Journal of Software Engineering

ISSN 1819-4311

Academic Journals Inc.

www.academicjournals.com

Journal of Software Engineering 5 (1): 20-37, 2011
ISSN 1819-4311 / DOI: 10.3923/jse.2011.20.37

Measurement-based Software Process Modeling

[1,2]Wen-Hsiang Shen, [2]Nien-Lin Hsueh and [2]Peng-Hua Chu
[1]Department of Information Management, Overseas Chinese University, 100, Chiaokwong Road, Taichung 40721, Taiwan, Republic of China
[2]Department of Information Engineering and Computer Science, Feng Chia University, 100 Wenhwa Road, Taichung 40724, Taiwan, Republic of Chaina

Corresponding Author: Wen-Hsiang Shen, Department of Information Management, Overseas Chinese University, 100, Chiaokwong Road, Taichung 40721, Taiwan, Republic of China

ABSTRACT

Software process modeling has been recognized as an important topic since the early days of software engineering. Process modeling language plays a crucial role when it is used to define and analyze the processes. Complex processes have many elements and there are many potential ways to improve them. Without a quantitative understanding of the process steps, it is difficult to tell which ones are effective. Processes can be measured for size, effort, schedule and cost under successful performance. A process should be defined based on organization's intents or business goals. Although current studies on software process improvement have aroused interest of many researchers in this field, software development strategies do not provide a clear indication for constituents of software processes. Under this circumstance, an approach based on Unified Modeling Language (UML) was proposed in the present study to define a software process as an effective way for identifying and defining the essential process elements of good evaluating practice. The study aimed to embed quantitative evaluation faculty into project's process definition stage and satisfy the goal of organization process improvement. The proposed approach focused on the study of UML static models as class diagrams complemented with a set of Object Constraint Language (OCL) constraints.

Key words: Unified modeling language (UML), process modeling, software process improvement, practical software measurement (PSM), object constraint language (OCL)

INTRODUCTION

Many researchers have recognized the need for software process improvement since the 1930s (Chrissis *et al*., 2003). Software process improvement applied to process modification in software development can improve product quality, decrease costs and increase profitability. In order to achieve software development process standardization, models for software process quality or standards for software quality assurance have been proposed by previous researchers, including CMMI (Chrissis *et al*., 2003), ISO15504 (Van Loon, 2007), ISO15539 (ISO, 2002). The technology for software process enactment and control was also focused (Fuggetta, 2000).

The kernel of modern quality management can be defined as process management. As the definition of standards mentioned above, process is a set of relative resources that can activate and transform input to output. In the present study, the processes can be divided into two classes. One is the evaluation process and the other is software development process. Generally, the evaluation process is at the organization level, such as internal process quality audit, work product and process information collection, analysis, evaluation and so on. The software development process is at the

project level, such as requirement analysis, design, implementation, testing and so on. Certain time order or predication relationship among these factors exists. Hence, computer designers are suggested to care about the evaluation of process rather than the process method. For example, Fadila and Said (2006) presented a textual approach for software process modeling. A kind of Petri net extended with time, resource and message factors is proposed to model tasks and emergency response process within a single organization (Meng *et al*., 2011).

The goal of process evaluation focuses on continuous improvement of software process driven by the implementation of software process metrics. As a result, process metrics plays a major role in the process capability assessment and its management (Muketha *et al*., 2010). Based on requirement of CMMI level 4, the researchers presented a process meta-model with software process measurement which defined the roles, activities, relative meta-classes and methods of process metrics implementation. Meanwhile, this model can also achieve the goal to improve process by means of obtaining information needs and the feedback for the process manager. Under this circumstance, it can provide a method of commonly used software process data for validating and analyzing.

The success of software development lies in the degree of efficiency and quality of the software development processes that lie beneath the core abilities of an organization (Petty, 1996). Successful organizations are constantly improving their processes and stay ahead by taking advantage of familiar technologies while the followers are not far behind in catching up with the frontier firms. Organizations that are not in the race soon cease to exist in global competition today. Hence, there is a greater need for managers and consultants to not only know how to design and analyze a development process but also identify ways to improve the process and sustain the improvements in the long run to meet the organizational objectives more effectively (Chrissis *et al*., 2003). For this reason, the process measurement of current process is important to understand the efficiency of process and degree of organization satisfaction (Kan, 2002).

Software process improvement (McFeeley, 1996) is a common term used for modifying processes in software development that aims to improve product quality and business value. Successful improvement of the development process should take advantages of complementary use of both quantitative and qualitative methods. Complex processes have many elements and there are many potential ways to improve the processes. Without a quantitative understanding of the process steps, however, it is difficult to tell which ones are effective. Process measurements provide the data you need to objectively understand how your process works and to see what you can do to improve it (Bobkowska, 2001).

The process measurement involves the collection of data about time, size and defects. Such analysis can improve software process quality (Humphrey, 1995). Defining a process lies in organization's intents that are composed of business goals (Koulopoulos, 1995). Yet, process definitions still suffer from the following drawbacks:

- Currently the processes in organizations are closely related to the fixed forms. The changes to the forms and processes are considerably difficult so that the text actually advises against doing so (Chrissis *et al*., 2003; Fadila and Said, 2006)
- The process measurement method, forms and procedures are not concurrently designed to support a process enactment. For this reason, one must immediately redesign the forms and scripts embedded to process definition in order to apply the proper metrics to any other kind of work product (Feiler and Humphrey, 1992)

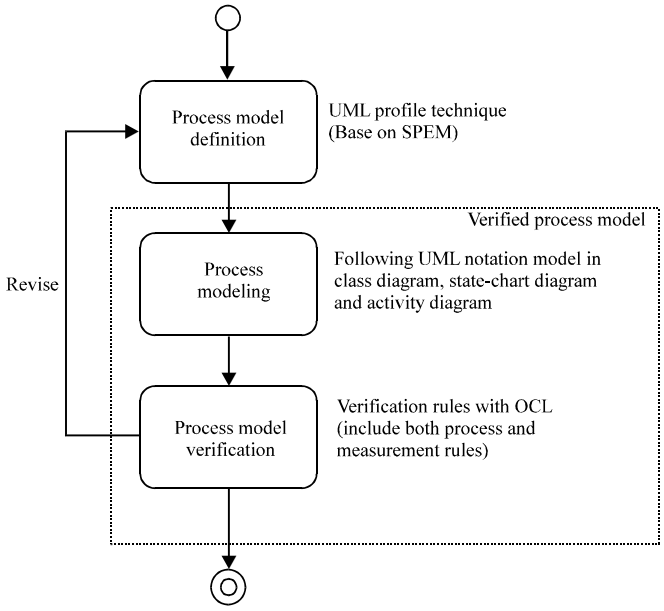

Fig. 1: Approach overview

As the integrated process measurement for the process modeling is required, obtaining high process data quality in the analysis stage may not be helpful. Many researchers indicated that integrated measurement support will improve the process more easily (Garcia *et al*., 2003; McGarry *et al*., 2002). Since the processes must be validated against user's needs, limited prototypes may be needed before full-scale development is undertaken. As Kheirkhah *et al*. (2009) claimed, a screen-based prototyping technology can be used to narrow the gap between end-users and software developers. Regardless of the degree of verification, no process defects and process efficacy are found before general process instantiation and enactment. Because software processes typically require at least partial human enactment, there is a major requirement as well as usability problem and it is much more difficult to do effective verification without end-user involvement (Feiler and Humphrey, 1992).

This study focused on verifying, measuring and improving processes for resource and production in general. The study thus aimed to (1) create a clear understanding of various development processes, (2) identify potential improvements (3) learn existing techniques for performance enhancement and (4) design a methodology of process modeling with efficiency measurement to implement the process improvement strategies.

Figure 1 depicts an overview approach in the present study. Using our approach, an organization develops its processes in the following steps; that is, process model definition and process model verification. The model in the present study is easy to use and verifiable due to the popularity and formality of UML (Jager *et al*., 1999; OMG, 2003a). Embedding a set of process related to rules and measurement mechanism in the profile can help verify if the developed process definition meet the basic requirements of the process framework. For example, one of the rules is all entry criteria in a process must be satisfied. The present approach can help verify this rule with the aid of Object Constraint Language (OCL) (Evans *et al*., 1998). Process model verification also provides an opportunity for modelers to validate the appropriateness for the developed process before applying the process. If the process is not suitable for operation, it is recommended that the designer go back to the definition stage before revision of OCL rule. For example, the defined

process can be revised when we neglect to define that the unit test which should be performed in a code development phase. Therefore, while or after defining, the researcher can do process verification to verify whether the modeled processes are consistent with the rules defined in the process profile (Hsueh *et al*., 2008).

THEORETICAL BACKGROUND

In recent years, a growing number of software organizations have begun to focus on quantitative management which implies an understanding of variation. Some relevant researches are reviewed in this section, including Practical Software Measurement (PSM) (McGarry *et al*., 2002), Software Process Engineering Meta-model (SPEM) (OMG, 2002), Workflow (Casati *et al*., 1995; WMC, 1995; WMC, 1999), OCL (OMG, 2003b), process modeling approaches and techniques (Jager *et al*., 1999; Petty, 1996; Salimifard and Wright, 2001; Van der Aalst, 1998).

Functions of measurement: Measurement can be an effective way because it provides information for project managers to monitor key issues related to quality and monitor performance compliant with a plan. In other words, measurement not only allows managers to make the exact decisions based on objective information but also provides the information management for process evaluation and sense of the quality to develop a project. Although it cannot guarantee the successfulness of a project, it does help all decision makers take an early warning to manage critical issues inherent in software projects.

According to Ishigaki and Jones (2003), measurement helps managers complete the following tasks:

- Communicate effectively
- Identify and correct problems early
- Make informed trade-offs
- Track specific project objectives
- Manage risk
- Defend and justify decisions

Actually, performing measurement activities contains performing and integrating the data collection (Base Measure), performing the analysis, aggregation and metric (Derived Measure), generating the information that will present the analysis results and communicating the information products to the measurement users (Indicator).

Software Process Engineering Meta-model (SPEM): Software Process Engineering Meta-model (SPEM) is a UML extension mechanism used to describe a concrete software development process or a family related to software development processes defined by the OMG (2002). It takes an object-oriented approach to model different facets of software processes, such as use case, class and state chart diagrams, Simply stated, UML consists of a set of diagrams that allow modelers to examine a software process from several different points of view prior to running the process. Based on the UML approach, Changchien *et al*. (2002) proposed a Hierarchical Aggregation Model to evaluate the object-oriented software design quality. SPEM is integrated in the pyramidal architecture of MDA organization as a Meta-Object Facility (MOF) meta-model and as a UML profile (Atkinson and Kuhne, 2002). SPEM is based on the idea that a software development process is collaboration between active, abstract entities, (roles which perform

operations called activities) and concrete, real entities (roles which perform operations called work products). The different roles act upon one another or collaborate through exchanging products and triggering the execution of certain activities (Breton and B'ezivin, 2001).

SPEM is dedicated to software processes modeling. Many features of the UML provide necessary bases for modeling processes and many other UML features provide useful additional modeling capacities. However, as SPEM is a large specification, it is not easy to apply to general context. In addition, as SPEM is used for general purpose, it cannot offer any support when defining the processes for measurement. To resolve these problems, this research focused most parts on this study.

Workflow: The concept of workflow has been developed from the notion of factory automation (Koulopoulos, 1995). Its history traced back to the 1970s when the first prototype of workflow (Zisman, 1997) was developed. Workflow management refers to an approach to the problem of controlling, monitoring, optimizing and supporting business processes (Van der Aalst, 1998). A workflow itself is a process in which documents, information or tasks are passed from one participant to another. It is a set of activities involving the coordinated execution of multiple tasks performed by different processing entries (Casati *et al*., 1995) and covering the flow of information as well as control within and between organizations. The Workflow Management Coalition (WMC, 1995, 1999) defines a workflow as a computerized facilitation or automation of a business process, in a whole or part. A specific software system, called WfMS, controls automated aspects of workflow. It defines, manages and executes workflow through the execution of software whose order of execution is driven by a computer representation of the workflow logic (Mangan and Sadiq, 2002). Most existing WfMSs have evolved from some computer-based systems such as database management system, document imaging or electronic mail. Since the late 1980s, workflow has become a motivating and challenging research field.

Functions of Object Constraint Language (OCL): There is increasing interest in the UML community as introducing more formality into models using various diagram of UML. OCL defines models as the likes of PMP (Hsueh *et al*., 2008). Most formal specification languages applied in requirements can design phases to write the requirements precisely at a high level of abstraction. They help a developer gain deeper understanding of system requirements and constraints. Meanwhile, these languages also assist in capturing design problems and other inconsistencies during specification before implementation effort within such processes. OCL also prevents errors and provides a basis for the automated support in the process of definition. The semantics of the formalism provides precise rules of interpretation that may overcome many problems with natural languages. In other words, modeling the domain and elaborating the goals, the program designers should understand that requirements on the software and assumptions about the environment are inevitable. In addition, the modeling tool with OCL may not only help design a functional model for the software or the software architecture, but also modify or reengineer the software.

OCL can be used for various purposes, including model verification and validation, code generation, test driven development, transformations and application domains, such as domain specific languages and web semantics. In this paper, the researcher proposed a process meta-model profile from a subset of OCL to process modeling and used this profile for checking the unsatisfied ability of sets of OCL constraints.

The researcher also proposed a quantitative model for implementation of SPI. These ideas are among the principles underlying the SPEM and PSM methodologies such as (OMG, 2002) and (Cook and Wolf, 1999).

A FRAMEWORK FOR PROCESS MODELING

The organizations of software have found that by defining processes, the effectiveness of software development can be improved (Humphrey, 1995). Software process definitions make high-quality software easier and more economical to produce with the hope to help them become widely valued and used.

Since process development is expensive, there is considerable motivation for process commonality and sharing. It can be enhanced by the fact that different projects in the same organization will likely have many common activities. Process in the context of large organizations can be defined as a procedure for the participation of various stakeholders who share with other related people and share in the process of management. The more logical and explicit definitions or relationship they can provide, the more likely the common elements of the project processes can be found.

Another question was concerned about the degree of process refinement. A complex software process can be regarded as a nested set of abstractions. Each process is composed of several sub-processes and each of which in turn may have smaller elements. While there is no clear technical limit to the level of refinement for a software process, there are still certain practical concerns, including:

- The scale of the projects for which the process is designed
- The degree to which the work is partitioned among implementers
- The resources and time available for process definition
- The level of capability or understanding of the process users
- The scalability of the process itself

In this study, PSM is designed to put measurement into practice at the project level, thereby providing the data addressing enterprise level performance, process improvement and business-related questions. In this research, the evaluation measurement activity applies measurement and analysis techniques to the measurement process itself. It encompasses the assessment of both the applied measures and the capability of the measurement process and it helps to identify associated improvement actions.

Process structure: Figure 2 describes the meta-model of the static process structure with measurement framework. The modeling elements present a basic organization's process structure, including process elements and their relationships. Process is a kind of operation that describes the work performed in the process. Its main subclass is Practice. A Practice may be a subProcess and may consist of a set of atomic elements called Step. For example, a Process-Defect Rate Control contains a subProcess-Debug. Debug contains a Step-code review. Each Process covers at least one EntryCriteria and one ExitCriteria. The EntryCriteria of a Defect Rate Control may be under the condition that product defect rate was too high. A ProcessMember defines a member in an organization. It has two subclasses: ProcessOwner and ProcessPerformer. A ProcessPerformer defines a performer as a set of Practices in a process. A ProcessOwner defines an owner as a set of Processes.

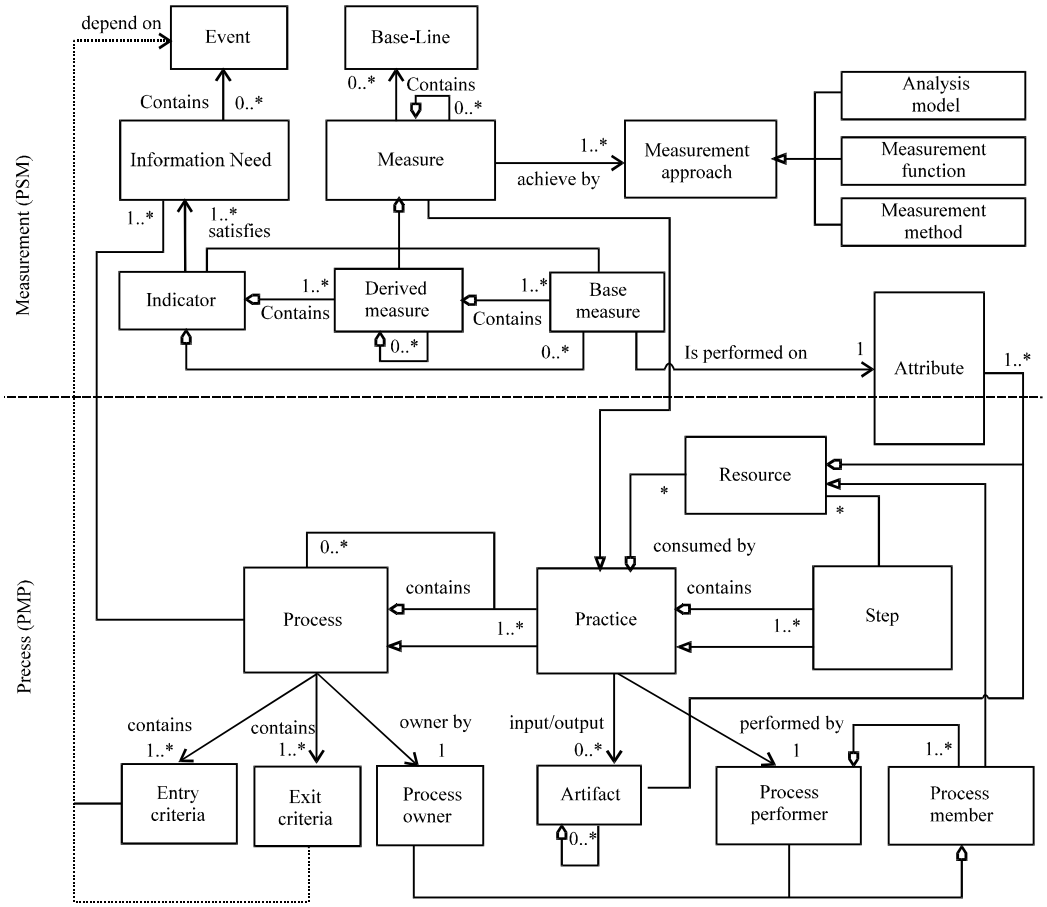

Fig. 2: Static process meta-model with measurement feedback

The modeling elements above the dash line in Fig. 2 are the elements related to PSM framework which is an overview of how an information need evolves into a plan for generating the process's measurement products. It outlines the basic structure of a measurement construct. The measured things include specific attribution of software processes and products, such as size, effort and the numbers of defects. The measurement construct describes how the relevant software attribution is quantified and converted to indicators that provide a basis for decision-making. A single measurement construct may involve three types of measures, including base measures, derived measures and indicators.

Figure 3 shows an example of a PSM framework which represents a decision maker (McGarry *et al.*, 2002) and evaluates whether or not the rate of code generation on a project is sufficient to meet the objects of project schedule. Such framework also exhibits a process for information driven measurement that addresses the unique technical and business goals of an organization. The measurable concept is that progress is related to the amount of work planned and the amount of work completed. Planned work items and completed work items are the entities of concern. This example assumes that the degree of completion of each software unit is reported by the developer assigned to it. Thus, data for the base measures must be collected while data for the derived measures should be computed for each work item in the plan. A simple numerical threshold is used as a decision criterion rather than statistical limits. Since the status of units is a subjective assessment, the indicator is subject to the influence of dreaming on the part of the programmers.

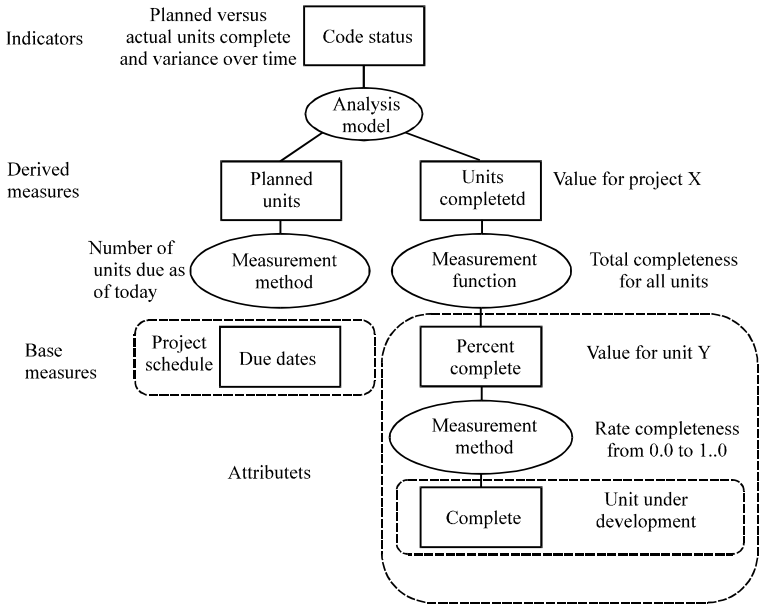

Fig. 3: An example of measurement construct-coding progress

Process profile definition: Process profile defines a number of useful predefined modeling elements and depicts how they interact with one another. These stereotypes are derived from UML entities (Heaven and Finkelstein, 2004), such as classes and associations. The stereotypes augment the semantics of these standard elements with newly defined meanings, allowing us to model previously unsupported concepts. The extension of process profile is carried out by using UML profiles. A profile can be defined by using stereotypes, tag definitions and constraints that are applied to specific model elements, such as classes, attributes, operations and activities. A profile also refers to a collection of some extensions and limitations for knowledge that collectively customizes UML for a particular domain. The main concept of extension in the profiling mechanism is the stereotype. A stereotype is one of three types of extensibility mechanisms in the UML. The extensibility mechanisms allow designers to extend the vocabulary of UML in order to create new model elements, derived from existing ones that have specific properties that are suitable for a particular problem or other specialized usage. Modeling frameworks for particular problem domains can be easily specified in UML with profiles. Table 1 shows a summary of process profile stereotypes.

Based on the specification of SPEM, certain stereotypes for process profile were proposed as shown in Table 1. Stereotypes shown in the first column inherit stereotype parents as indicated in the third column. For instance, Practice is the main subclass of Process. It describes a piece of work performed by one ProcessPerformer: The tasks, operations and actions that are performed by a role or with which the role may assist. A Practice may consist of atomic elements called Step. Practice inherits from Process for the fact that it has input and output types of Artifact. A Practice is assigned by a ProcessPerformer that is the performer of the described activity. It may refer to additional ProcessMember that is the assistant in the activity. A Practice can be divided into the atomic work steps. A Step is depicted in the context of the enclosing Practice in terms of the ProcessPerformer and Artifact. In the measurement aspect, Indicator, DerivedMeasure and BaseMeasure inherit from Measure that can be achieved by In the measurement aspect, Indicator,

Table 1: Stereotypes of the process profile

Stereotype	Base class	Stereotype parent	Example
Artifact	Core::Class		
	Activity graphs::Object flow state		Customer requirements, source program
Entry criteria	Core::Constraint		Do defect prevention when poor quality happens (PQI<0.5).
Exit criteria	Core::Constraint		Stop defect prevention after PQI>=0.5.
Process	Core::Operation		
	Activity graphs::Action state		A self-improvement process
Practice	Core::Operation		
	Activity graphs::Action state	Process	Defect prevention, requirement analysis
Step	Core::Operation		
	Activity Graphs::Action State	Practice	Baseline requirements
Process member	Use cases::Actor		Project leader
Process owner	Use cases::Actor	Process member	Process definition team
Process performer	Use cases::Actor	Process member	Requirement analyzer
InformationNeed	Core::Class		Understanding the quality of personal development process
Measure	Core::Class		Measures and indicators
Base measure	Core::Class	Measure	Size measurement
Derive measure	Core::Class	Measure	Program quality
Indicator	Core::Class	Measure	Process quality index (PQI)
Event	Core:: Signal		With poor quality due to post development defects
Measurement approach	Core:: Model		Defects density measured
Analysis model	Core:: Model	Measurement approach	PQI=DQ*DRQ*CRQ *CQ*PQ [1]
Measurement function	Core::Class	Measurement approach	Minimum (1.0, design time/coding time)
Measurement method	Core::Class	Measurement approach	Count the amount of source program (LOC [1])

[1] Note: PQI (Process Quality Index), DQ (Design Quality), DRQ (Design Review Quality), CRQ (Code Review Quality), CQ (Compiler Quality), PQ (Product Quality), LOC (Lines of Code)

DerivedMeasure and BaseMeasure inherit from Measure that can be achieved by Measurement approach, respectively. AnalysisModel, MeasurementFunction and MeasurementMethod inherit from Measurement approach. The blank cells in the third column of Table 1 indicate that no inherent interrelationship between both types was found in the present study. The profile of Rational XDE (eXtended Development Environment) stored as an XML (eXtensible Markup Language) file was built and could be applied to a process model to customize the present modeling environment.

Process measurement: Process measurements aim to measure quantitative data for the software process. Humphrey (2005) suggests that measurement plays an important role in small-scale, personal process improvement. Process metrics can be used to assess whether or not the efficiency of a process has been improved. For example, the effort and time devoted to changing request process can be monitored. Product metrics must also be collected and related to the process activities. Three classes of process metrics can be collected:

- The time taken for a particular process to be completed. This can be the total time devoted to the process, calendar time, the time spent on the process by particular engineers and the like
- The resources required for a particular process. The resources might represent total effort in labor cost, costs for raw materials, computer resources and the like

- The number of occurrences of a particular event. Examples of events that might be monitored include the number of defects discovered during code inspection, the number of requirement changes requested, the average number of lines of code modified in response to a requirement change and so on

The crucial difficulty in process measurement lies in the validity and reliability issues, including what is supposed to be measured and how reliable the data represent after the measurement. Basili and Rombach (1998) have proposed what they called the GQM (Goal-Question-Metric) paradigm. This is used to help decide what measurements should be taken and how they should be used.

PROCESS MODELING

As the basic three elements, process definition, validation and measurement were defined and identified when conducting software process improvement, another crucial issue that program users should consider is the use of process modeling language. Process Modeling Language (PML) plays an important role in topics related to software process improvement for its value of defining and analyzing processes in general context. In order to simplify the complexity of PML when applying for a general process enactment role of a software development, Unified Modeling Language (UML) was developed.

Basic UML diagrams can be used to present different perspectives of a software process model. Therefore, the researchers use these well-defined diagrams to present processes. However, the following UML notations must be constructed in this profile: class diagram, state-chart diagram and activity diagram. Under this circumstance, the researchers can model, present and understand processes with the use of this methodology.

Process verification: The UML is a general-purpose modeling language which is widely accepted for the specification of different aspects of software systems. Comparing to other formal process modeling languages like Petri net, UML is easier to learn due to its familiarity. The modelers do not need to learn new notation and semantics when defining processes. In spite of its generality, on the one hand, UML provides two kinds of extension based on the direct modification of the UML meta-model (by modifying any of the provided modeling elements). On the other hand, UML provides Object Constraint Language (OCL) (OMG, 2003a) for modeling precise and verifiable semantics (with profiles) to the implementation of domain-specific languages, such as embedded CMMI goals or measurement analysis modeling.

Process verification is the activity for verifying the correctness, completeness and consistency of a process. It is done automatically by OCL. OCL is a notational and navigational language to describe business rules and to constraint a model. OCL expressions return a value without changing the model or the state of elements in the model. The way to apply and practice OCL rules to define process constraints can be described in the following section.

Process profile rule: Every element of process profile is associated with validation rule. It inherits from UML meta-model. The rules are adopted to ensure the correctness of process model. If the rules were violated, they would generate error messages. The rules can be divided into Process-related rule and Measurement-related rule.

Process-related rule: ProcessPerformer is a specialization of Classifier related to the process definition. A ProcessPerformer defines a performer as a set of Practice in a process.

ProcessPerformer can assign Practice that has only one specific owner. A ProcessPerformer could be a given person who can act in several performers while several persons may act as a single given performer. Take the Rational Unified Process as an example, the ProcessPerformer can be an Architect, Analyst, Technical Writer and Project Manager to name a few (OMG, 2002).

Process-related rules can be utilized to constrain process structure. The following rule is the one that constraints process multiplicity. This rule indicates that a Practice is performed by only one ProcessPerformer. There is a Performed by relationship between Practice and ProcessPerformer. The multiplicity is constrained to keep only one relationship.

Rule: A practice is performed by exactly one processperformer

```
Context Practice inv:
 {self. Relationship ->select (oclIs Type of (Dependency)) -> select (k | k.Tagged Value Set Reference-
      >exists (Name = 'Performed by' and oclIs Type of (Stereotype))). oclAsType (Dependency). NonOwningEnd-
      > select (oclIsType of (Class))->select (p | p.TaggedValueSetReference->exists (Name = 'ProcessPerformer'
      and oclIs Type of (Stereotype))) ->size () =1
}
```

Measurement-related rule: Processes need continuous change and refinements to increase their ability to deal with the requirements and expectations of the market and of the company stakeholders. Hence, processes need unceasing assessment and improvement. An important part of process improvement is the process measurement (Shen *et al*., 2011). Quantitative metrics and parameters are usually obtained from the historical data. Most of the organizations may suffer from collecting data that need to establish the quantitative measurements and thus need the way to obtain the efficiency data for processing and indicators from runtime. In order to obtain efficiency data, the measurement construct is of critical and inevitable. The measurement construct describes how the relevant software attributes are quantified and converted to indicators that provide a basis for decision-making. A single measurement construct may involve three types of measures: Base measures, derived measures and indicators. The following diagram demonstrates the way to define measurement-related rules to assist organization in defining processes that meet information need requirements. These rules are measurement-related rules and comply with PSM discipline.

The measurable concept is that process quality is related to the program quality which can help produce the amount of design and figure out the number of defects. Hence, the lists of defects are the entities of concern. Computing the program quality can normalize the process quality allowing a derived comparison. Under this circumstance, data collected for the base measures and information inferred for the derived measures must be made for necessary information in each process (McGarry *et al*., 2002). For example, the following rule illustrates the necessary information for Process Quality Index under the constraint of the software development process. In order to obtain the information, project manager needs to accomplish the tasks for DerivedMeasure-Program Quality.

Rule: Process quality index should be satisfied by achieving derived measure (Program Quality)

```
meta OCL express:
{self.Relationship->select (oclIs Type of (Dependency))->select (k | k.TaggedValueSetReference-
      >exists (Name = 'completed' and oclIsType of (Stereotype))). oclAsType (Dependency). NonOwningEnd-
      > select (oclIsType of (Class)) -> select (p | p.TaggedValueSetReference->exists (Name =
      'DerivedMeasure_Program Quality' and oclIs Type of (Stereotype ) ))->size () = 1
      }
```

CASE STUDY

According to Humphrey (2005), there were only three basic PSP measures: size, time and defects. All other PSP measures were derived from these basic measures. Based on the framework, the present study was conducted to verify this approach and provide an example for process modeling in the measurement of Process Quality Index (PQI) in PSP process. Quality was then measured to improve process quality. The key questions then are these: what do you measure and how do you use the resulting data? Before using the data, the quality measures must be decided. The defect content of a finished product indicates the effectiveness of the process for removing defects. Thus, both the work and the work products need to be measured. The available work measures consist of defect-removal yield, cost of quality, review rates, phase-time ratios and the process quality index.

The process quality index is obtained by multiplying these five values together. The quality criteria of process quality index are as follows:

- **Design quality:** Design time/coding time, with a range from 0.0 to 1.0
- **Design review quality:** 2*design review time/design time, with a range from 0.0 to 1.0
- **Code review quality:** 2*code review time/coding time, with a range from 0.0 to 1.0
- **Compiler quality:** 20/(10 + compile defects/KLOC), with a range from 0.0 to 1.0
- **Product quality:** 10/(5 + unit test defects/KLOC), with a range from 0.0 to 1.0

The PQI goal, however, should be 1.0. As the value PQI<0.5 identifies the event (with the poor quality due to post development defects), the process of defect prevention can be achieved when the value PQI> 0.5 occurs.

Modeling static process structure with class diagram: Class diagrams describe the attributes and operations of a class and the constraints imposed on the process. They not only present process types and their instances but also show at least some of the relationships among and between these elements and potentially even show their internal structures. Moreover, class diagrams show the static structure of the process being modeled and focus on the elements of a process irrespective of time. A static structure is conveyed by showing the types and their instances in the process. Figure 4 exhibits the relationship of models between processes and measurement frameworks of process quality index.

Modeling behavior of process element with state-chart diagram: State-chart diagram describes the flow of control from one state to another state. States are defined as a condition in which an object exists and it changes when some event is triggered. State-chart diagrams can be used to illustrate the behavior of model elements, especially the behavior of Artifact. A transition is a kind of Process. This makes it easy to know what states these process elements are and where the process stands. Figure 5 shows an example of a possible state transition of work artifact, a Artifact (source code) possible state transitions. After doing Implementation, the state changes from Draft to Completed.

By using the tool like Rational XDE, the activity on the transition must be the operation defined in each process class. This provides a constraint to restrict erroneous definition of operation.

Modeling dynamic process sequence with activity diagram: Basically, an activity diagram refers to a flow chart which represents the flow from one activity to another. The activity can be

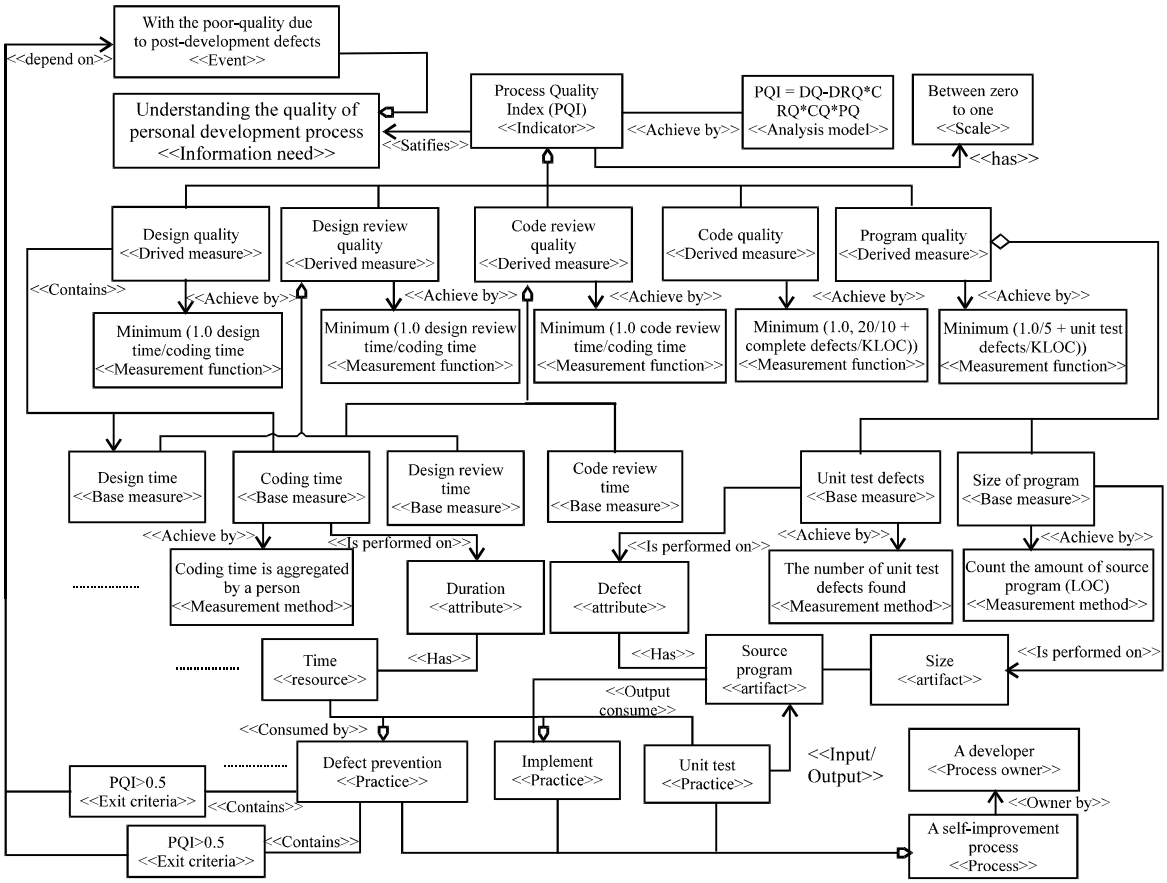

Fig. 4: Class diagram of a self-improvement process with PQI indicator

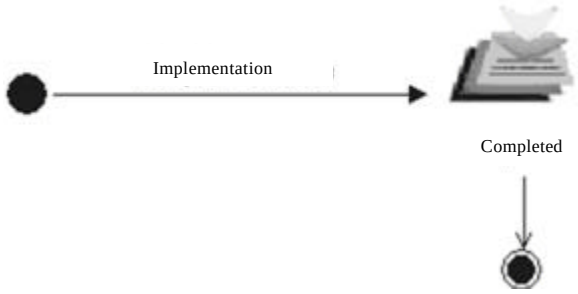

Fig. 5: The state transition diagram of the source code artifact

described as an operation of the process. It captures the dynamic behavior of the process. Such diagram also allows presenting the sequence of activities with their input and output artifacts as well as object flow states. An activity diagram is shown in Fig. 6. This diagram is drawn with the some main activities. The activities are built based on the application of PSP implementation, i.e., the phase of the software life cycle that includes planning, development, measurement and analysis.

Generally, the development activity covers the input Artifact Defect type standard and code standard with the states baseline, Size estimating template, Planning time recording log and A

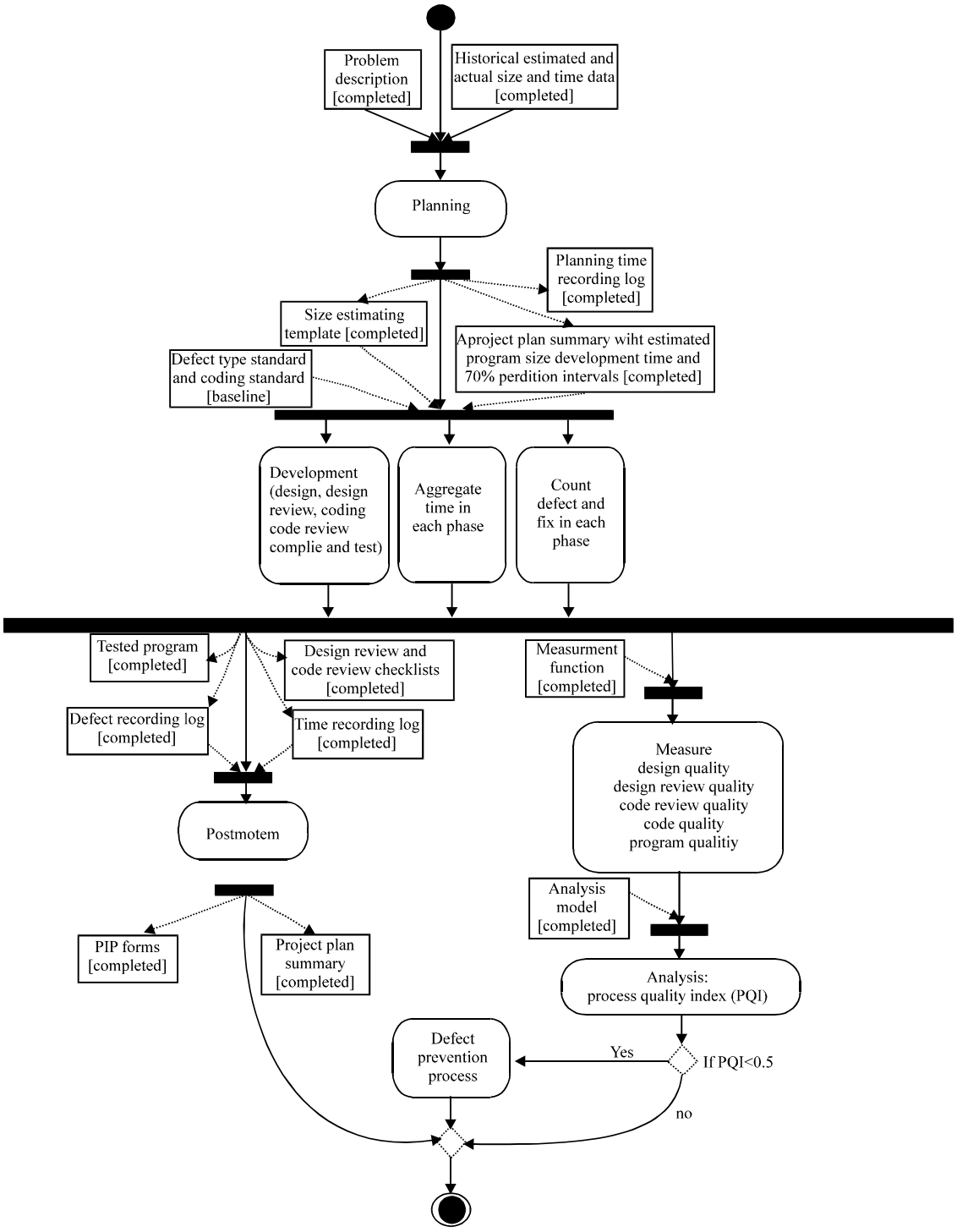

Fig. 6: Activity diagram of PSP with PQI's measurement

project plan summary with estimated program size development time and 70% perdition intervals with the states completed. The output Artifact-Tested program, Design review and code review checklists and Defect/Time recording log with the states completed. This development activity can be triggered when the planning activity has been completed. After that, the PracticeMeasure comes

before Analysis. After analyzing the results, condition checks are performed to check if the process is in normal or poor quality. Meanwhile, after the value of PQI is identified, a defect prevention process is performed or is marked as the termination of the process from the perspectives of measurement and analyses.

Note that the class modeling, state-chart modeling and activity modeling are iterative and interrelated. The EPG may start with class modeling to first identify process elements and their relationships. Then, it further uses state-chart modeling to describe possible state transitions of identified process elements. Finally, the activity modeling took place to identify dynamic process sequences. The benefits of modeling dynamic process sequences with activity diagrams are the constraints inherited from UML meta-model. Therefore, the modeling is not only easy to follow but also easy to integrate.

Since the elements in class diagram, state transition diagram and activity diagram inherit the same UML meta-model, their inter-relationship is easy to verify. For example, the object state source code [completed] in activity diagram refers to the state completed in state transition diagram and the object source code in the class diagram. Inconsistencies between diagrams can be therefore, identified.

The practices of process verification with measurement: The process verification is an automatic mechanism. The rules defined in this profile can verify the process automatically when using such profile to model targeted processes. As seen in Fig. 7, the Practice lacks an EntryCriteria in the model. Therefore, when carrying out the model check, the model tool will display error messages indicating which model is incorrect and in what manner. In the example, the modeller does not specify the EntryCriteria of the practice which violates a rule (The Process contains at least one EntryCriteria) in the present rule base. A successful process model that passes all verification rules will return a complete diagram without any error messages (Fig. 8). Following these steps of

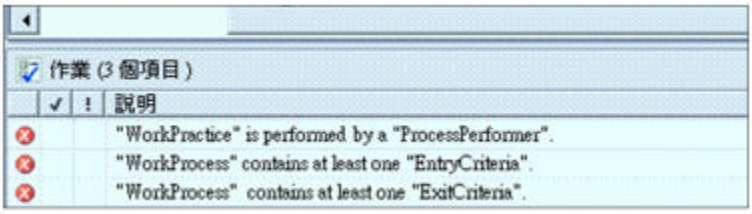

Fig. 7: An example of process verification

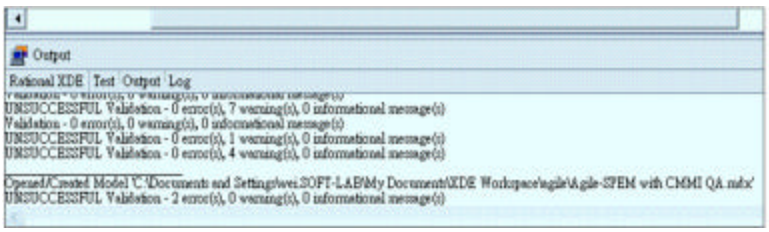

Fig. 8: Successful verification

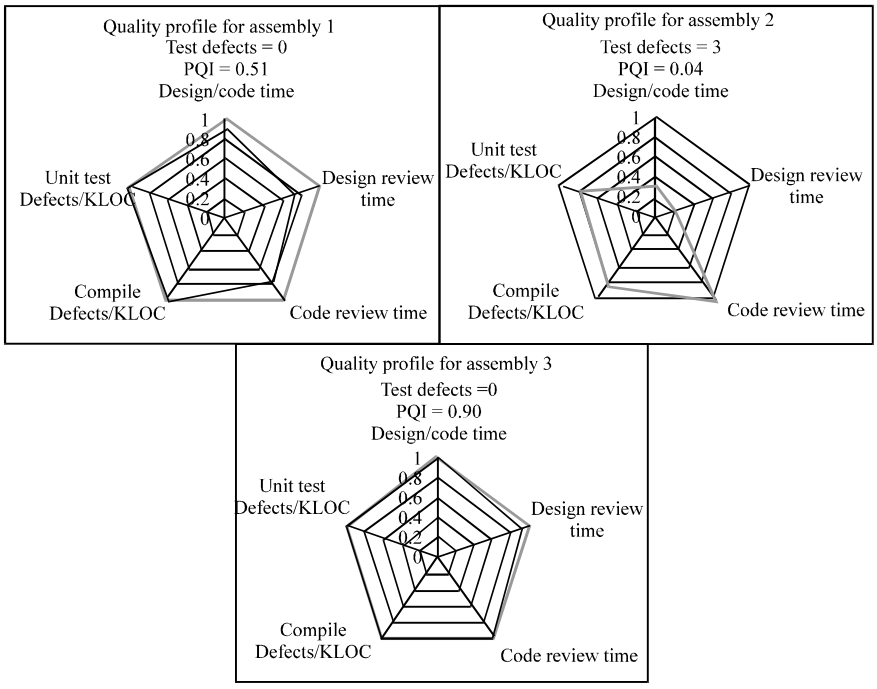

Fig. 9: Three programs for process quality index (PQI indictor)

process verification, the process model can be automatically verified; namely, the model is correct and it can thus pass the rules in the modeling environment. The automatic verification mechanism helps us verify processes and reduce much effort in manual process verification.

However, the measurement-related rules can help us check modeling condition of software processes. For example, they can check which parts do not meet the requirement of information needs. The verification of measurement-related rule in the present approach also provides an advantage to give modeler with more knowledge about the relationship between process and measure codes. The process quality index is obtained by multiplying these five values (Design/Code Time, Design Review Time, Code Review Time, Compile Defects/KLOC and Unit Test Defects/KLOC). Figure 9 shows the PQI profiles for three programs. One program with the poorest-quality profiles (numbers 2) was the only one with defects after unit testing. The rest of the programs show that PQI values above 0.5 generally had no defects after unit testing. Following these steps of process verification, it can automatically verify process model and ensure that the model is correct and can pass the rules in the modeling environment. The automatic verification mechanism helped us verify processes and reduce much effort in manual process verification.

CONCLUSION

In this study, the researcher adopted PSM framework to guide the process definition and verification. In the process definition stage, a specification to define processes and represent the processes for measurement framework was proposed. In the process verification stage, process rules were represented as OCL and embedded in the process model to represent the discipline and constraint in the process helpful for automatic process verification. This approach was used to model software process using UML based on the SPEM. Analytical models such as the PSM was widely adopted to resolve the problems of data measurement and obtain the information needed. This

approach extends the approach in previous research (Hsueh *et al*., 2008). An SPEM-based process model was developed using UML and qualitative and quantitative profile was defined in the modeling step. However, this study provided concise and essential measurement rules to verify organization processes. With the hope that this approach requires further proof though more experiments, the rules for relevant measurement rules can be enriched and further studies may be validated through the use of the proposed approach in the study.

REFERENCES

Atkinson, C. and T. Kuhne, 2002. Profiles in a strict meta-modeling framework. Science of computer programming, special issue on unified modeling language, UML 2000, pp: 5-22.

Basili, V.R. and H.D. Rombach, 1998. The TAME Project: Towards improvement-oriented software environments. IEEE Trans. Software Eng., 14: 758-773.

Bobkowska, A., 2001. Quantitative and qualitative methods in process improvement and product quality assessment. Proceedings of the 12th Conference on European Software Control and Metrics, April 2-4, UK., pp: 1-10.

Breton, E. and J. B'ezivin, 2001. Process-centered model engineering. Proceedings of 5th IEEE International Enterprise Distributed Object Computing Conference, Seattle, Washington, USA. Sep. 4-7, IEEE Computer Society, pp: 179-182.

Casati, F., S. Ceri, B. Pernici and G. Pozzi, 1995. Conceptual modeling of workflows. Proceedings of 14th Object Oriented and Entity-Relationship Approach, Gold Coast, Australia. Dec. 12-15, Lecture Notes in Computer Science, pp: 341-354.

Changchien, S.W., J.J. Shen and T.Y. Lin, 2002. A preliminary correctness evaluation model of object-oriented software based on UML. J. Applied Sci., 2: 356-365.

Chrissis, M.B., M. Konrad and S. Shrum, 2003. CMMI: Guidelines for Process Integration and Product Improvement. 2nd Edn., Addison-Wesley Professional, Reading, MA., USA., ISBN-13: 978-0-321-15496-5.

Cook, J.E. and A.L. Wolf, 1999. Software process validation: quantitatively measuring the correspondence of a process to a model. ACM Trans. Software Eng. Methodol., 8: 147-176.

Evans, A., R. France, K. Lano and B. Rumpe, 1998. The UML as a formal modeling notation. Comput. Stand. Interfaces, 19: 325-334.

Fadila, A. and G. Said, 2006. A new textual description for software process modeling. Inform. Technol. J., 5: 1146-1148.

Feiler, P. and W.S. Humphrey, 1992. Software process development and enactment: concepts and definitions. Technical Report, CMU/SEI-92-TR-004. http://www.sei .cmu. edu/ reports /92tr004.pdf.

Fuggetta, A., 2000. Software Process: A roadmap. Proceedings of the Conference on The Future of Software Engineering, June 4-11, New York, NY, USA., pp: 25-34.

Garcia, F., F. Ruiz, J.A. Cruz and M. Piattini, 2003. Integrated measurement for the evaluation and improvement of software processes. Lect. Notes Comput. Sci., 2786: 94-111.

Heaven, W. and A. Finkelstein, 2004. UML profile to support requirements engineering with KAOS. Proceedings of IEE Software, Feb. 9, IEEE, pp: 10-27.

Hsueh, N.L., W.H. Shen, Z.W. Yang and D.L. Yang, 2008. Applying UML and software simulation for process definition, verification and validation. Inform. Soft. Technol., 50: 897-911.

Humphrey, W.S., 1995. A discipline for software engineering. Addison-Wesley, Reading MA., USA., ISBN-13: 9-780201-546101.

Humphrey, W.S., 2005. A Self-Improvement Process for Software Engineers. Addison-Wesley, New York, ISBN-13: 9-780321-305497.

ISO, 2002. ISO/IEC 15939 software measurement process model. http://segoldmine.ppi-int.com/content/standard-isoiec-15939-software-measurement-process.

Ishigaki, D. and C. Jones, 2003. Practical measurement in the rational unified process, the rational edge. Proceedings of the 3rd International Conference on Quality Software. (QSIC'03), Rational, pp: 1-18.

Jager, D., A. Schleicher and B. Westfechtel, 1999. Using UML for software process modeling. Springer, Berlin, ISBN: 3-540-66538-2.

Kan, S.H., 2002. Metrics and models in software quality engineering. 2nd Edn., Addison-Wesley, New York, ISBN: 0201729156.

Kheirkhah, E., A. Deraman and Z.S. Tabatabaie, 2009. Screen-Based prototyping: A conceptual framework. Inform. Technol. J., 8: 558-564.

Koulopoulos, T.M., 1995. The workflow imperatives: Building real world business solution. Van Nostrand Reinghold, New York.

Mangan, P. and S. Sadiq, 2002. On building workflow models for flexible processes. Aust. Comput. Sci. Commun., 18: 103-109.

McFeeley, B., 1996. IDEAL: A user's guide for software process improvement. Software Engineering Institute/Carnegie Mellon University, Pittsburgh, PA., CMU/SEI-96-HB-001. http://www.sei.cmu.edu/reports/96hb001.pdf.

McGarry, J., D. Card, C. Jones, B. Layman E. Clark, J. Dean and F. Hall, 2002. Practical software measurement: Objective information for decision makers. Addison-Wesley, New York, ISBN: 0-201-71516-3.

Meng, D., Q. Zeng, F. Lu, J. Sun and J. An, 2011. Cross-organization task coordination patterns of urban emergency response systems. Inform. Technol. J., 10: 367-375.

Muketha, G.M., A.A.A. Ghani, M.H. Selamat and R. Atan, 2010. A survey of business process complexity metrics. Inform. Technol. J., 9: 1336-1344.

OMG, Inc., 2002. Software process engineering meta-model specification. Version 1.0, Object Management Group, formal/02-11-14, http://www.omg.org/technology/documents/formal/spem.htm.

OMG, Inc., 2003a. OMG unified modeling language: Superstructure (final adopted spec, version 2.0, 2003-08-02). Technical report, Object Management Group, http://www.omg.org.

OMG, Inc., 2003b. UML 2.0 OCL specification. http://www.uml.org.

Petty, D.M., 1996. Application of process modeling: An industrial view. J. Mater. Process. Technol., 60: 421-426.

Salimifard, K. and M. Wright, 2001. Petri net-based modeling of workflow systems: An overview. Eur. J. Operat. Res., 134: 664-676.

Shen W.H., N.L. Hsueh and W.M. Lee, 2011. Assessing PSP effect in training disciplined software development: A Plan-track-review model. Inform. Software Technol., 53: 137-148.

Van Loon, H., 2007. Process assessment and ISO 15504. Springer, New York, ISBN: 9-780387-300481.

Van der Aalst, W.M.P., 1998. The application of Petri nets to workflow management. J. Circuit Syst. Comput., 8: 21-66.

WMC, 1995. The workflow reference model. Document number WfMC-TC00-1003, Issue 1.1. http://www.wfmc.org/standards/docs/tc003v11.pdf.

WMC, 1999. Terminology and glossary, Document number WfMC-TC-1011, Issue 3.0. http://www.wfmc.org/standards/docs/TC-1011_term_glossary_v3.pdf.

Zisman, M.D., 1997. Representation, specification and automation of office procedures. Ph.D. Thesis, University of Pennsylvania, Wharton School of Business.