to test the proposed changes in the maintenance process. The test data should include a wide range of possible data. Test paths and the regression testing procedures are to be defined. Actual testing takes place after the implementation of the proposed changes to the system.

Implementation Plan

In this task, implementation of the proposed modification is planned based on the output produced by other tasks. It includes how to update the existing specification and design documents, and how to re-code and configure the new and modified components of the system.

This is the final phase in the maintenance process. When this task is completed the usual development process is activated: the maintenance process triggers the requirement analysis phase of the development process. It is important to note that our maintenance process actually does not implement any features in the existing software product. Ways in which the implementation details of the maintenance process can be carried out in conjunction with the development process in a complete life cycle setting is discussed in the next section. In fact, in the present context, maintenance is seen as a continuation of the development process that begins the moment a software product starts its operation. It has been claimed that a significant part of software maintenance is in itself the development of new functions (Wild et al. 1991).

All information produced by the tasks is stored in a database repository. A Repository or a database plays a central role in storing the data and keeps all data for the projects. It is usually an active database that reacts to certain activities performed to its data (Froehlich *et al.* 1995). It facilitates human understanding and communication regarding the project. Hypertext technology can be used to present multiple views of the knowledge structure. The repository can also expose the expected behaviour of the task and the actual behaviour that it has performed. The discrepancies between the expected function and the actual performance of the tasks can be easily traced from the repository as well as. And the process behaviour could be tuned accordingly. All tasks have access to this repository.

However, the tasks cited in Table 1 show the process of software maintenance only. The maintenance process needs to be integrated into the entire software life cycle which will show how it can synchronise with the development process as well. In such a model, it is important to show that the development environment can support the underlying maintenance methods and activities within a software development life cycle framework. A complete and mature life cycle model must satisfy both the development and maintenance processes.

MAINTENANCE PROCESS AND SOFTWARE LIFE CYCLE

The concept of a software life cycle is a model used to describe and explain the software development and the maintenance processes. But within the software engineering context, it has not been yet well established how software maintenance would relate to the software development process, and how these two differ from each other. Software life cycle models presented so far fail to focus on many fundamental aspects of the evolutionary nature of software products. Researchers have proposed a number of software life cycle models partly or completely ignoring the phase of software maintenance. Most of these models do not provide clear guidelines on how to integrate the maintenance into a development process.

There are reasons to believe that software maintenance is a separate process, but it is essentially related to software development as some of the tasks are fundamental for both processes, and information from the development process is always needed in the maintenance process. The intersection of activities between the software development and maintenance processes is shown in Figure 2.

The top-down waterfall model (Royce 1970, Boehm 1976) has been widely accepted by the software community while the spiral model (Boehm 1988) has received considerable attention. The latter emphasises the risk analysis aspect of a software project, while the former views maintenance as a single phase in the post development chain. Software development and software maintenance, the two processes in software engineering, constitute a cycle for the entire life span

of a software system.

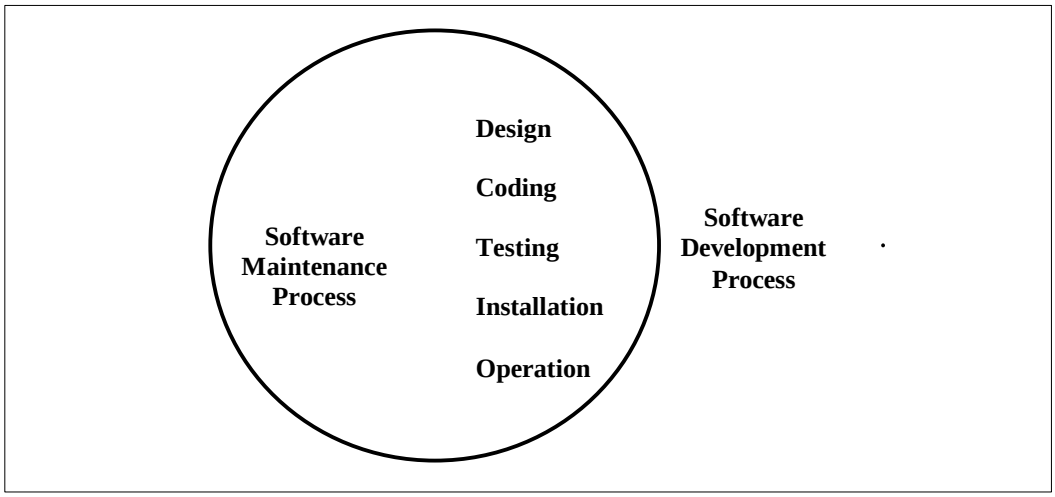


Figure 2: Intersection between software maintenance and development processes

There is a need for a well defined software maintenance process for the practitioners supporting a more complete software life cycle (Foster 1992, Chapin 1988, Rombach *et al.* 1988). It has been pointed out that a well-defined process should have the ability to blend maintenance and development homogenously into a single cycle (Curtis 1992). After integrating our maintenance process with the development process, the resulted scenario of the complete life cycle model is illustrated in Figure 3. This high level life cycle scenario focuses on phenomena that occur during software construction and reconstruction. Thus, this can be used in both the software maintenance context as well as the software development context.

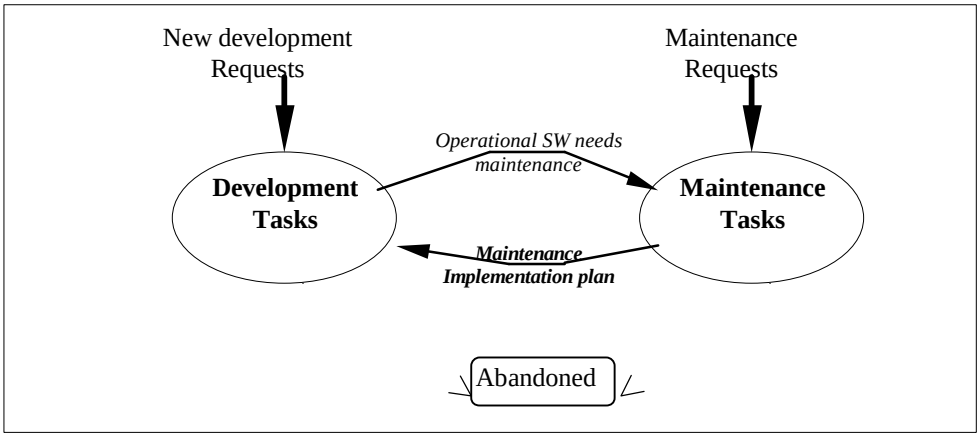


Figure 3: A High level software life cycle model

The life cycle shown in Figure 3 can be interpreted as a generic software re-engineering model. It suggests that the re-engineering process begins with the existing system, and produces a new form of the old system. This approach may be seen as similar to the spiral model, where the development cycle is repeated in ever increasing spirals.

COMPARISON WITH OTHER MAINTENANCE MODELS

The software life cycle model expressing the maintenance process integrated with software development proposed by Skramstad *et al.* (1992) shows similar steps but in a reverse time sequence. The phases of two processes are identical to each other as presented in their model. Theater Software Maintenance Environment (TSME) model (Cherinka *et al.* 1994) shows how the software maintenance process can be supported and automated through the use of an integrated software engineering environment. This high level process model does not address how this automated maintenance environment would be related to the development life cycle model. Ripple effects analysis and localisation activities which are important for a maintenance process are not explicitly identified in the model.

The maintenance model proposed by Makoto Ino (1992) is very similar to the development process model. The five phases of this maintenance model are: analyse user's maintenance requirement, design and approval, implementation, testing, and installation. This work does not actually focus on how a phase is actually constituted.

A generic maintenance process model adopted within the ESF/EPSOM project (Harjani *et al.* 1992) does not include an important maintenance phase like 'program comprehension'. This model comprises eleven main activities. It is not clear how these activities are supported by tools, methods and input information. The cyclic nature of the entire life span of a software product including the development process is missing in this model.

The Maintenance Assistance Capability for Software (MACS) (Desclaux 1991) basically concentrates its activities on program understanding process of the existing application software. It covers the phases of reverse engineering, modification management, and ripple effect analysis.

The Spiral Model (Boehm 1988) claims to cover software development and software maintenance coherently. However, the model does not explicitly integrate the two processes, and does not explain which information is shared between the development process and the maintenance process, nor does it show how the information is interchanged between the two processes.

The IEEE Standard 1219-1992 (IEEE 1992) describes a process for managing and executing software maintenance activities. The process model is described as a sequence of process task: *Problem identification, Analysis, Design, Implementation, System test, Acceptance test,* and *Delivery*. Each task has specific Control, Input, Process, Output, and Factors/Metrics. Each step is described in detail, and the input/output is clearly identified. However, in the IEEE standard, a clear interaction between the development and maintenance processes is lacking with respect to information from the software development process to maintenance, and the activities related to program understanding is not explicitly covered. The strength of this standard is the inclusion of the supporting issues such as planning, verification and validation, risk assessment, quality assurance, configuration management, and software metrics. The standard also has specific references to toolsets for each of the activities.

Most of the maintenance process models presented in various forums do not address how each phase will be constituted, and what is their relationship with the development process in a fuller life cycle setting. The associated tools, methods, and input-output information are not often represented in the model.

CONCLUSION

The software maintenance process and its relationships with the development process as presented in this paper provide software organisations with a task-oriented framework on how to control their process for maintaining software. The proposed framework includes various tasks, methods, input-output information sources, and communication protocols between tasks related to software maintenance process. Finally, the paper has made an attempt to define the intersection of common tasks between the software development and maintenance processes. This model could be related to other existing development processes in the organisation. The tasks may be tuned to be compatible to the needs of the projects. It may require constant monitoring to repair the model faults during the real maintenance project management. There should be a provision to accommodate for the changes the way the process works. The software maintenance process can be dynamic depending on the types of maintenance. To carry out a maintenance process it is important that the process is capable of sustaining the changes it needs. If the nature of process activities changes, then associated tools may change, also in certain extent the methods too. This framework is intended to be flexible to accommodate more tools and methods, and the communication protocols between tasks can be adjusted if required by the maintenance project.

REFERENCES

Boehm, B. W. (1976), 'Software Engineering', *IEEE Trans. on Computers*, Vol. C-25, December, 1976, 1226-1241.

Boehm, B. W.(1988). 'A Spiral Model of Software Development and Enhancement', *IEEE Computer*, May 1988, 61-72.

Bohner, S. A., and Arnold, R. S. (1996). :*Software Change Impact Analysis*, IEEE Computer Press, 1996, 1-376.

Chapin, N. (1988), 'Software Maintenance Life Cycle', *IEEE Proc. Conf. on Software Maintenance*, 1988, 6-13.

Cherinka, R., Overstreet, C. M., Cadwell, A., Ricci, J. (1992). 'Issues in Software Process Automation -From a Practical Perspective', *IEEE Proc. Conf. on Software Maintenance*, 1994, 109-118

Curtis, B. (1992). 'Maintaining the Software Process', *IEEE Proc. Conf. on Software Maintenance* 1992, 2- 8

Desclaux, C., and Ribault, M., (1991). 'MACS: Maintenance Assistance Capability for software A K.A.D.M.E.', *IEEE Proc. Conf. on Software Maintenance*, 1991, 2-12.

Durant, J. (1989). 'Classifying Software Tools', Software Maintenance News, Vol. 7., No. 3, December 1989, 16.

Foster, J. (1992). 'Survey Report', *European SIG in Software Maintenance Newsletter* Issue 3: June 1992. 5-7.

Froehlich, G. and Sorenson, P. (1995). 'Providing Support for Process Model Enaction in the Metaview Matasystem', *IEEE Proc. Seventh International Workshop on CASE*, Toronto, July, 1995, 141-149.

Harjani, Del-Raj, Queille, Jean-Pierre (1992). 'A Process Model for the Maintenance of Large Space Systems Software', *IEEE Proc. Conf. on Software Maintenance* 1992.127-136.

Haziza, M., Voidrot, J. F., Minor, E., Pofelski, L., and Blazy, S., (1992). 'Software Maintenance: An Analysis of industrial Needs and Constraints', *IEEE Proc. Conf. on Software Maintenance*, November 1992, 18-26.

Holbrook, H. B. and Thebaut, S. M. (1987). 'A Survey of Software Maintenance Tools that Enhance Program Understanding' , SERC-TR-9-F, University of Florida, September 1987.

Humphrey, W. (1989a). *Managing the Software Process*, Addission-Wesley, 1989.1-494

Humphrey, W. (1989b). 'Quality from Both the Developers and User Viewpoint', *IEEE Software* , September 1989, 84-100.

IEEE (1992). IEEE Standard 1219-1992 , *IEEE Standard for Software maintenance.*

Ino, M. (1992). 'Current State of Software Maintenance in Japan: In Depth View', *IEEE proc. Conf. on Software Maintenance* 1992. 27- 29.

Khan, M. K., Ramakrishnan, M. K., and Lo, W. N. Bruce. (1997). 'Assessment of Software Maintenance Model: A Conceptual Framework*', Proc. Pacific Asia Conf. on Information Systems (PACIS'97)*, QUT, Brisbane, April 1997, 527-536.

Khan, M. K., Rashid, M. A. and Lo, W. N. B. (1996). 'A Task-Oriented Software Maintenance Model', *Malaysian Journal of Computer Science*, Vol. 2, December 1996, 36-42.

Lehman, M. M. (1980). 'Programs, Life Cycles, and the Laws of Program Evolution', IEEE *Proc. Conf. Software Engineering*, 1980, 1060-1076.

Lehman, M. M. (1987). 'Process Models, Process Programs, Programming Support', *ACM Proc. 9th Int'l Conf. on Software Engineering*, 1987, 14-16.

Notkin, D.(1993). 'Software Evolution', *ACM SIGSOFT Software Engineering Notes,* Vol.18, No.1, January 1993, 47.

Rombach, H. D. and Basili, V. (1988): 1A Panel Discussion, Position Statement, *IEEE Proc. Conf. on Software Maintenance*, 1988.

Sharon, D. and Tracey, A.(1997). 'A Complete Software Engineering Environment', *IEEE Software*, 1997, March/April 1997, 123- 125.

Skramstad, T. and Khan, M. K. (1992). 'A Redefined Software Life Cycle Model for Improved Maintenance', *IEEE Proc. Conf. on Software Maintenance 1992*. 193-197

Royce, W. (1970). 'Managing the development of large software systems', *Proc. WESCON*, August 1970.

Wild, C., Maly, K., and Liu, L. (1991). 'Decision-Based Software Development' *Journal of Software Maintenance*, Research and Practice, John Wiley & Sons, vol. 3, March 1991, 17-43

COPYRIGHT