A Software Process Control Meta-Model Based on Software Metrics

Lei Liu1, Xiaojuan Hu1, Ningjia Qiu*2, Chun Shen1, Peng Zhang1
1College of Computer Science and Technology, Jilin University, China, 130022
2School of Computer Science and Technology, Changchun University of Science and Technology, 130022, China
E-mail: 269212811@qq.com

Abstract - Software process assessment plays a fundamental role in software quality assurance. In view of the great diversity of entities and difficult quantification of software process, metrics and evaluation methods must be designed in a consistent and integrated way to facilitate the management of software process control. An integrated metrics method of the software process control is presented. Experiments show that the model can effectively improve the degree of control of the software process to ensure the reliability of the infrastructure software.

Keyword: Software quality control, software process meta-model, software metrics, software process

I. INTRODUCTION

With the development of economy and society, software applications are increasingly used in all walks of life when the important of software processes was aware. The quality and the efficiency are often talked in the software development [1], both of them require software development process more efficient and controllable. At present, enterprises pay more attention to the management control of software development process [2]. Rising economic benefit prompted them to seek better ways to achieve maximum benefits, which is bound to promote the software development. Software process modeling is a primary method for software development to meet the growing quality requirements and software process standardization requirements [3].

Software process can be regarded as the set of activities, methods and improvements for developing and maintaining related products [4], and affected by user requirements, operating environment, software technology and other factors. Compared with the ordinary products in the production process, the factors make it more complex and unpredictable [5]. Software process is performed in accordance with plans, which includes design plan, development plan, execution plan, testing plan, update plan, etc. The correctness and validity are absolutely affected the normal operation and completion. Because of the artificial and technical reason, there are some exist errors and defects in software planning. In order to detect defects in earlier stage, it is proposed with modeling approach for the improved of process control and evaluation.

It should be in focus with existing software process models and meta-models on the software process assessment, metrics and analysis of influencing factors [6]. It plays a very important role for software process control in the actual project development. Therefore, this paper proposes Process Control Meta-model (SPCM), which can achieve software standardization metrics process through methods of decomposing and measuring software plans. In addition, it can recognize the differences in software plans between the expected and the setting goal. So it is effective to improve the degree of software process control and ensure software quality reliability. The paper is organized as follows. Section II summarizes related work about technical basis of SPCM and research status of software process modeling. Section III describes the modeling process of SPCM. In Section IV, SPCM is instantiated, analyzed by an example. Section V outlines conclusions and future work according to the findings obtained.

II. RELATED WORK

The basic point of the Model Driven Engineering (MDE) is software development and maintenance process with a series of activities performing around models, which indicates the model becomes the core of the software instead of coding [7]. People pay more attention to the model driven, among which the OMG company raised Model Driven Architecture (MDA) [8]. Atkinson proposed two different goals on the role of Model Driven Development (MDD). The first goal is to improve the short-term productivity by enhancing the basic value of software delivered products, and the second one is to improve the long-term productivity by prolonging software products life and reducing the outdated rate [9].
OMG proposed a series of criteria to achieve MDA. The most important one is Meta Object Facility (MOF), which is used for describing UML extensions and creating a similar UML modeling language [10]. As shown in Figure 1, MOF is a conceptual architecture with four levels, which are meta-object device layer (level M3), meta-model layer (level M2), the model layer (level M1), and data layer (level M0). The upper layer is used for defining the lower layer in order to describe the relationship of these four levels [11]. Where, Level M3 is known as meta-meta model, which can describe different modeling language for different areas as self-describing. In other words, MOF can be described by itself. Level M2 is defined by Level M3, for instance, Software Process Engineering Meta-model (SPEM) belongs to this category [12]. Level M1 is model layer, which is composed by generalization examples such as requirement analysis model defined by SPCM is this category [13].

According to the ideas of software process quantification and using SPEM as a fundamental description language, a framework has been designed to promote the software process control, which is based on Software Capability Maturity Model (SW-CMM), MDA, and software metrics techniques. Software process elements of SPEM are redesigned in SPCM, and with the expansion of SPEM, it remedies the deficiency that SPEM can’t describe metrics and control of the software process. Considering the relationship of the key factors, such as goals, tasks and issues in software process, the software process metrics algorithm was designed for solving the problem of software process quantification control, and then it had been verified for the completeness, correctness and consistency of SPCM in the instance analysis. In comparing with the existing software process meta-models, it was added in SPCM as the standardization and quantification, which could solve problems of evaluation and control for software plans in the early stage to find deficiencies in the software process immediately and improve the efficiency of development and maintenance.

At present, the subsistent software process modeling languages are divided into syntax-based process modeling language such as Petri net and Rules [14], the unified modeling language (UML) such as UML2.0 [15], and model-based process modeling language such as SPEM2.0 [16]. Different from the general software process models, SPEM describes common activities and implementation roles [17-18].

In order to solve the problem of software process quantification control, SPCM decomposes software process and adds constraint package for increasing the degree of software process control capability, which extends the ProcessStructure Package, ProcessElements Package and WorkDefinition package for SPEM. Where, the SW-CMM abstract meta-class is generalized by WorkDefinition package as the guidelines of software process [19]. The relationship is presented in Figure 2.

III. SOFTWARE PROCESS CONTROL META-MODEL

We introduce the approach of software process control, and describe the establishment of software process control meta-model in this section, which includes the design concept, definitions of the process control decomposition, algorithms, and the design of meta-classes and meta-modeling.

A. Software Process Control Design Concept

The progress speed of software process is difficult to quantify, and so does the real-time completing extent of software plans. The accurate monitoring plays an important role to ensure software quality. In order to realize the quantitative control, we propose an improved approach of software process control based on the existing assessment methods. According to the approach, we established a meta-model for process control.

The software process control is an effective approach to assure software quality. As shown in Figure 3, the software process is performed in accordance with the software plans. Initially, each plan could be divided into a series of tasks, and corresponding goalsare set for each task. Goals are converted to a set of quantifiable questions, and a goal corresponds to a set of questions. In addition, the indicators are designed as a set, which can answer the question to determine the completion degree. The needed dataset of each indicator is obtained with metric algorithm from the actual execution data of the task. Ultimately, we could gain the assessment result, and the baseline of the software process would be completed. Using this method, we can accurately grasp the implementation degree of each task, and evaluate the software plans to realize the aim of...
controlling the software process.

B. Definitions of Process Decomposition

In order to find defects in software plans earlier and achieve the exhaustive control, we adopt the quantification mind for the real-time control. With the SW-CMM standard, the software plans would correspond to a set of key practices. The software process is constituted of multiple KP, and the definitions are as follows.

Definition 1 (Software process–KP) Define software process as SP, $SP = \{KP_id\}$, parameter $id$ is number of key practices in software process. Using tuple describes the attribute of KP, and expresses as $KP = (Type, \text{MaturityLevel}, \text{Emetrics}, \text{CmmRole}, \text{WorkProduct}, \text{KPBaseline}, \text{KPGoal}, \text{PriorityLevel})$.

The relationship between software process and key practice is described in Definition 1, and the attributes of the key practice are designed. Where, $Type$ is the KP type, $\text{MaturityLevel}$ is the level of maturity, $\text{Emetrics}$ is the set of indicators collected in KP, $\text{CmmRole}$ is the set of execute role in KP, $\text{WorkProduct}$ is the set of work products, $\text{KPBaseline}$ is the baseline set of KP, $\text{KPGoal}$ is the goal set of KP, $\text{PriorityLevel}$ is the priority level of KP. The lower the maturity level, the higher its priority, and key practices in the low maturity level are prior to match and improve.

The key practice could be decomposed into a set of tasks, which is the smallest unit of software execution process, and the definitions of task are as follows.

Definition 2 (KP–Task) Each key practice can be considered as a set of tasks. The key practice is marked that, $\text{KP} = \{\text{ATask}_id\}$, KP, means one key practice, and $id$ is the number of tasks. The attribute of task can be expressed by tuple, $\text{ATask}id = (Type, \text{ParentKP}, \text{CmmRole}', \text{Schedule}, \text{Aschdule}, \text{Precondition}, \text{Postcondition}, \text{TBL})$.

The relationship of the key practice decomposed is defined in Definition 2. Where, $Type$ is the type of task and $Type = \{\text{GTask, STask}\}$, $\text{Gtask}$ and $\text{STask}$ are respectively the generic task and the specific task, $\text{ParentKP}$ records the key practice which includes the task, $\text{CmmRole}'$ is the set of existing role in the task, and contained in $\text{CmmRole}$, $\text{Schedule}$ is the set of expected task scheduler, and $\text{Schedule} = \langle \text{effort}, \text{starttime}, \text{endtime} \rangle$. Three properties separately represent the workload, the start time and the end time. $\text{Aschedule}$ is the set of actual task scheduler. $\text{TBL}$ is the performance baseline set.

When the decomposition of the software process is completed, next work is to decompose a task goals, specifically definition is as follows.

Definition 3 (Task–Goal) A goal can be expressed by $\text{TGid}$, and $\text{TGid} = (\text{GType}, \text{TQ}, \text{ATaskid})$.

A task contains one or more goals. The attributes of each goal is stated in Definition 3. Where, $\text{TGid}$ is the uniquely identifies of the goal, $\text{GType}$ is the type set of the goal, and $\text{GType} = \{\text{SG}, \text{GG}\}$. SG is special goal, and $\text{GG}$ is generic goal. $\text{TQ}$ is the question set of the goal, $\text{TQid}$ indicates the corresponding task.

Definition 4 (Goal–Question) A question can express by $\text{TQid}$ and $\text{TQid} = (\text{TQtype}, \text{Indicator}, \text{ATaskid})$, which is the unique identifier of question.

A goal can transform into a set of questions, which is express in Definition 4. Where, $\text{TQtype}$ is the type of question. It belongs to the set $\{\text{GG}, \text{SQ}\}$. $\text{GQ}$ indicated question with generic feature, and $\text{SQ}$ indicated question with special feature. $\text{Indicator}$ is the set of indicator which can abstract question for the goal. $\text{ATaskid}$ indicates the corresponding task.

Definition 5 (Question–Indicator) Metrics are recorded as Indicator, and $\text{Indicator} = \{\text{Iid}\}$. $\text{Iid}$ is designed as $\text{Iid}$, and $\text{Iid} = (\text{Data(TQid)}, \text{M(TQid)})$.

Each question has several indicators as judging criteria, which is stated in Definition 5. Where, $\text{Iid}$ means the unique identifier. $\text{Data(TQid)}$ is the dataset of indicators $\text{Iid}$ for questions $\text{TQid}$. $\text{M(TQid)}$ is the evaluation method set of indicator $\text{Iid}$ for question $\text{TQid}$.

Definition 6 (Metric algorithm) $\text{MetricA}$ is derived from Definitions 4 and 5, and $\text{MetricA} = (\text{Indicator}, f(\text{Iid}), \text{TQ}, s(TQid))$.

$\text{MetricA}$ describes the relationship between the indicator and the dataset in Definition 6. Where, $\text{Indicator}$ is a collection of algorithms used in metrics, $f(\text{Iid})$ is a method of...
metrics indicators. \( I_{id} \), \( T_Q \) is a set of evaluation questions in metric algorithms, \( s(T_Q) \) is the method of evaluation questions.

C. Algorithms of Process Control

According to the organization scale, the maturity level and the number of KPA could be determined. Software process decomposition is executed with historical models and project plans analyzed. Specific process is as shown in Algorithm 1.

**Algorithm 1** Software process decomposition algorithm:

**Input:** Algorithm:

**Output:** SP (software process metrics).

Step1 While \( \{KP_i\} \) is not null
Step2 take one key practice kp form \( \{KP_i\} \)
Step3 \( \{KP_i\} = \{KP_i\} - \{kp\} \)
Step4 Set up the attribute of kp (MaturityLevel, Emetrics, CmmRole, WorkProduct, KPBaseline, KPGoal, PriorityLevel)
Step5 kp = \( \{ATask_j\} \)
Step6 While \( \{ATask_j\} \) is not null, ,
Step7 take one task t form \( \{ATask_j\} \)
Step8 \( \{ATask_j\} = \{ATask_j\} - \{t\} \)
Step9 Set up the attribute of t (Type, ParentKP, CmmRole, Schedule, Aschdule, Precondition, Postcondition, TBL)
Step9 Loop Step6
Step10 Loop Step1
Step11 Return SP = \( \{ATask_j\} \)

The goal is mostly non-quantitative text descriptions and attributes. It is a worthy problem for how to transform non-quantitative goals into quantitative indicators. With the results of Algorithm 1 are regarded as conditions, the target can be transformed into quantified problem set as follows.

**Algorithm 2** Conversion algorithm:

**Input:** TG = \( \{TG_i\} \)

**Output:** TG = \( \{TQ_j\} \)

Step1 While \( \{TG_i\} \) is not null
Step2 take one goal tg form \( \{TG_i\} \)
Step3 \( \{TG_i\} = \{TG_i\} - \{tg\} \)
Step4 Set up the attribute of tg (TargetType, TQ, ATask)
Step5 \( T_Q = \{TQ_j\} \)
Step6 While \( \{TQ_j\} \) is not null,
Step7 take one question tq form \( \{TQ_j\} \)
Step8 \( \{TQ_j\} = \{TQ_j\} - \{tq\} \)
Step9 Set up the attribute of tq (TQType, Indicator)
Step10 Loop Step6
Step10 Loop Step1
Step11 Return TG = \( \{TQ_j\} \)

**Algorithm 3** Software process control algorithm:

**Input:** \( SP = \{ATask_i\} \) and \( TGoal = \{TG_i\} \)

**Output:** the results of software process control metrics

Step1 While \( \{ATask_i\} \) is not null
Step2 take one task form \( \{ATask_i\} \)
Step3 Extract a TGid of the task t from \( \{TG_i\} \), and mark it as \( \{tg_m\} \),
Step4 While \( \{tg_m\} \) is not null
Step5 take one goal tg form \( \{tg_m\} \), and gain the corresponding question \( \{tq\} \),
Step6 While \( \{tq\} \) is not null
Step7 take one question \( \{tq\} \) form \( \{tq\} \), and gain the corresponding indicator \( \{I_i\} \), and collecting data of ind
Step10 Execute task t, gain the practical implementation data of task and the comparative value
Step13 Loop Step4
Step14 Loop Step2
Step15 Return the results of software process control metrics

The relationship between tasks and goals can be expressed as MP based on Algorithm 3. According to Definitions 2 and 3, the measure relationship can be marked as MP = \( \{\} = \{\} \)(Type, ParentKP, CmmRole, Schedule, Aschdule, Precondition, Postcondition, TBL). It is obtained from this relationship whether the implementation of each task meets task goal or not.

The output of algorithm 3 is the result of the quality evaluation complying with the software plan, which is a comparison between the actual execution result and the expected goal. If the evaluation result is not satisfied with the result, it is necessary to adjust the task decomposition process and the expected goal setting process. Then, re-execute the task and assess the result, until it reaches the desired objectives.

D. Software Process Control Meta-modeling

According to the above definitions and algorithms, we use the process description language SPEM to describe the software process control meta-modeling for the particularity and universality of the software process control. Measurement control method based on real-time transformation software is proposed to make it more in line with demand for the product software control process. Redesign Process Element meta-class and Work Definition meta-class of SPEM. The relationships between two meta-classes and the other meta-classes are more fitting software process control and measurement through the decomposition of two meta-classes. The specific refinement process of two meta-classes is shown in Figure 4, and all elements involved in the Figure are given below in detail.

MaturityLevel describes the maturity capability of software process, and the attributes have name, islevel and isachieved. Where, name is the name of maturity level,
islevel shows which level it can get, and isachieved means whether the maturity level is reached. Each maturity level has a number of Key Process Areas (KPA), which is a set of activities. The purpose of implementing the activities is to reach a certain maturity level. Name and issatisfied are the attributes of KPA. Where, name is described the name of KPA, which has different range and corresponds to different MaturityLevel. Issatisfied indicates whether the key process area is satisfied. The maturity level is determined by the scale of software and organization. In other words, the scale of software described by SoftwareScale meta-class, the scale of organization is described by OrganizationScale meta-class.

Each KPA is composed of a series of Key Practices (KP), which describes the main activities in KPA. The type of main activities is divided into GeneralActivity meta-class and SpecialActivity meta-class. Where, GeneralActivity meta-class describes the software process activities with common features, and SpecialActivity meta-class describes the software process activities with special features.

![Figure 4. Main Meta-Classes of SPCM](image)

A KP has one or more Key Practice Goal (KPGoal) which describes the prospective of KP. The attributes are name, content and isachieved. Where, name is the identifier of the goal. Content expressed details of the goal, and isachieved indicates whether the goal is achieved. The type of KP Goal can be divided into general goals (expressed by GeneralGoal meta-class) and specific goals (expressed by SpecialGoal meta-class).

KP is performed by multiple executive roles (expressed by ProcessRole meta-class). Process Role can be divided into multiple working groups (named CmmGroup). A CmmGroup contains several CmmRole, which performs one or more activities. The attributes are workID, name and CmmGroupId. Where, workID is a unique identifier for each CmmRole. Name describes the role type, and CmmGroupId expresses the belonged working group.

MeasureApproach describes software process metrics methods, and the attributes are name, function and usablerange. Where, name is the name of metrics method. Function describes the function of the metrics method, and usablerange expresses the application range of the method. There are common activity decomposition strategies and
general metrics methods for general goals and activities. Similarly, there are specific activity decomposition strategies and general metrics methods for specific goals and activities.

Each maturity level has a baseline set, and Baseline meta-class describes the implementation baseline of KP. When all the set of KP baseline in entire maturity level are satisfied, it is considered the baseline of maturity level to be done.

Precondition meta-class describes the pre-conditions of the activity. When it is satisfied, the activity can start. The attributes are name and isSatisfied. Name is the name of the precondition, and isSatisfied indicates whether the precondition is achieved.

Postcondition meta-class describes the post-condition of the activity. When it is conformed after the activity, the next activity would continue to proceed. Ditto to that the attributes are name and isSatisfied. Name describes the post-conditions name, and isSatisfied indicates whether the post-condition is attained. If an activity is completed with the pre-conditions fit, but not satisfied post-conditions, it is called an exception to explain this phenomenon.

Software process is decomposed into a set of tasks. Each task is corresponding to one or more task goals. Then the goal transformed into a set of quantifiable questions, and each question corresponds to one or several indicators. We detect the completed degree of metrics from the implementation of practical task. The completed degree of the metrics is the answer for each question. So we measure the actual finished extent of each task to answer the question, and to achieve the quantification of the software process metrics.

IV. EXPERIMENTAL VERIFICATION

The complexity of meta-model mainly depends on the space size of the software system. In order to verify the availability of the approach, we illustrate the approach with establishing the model of the software requirement analysis process (RAP). We transformed the goal into questions, and extracted dataset for the indicator to complete the decomposition, metrics and assessment of the RAP.

The software maturity level was set as level 3 according to the scale of organization and software. The key process areas include Requirement management (ReqM), Software Quality Assurance (SQA), Software Project Planning (SPP), Software Configuration Management (SCM), and Quantitative Process Management (QPM) to meet software requirement analysis process.

Design all the key practices of RAP, and key practice set was marked as KP = {requirements management, configuration management, optimizing management, quality management, requirement feature management}. The goal set was marked as KPGoal = {management institutionalization, quality management institutionalization, optimization process institutionalization, software configuration management institutionalization, effective requirement management}. After all, key practices were decomposed into tasks, the specific situation of content is shown in TABLE 1.

<table>
<thead>
<tr>
<th>TGoal</th>
<th>TG Type</th>
<th>TG Type</th>
<th>TQ Type</th>
<th>Indicator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Institutionalize managed process/TG1</td>
<td>GG</td>
<td>GG</td>
<td>With conformability of the plan/TQ1</td>
<td>Conformability of the plan/I1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>With conformability of products/TQ2</td>
<td>Org schedule plan/I1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>With traceability of products/TQ3</td>
<td>Identifiability of record identification/I1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>With identifiability of products/TQ4</td>
<td>Validity of record identification/I1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>With Controllability of process/TQ5</td>
<td>Controllable cost/I1, plan/I1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>With repeatability of process/TQ6</td>
<td>Reusability of process fragment/I1</td>
</tr>
<tr>
<td>Institutionalize quality management process/TG2</td>
<td>GG</td>
<td>GG</td>
<td>With stability of process/TQ7</td>
<td>Schedule conform the plan/I1, process changing/I1</td>
</tr>
<tr>
<td>Institutionalize optimizing process/TG3</td>
<td>GG</td>
<td>GG</td>
<td>With validity of SPI/TQ8</td>
<td>Cost conform the plan/I10, process duration time decrease/I14</td>
</tr>
<tr>
<td>Institutionalize configuration management process/TG4</td>
<td>GG</td>
<td>GG</td>
<td>With validity of SCM/TQ9</td>
<td>Configuration cost decrease/I15</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>With stability of SCM/TQ10</td>
<td>Configuration changing/I16, cost conform plan/I17</td>
</tr>
<tr>
<td>Effective requirement management/TG5</td>
<td>SG</td>
<td>SG</td>
<td>With stability of requirement/TQ11</td>
<td>Requirement changing/I16, conformability of requirement/I10</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>With conformability of requirement management/TQ12</td>
<td>Requirement conform Org/I30</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>With validity of requirement management/TQ13</td>
<td>Cost recovery/I21</td>
</tr>
</tbody>
</table>
According to the TABLE 1 and Algorithm 1, the results of RAP decomposition were marked as RAP = \{ATaskij\} = \{Function Management, Quality Management, Optimizing Management, Configuration Management, Validity Management\} = \{FunM, QuaM, OptM, ConM, ValM\}.

According to the TABLE 1 and Algorithm 2, the decomposition results of the task goals were marked as TGoal = \{TGid\} = \{Institutionalize managed process, Institutionalize quality management process, Institutionalize optimizing process, Institutionalize configuration management process, Effective requirement management\} = \{TG_1, TG_2, TG_3, TG_4, TG_5\}.

The first task was selected as FunM, and the corresponding goal was marked as Institutionalize management process (TG_1). Metrics results were expressed as MP_1 = <(GTask, ReqM, {CmmRolei}, (effot, starttime, endtime), (Aeffot, Asstarmtime, Aendtime)), Precondition, Postcondition, TBL), (GG, \{(TQ_i, GQ, \{I_1\}), (TQ_j, GQ, \{I_1, I_2\}), (TQ_k, GQ, \{I_1, I_3\}), (TQ_l, SQ, \{I_1, I_3, I_4\}), (TQ_m, SQ, \{I_4\}), FunM\}>

The second task was selected as QuaM, and the corresponding goal was marked as Institutionalize quality management process (TG_2). Metrics results were expressed as MP_2 = <(GTask, SQA, {CmmRolei}, (effot, starttime, endtime), (Aeffot, Asstarmtime, Aendtime)), Precondition, Postcondition, TBL), (GG, \{(TQ_i, GQ, \{I_1\}), (TQ_j, GQ, \{I_2\}), (TQ_k, SQ, \{I_1\}), (TQ_l, SQ, \{I_2\}), (TQ_m, SQ, \{I_3\})\}, FunM)>

The third task was selected as OptM, and the corresponding goal was marked as Institutionalize optimizing process (TG_3). Metrics results were expressed as MP_3 = <(GTask, SPP, {CmmRolei}, (effot, starttime, endtime), (Aeffot, Asstarmtime, Aendtime)), Precondition, Postcondition, TBL), (GG, \{(TQ_i, GQ, \{I_1\}), (TQ_j, GQ, \{I_2\}), (TQ_k, SQ, \{I_3\})\}, OptM)>

The forth task was selected as ConM, and the corresponding goal was marked as Institutionalize configuration management process (TG_4). Metrics results were expressed as MP_4 = <(GTask, SCM, {CmmRolei}, (effot, starttime, endtime)), Precondition, Postcondition, TBL), (GG, \{(TQ_i, GQ, \{I_1\})\}, (TQ_m, GQ, \{I_2\}), (TQ_n, SQ, \{I_3\}), ConM)>

The fifth task was selected as ValM, and the corresponding goal was marked as Effective requirement management (TG_5). Metrics results were expressed as MP_5 = <(GTask, QPM, {CmmRolei}, (effot, starttime, endtime)), Precondition, Postcondition, TBL), (SG, \{(TQ_1, SQ, \{I_2\}), (TQ_2, SQ, \{I_3\}), (TQ_3, SQ, \{I_4\})\}, ValM)>

No matter which type (common or special type) is selected for the task, the completion degree and the progress speed could be obtained by comparing the actual implementation with the corresponding metric indicators.

The metrics process is stated by the MP_i, and the details of the execution are demonstrated by the TQ_i and I_i. It is helpful for the meta-model to control the software process accurately, and to improve the quality reliability.

UML is symbol base of SPEM, and SPCM is an extension to SPEM, so UML is the symbol base of SPCM. Furthermore, UML, SPEM and SPCM are published model description language, which have the attribute of consistency and correctness. The actual task execution degree is obtained from the analysis of actual execution workload and actual execution time. With the execution degree zero does not exist, the statistical completeness could be guaranteed in metrics process.

V. CONCLUSION

In this paper, we present an approach for software process management and control, and establish an improved meta-model to process metrics and evaluation. A great diversity of entities metrics is existed in the evaluation of software process from the process meta-model to project planning, resources allocation, and the delivered projects. Definition algorithms transform software process goals into task goals. The meta-model is SW-CMM compliant, and it is necessary to manage the metrics and control process for software organizations with the initial support. It extends the Software Process Engineering Meta-model, and uses diversiform semantic process elements effectively. The meta-model applies to the actual control of any software entity in plans during the whole software lifecycle, especially when it is in early stages of software design development process. The approach assists companies to lower the cost of the late correction, which improves the control power of the software quality reliability.

Our future research is to extend the SPCM-based approach for obtaining disassembly strategy of control and metrics with software process, to develop and perform experiments with software process modeling tool, and to solve the model transformation problem with the existing studies.

ACKNOWLEDGMENT

The authors would like to thank the editor and all the referees for their useful comments and contributions for the improvement of this paper.

This research work was supported by the National Natural Science Foundation of China under Grant Nos. 61300049, the Specialized Research Fund for the Doctoral Program of Higher Education of China under Grant No. 201206112005, the China Postdoctoral Science Foundation under Grant No. 2013M500612, the Key Program for Science and Technology Development of Jilin Province of China under Grant Nos. 20130206052GX, 20140520069JH and the Fundamental Research Funds for the Central Universities under Grant No. 93K172014K12.
REFERENCES


