Tailoring and Verifying Software Process

Il-Chul Yoon
Software Engineering Department, ETRI
161 Gajeong-Dong, Yusong-gu, Taejon,
305-350, Republic of Korea
icyoon@etri.re.kr

Sang-Yoon Min, Doo-Hwan Bae
Division of CS, Department of EECS, KAIST
373-1 Kusong-dong, Yusong-gu, Taejon,
305-701, Republic of Korea
{sang, bae}@salmosa.kaist.ac.kr

Abstract

Process tailoring and verification are very important since project-specific processes are tailored from an organizational process standard and many quality assurance activities are based on the process standard. However, existing researches do not provide systematic method for the process tailoring and verification.

In this paper, we propose a systematic method for formalizing a process standard clearly with the encapsulated reusable process modules, for tailoring process modules, and for verifying tailored process. AAG (Activity Artifact Graph) is used to represent and tailor each process module. Additionally, a prototype tool is implemented to support proposed method. We believe that proposed method is helpful in tailoring a process standard and verifying a tailored process.

1 Introduction

It is well known that good software process is a basis for high quality software. However, it may be very inefficient if we redesign software process from scratch every time. Thus, an organizational software process standard is very important for software companies. Among the many concerns in employing the organizational process standard, tailoring for a project-specific process is one of the very practical concerns. In tailoring activity, there are very important issues described in the following paragraphs.

Clear definition of process: The ambiguity of process standards hinders standard-preserving process tailoring. Most standards are described in a natural language, and they can be perceived differently. Thus, to reduce discrepancy between process standard and the perceived process, process standard formalization is necessary [3, 12]. Without a formal process standard, a company may not determine how much a tailored process preserves the organizational process standard.

Reusability of the process standard: We can regard tailoring as a reuse activity. To reuse organizational process standard for tailoring, it is required to modularize a process standard as highly-cohesive and lowly-coupled units [9], which encapsulate process fragment in the form of internal structure with explicit interfaces. However, most process standards do not provide modularity explicitly.

Preserving properties in the standard during tailoring: Many process entities and dependencies among the entities exist scattered thorough the process standard. Although those relations have to be preserved after tailoring as much as possible, few researches provide a systematic method for the standard-preserving process standard tailoring and verification on the tailored process. Instead, it is left to the process designers to tailor process standard without violating the dependency relations among entities in the process standard.

In this paper, we formalize a process model using encapsulated process modules, and propose a systematic tailoring method for each process module. Process designer can reuse predefined process modules and tailor them using proposed tailoring method. We also propose several conditions to ensure process correctness, and a metric to evaluate standard compliance. Using the metric and conditions, tailored process can be verified whether tailored process is tailored as intended.

This paper is organized as follows. In the next section, we introduce related researches. In section 3, we define process model using process module and propose tailoring operations. After introducing correctness conditions and a metric for process verification in section 4, we show a case study in section 5. Finally, we conclude in section 6 with a brief summary and discussion of the future work.

2 Related Work

In this section, we briefly introduce two representative existing researches on the process standard tailoring and standard compliance evaluation.

2.1 Process Programming and Testing

ProcePT [13] by Welzel et al. proposed a method for describing and tailoring process standard. Requirements in GV-Model are described using PROLOG, and the states of each artifact are modeled as a finite state machine. For each artifact, attributes such as *from activity*, *to activity*, and *the state of the artifact* are described in the *product flow scheme* table. For GV-Model tailoring, about 90 deletion conditions are used. Each deletion condition is derived from GV-Model considering project characteristics. For example, an activity PM 1.1 of GV-Model can be deleted when '*Software Maintenance Project and Project Manual available*'. For the tailored process, the correctness and standard compliance are examined by checking *whether all dependency relations among remaining process entities are satisfied* and *how much the tailored process satisfies the requirements of GV-Model*, respectively.

Although ProcePT suggests a method for process tailoring and compliance evaluation, it is difficult to reuse part of GV-Model since GV-Model is not built into the encapsulated units for corresponding process phase. Additionally, *deletion* is the only tailoring operation even if other tailoring operations are also prevalent [8].

2.2 Emmerich's Approach

Emmerich [4][5] focused on the standard compliance between a process standard and an actual process under enactment. A process standard is modeled as a set of practices, and each practice is derived from a process standard by inspecting the process standard carefully. For each practice, properties related to documents are described in the first-order logic. When there is an attempt to read or write a document, registered events on the document occur. Since standard compliance is defined as *the consistency between the actual process and the normative model*, it can be managed by evaluating the properties assigned on related documents in run-time whenever an event occurs.

Although this research proposes standard compliance management approach by run-time process monitoring, process tailoring is not mentioned. Additionally, process standard is not built from encapsulated process units. Thus, it is difficult to reuse parts of the process standard, either.

3 Process Tailoring using Process Module

In this section, we propose a formal definition of process model and four tailoring operations: *Addition, Deletion, Splitting, and Merging*.

3.1 Formal Definition of Process Model

We assume that a process model consists of interrelated process modules defined as *an encapsulated reusable unit of process fragment*. Although the similar concepts with the process module are also proposed by other process modeling languages such as SLANG [1, 2], they are developed focused on the process execution, and does not support rich properties assigned on process entities in this paper. Additionally, many process standards do not include the behavioral process information. Thus, we need a simple process modeling notation to model standard process and tailored process using process module, and it is described by AAG that represents the relationship between activities and artifacts of the corresponding process module. Before define process module, we briefly define activity and artifact.

An activity is a process entity which consists of the following properties:

- *N*: Name of the activity
- *T*: Type of the activity
- *IR*: Input artifact set of the activity
- *OR*: Output artifact set of the activity

Similarly, an artifact is a process entity which consists of following properties:

- *N*: Name of the artifact
- *T*: Type of the artifact
- *CA*: Consuming activity set of the artifact
- *PA*: Producing activity of the artifact
- *SR*: Artifact set in the strong relation with the artifact
- *II*: Value representing if the artifact is input interface
- *OI*: Value representing if the artifact is output interface
- *P*: Value representing if the artifact is preexistent
- *D*: Value representing if the artifact is deliverable

Each activity or artifact has type as illustrated in Figure 1. An AAG has input interface set *I* and output interface set *O*. They are special artifact set of AAG, which can be connected with other process modules. Each input interface in *I* is an artifact required by a process module, and each output interface in *O* is produced by process module. Using defined activity and artifact, an AAG is defined as a bipartite graph that represents the structure of a process module. Activities, artifacts, and the relations between them are represented in the graph.

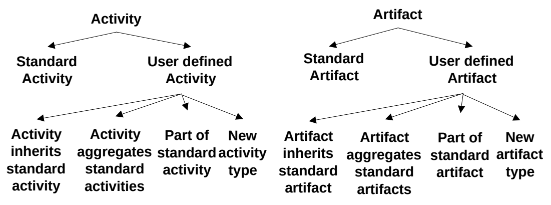

Activity
Standard Activity
User defined Activity
Activity inherits standard activity
Activity aggregates standard activities
Part of standard activity
New activity type
Artifact
Standard Artifact
User defined Artifact
Artifact inherits standard artifact
Artifact aggregates standard artifacts
Part of standard artifact
New artifact type

Figure 1. Type Hierarchy of Process Entity

Definition 3.1 *(AAG) An activity-artifact graph of a process module pm is a bipartite graph with vertex sets Act_{pm}, Art_{pm} and edge set Rel_{pm}, where*

- Act_{pm} *: a finite set of activities in the process module*
- Art_{pm} *: a finite set of artifacts in the process module*
- Rel_{pm} *: a finite set of dependency relations that is a subset of ($Act_{pm} \times Art_{pm}$) $\cup$ ($Art_{pm} \times Act_{pm}$), where*
  - *a dependency relation (a,b) $\in Rel_{pm}$ if activity a$\in Act_{pm}$ generates artifact b$\in Art_{pm}$*
  - *a dependency relation (b,a) $\in Rel_{pm}$ if activity a$\in Act_{pm}$ uses artifact b$\in Art_{pm}$.*
  - *there is no distinct activities a,b$\in Act_{pm}$ such that (a,c) $\in Rel_{pm} \wedge$ (b,c) $\in Rel_{pm}$ for an artifact c $\in Art_{pm}$*

A process module is defined using the Definition 3.1.

Definition 3.2 *(Process Module) A process module consists of $N_{pm}, I_{pm}, O_{pm}, AAG_{pm}$, where*

- N_{pm} *: a unique name of process module pm in the process,*
- $I_{pm} = \{i_k | \; i_k \in Art_{pm} \wedge i_k.II = true\}$
- $O_{pm} = \{o_k | \; o_k \in Art_{pm} \wedge o_k.II = true\}$

An entire process model is defined by process modules and the configuration among them. The following is a formal definition of entire process.

Definition 3.3 *(Process Model) A process model is defined as a tuple, Process = $\langle PM, Configuration\rangle$, where*

- *PM = $\{pm_1, pm_2, ..., pm_n\}$ is a finite set of process modules for n>0,*
- *Configuration : set of pairs (o_k, i_m) between $o_k \in O_{pm_a}$ and $i_m \in I_{pm_b}$ for some a, b, k, and m, such that satisfy the following conditions:*
  - *$\forall \; o_k \in O_{pm_a}$, if the value 'D' of o_k is not set as 'deliverable', there exists some $i_m \in I_{pm_b}$ for some a,b,m,k*
  - *$\forall \; i_m \in I_{pm_b}$, if the value 'P' of i_m is not set as 'pre-existent', there exists some o_k of pm_a for some a,b,m,k*

There can be a special relationship among artifacts. When artifact A is meaningful only when artifact B proceeds A, we say that artifact A is in *strong relation* with artifact B. In other words, artifact B can not be deleted before artifact A from the model since artifact A is meaningful when artifact B proceeds. This relation is specified by the attribute SR of artifact properties, and it can be defined in the process standard in advance. However, a process designer can add strong relations during process module tailoring. Strong relation is defined as follows.

Definition 3.4 *(Strong Relationship)*
For two artifacts $A, B \in Art_{pm_1} \cup Art_{pm_2} \cup ... \cup Art_{pm_n}$, if $A \in B.SR$, then B is in strong relationship with A.

3.2 Tailoring the Process Module

Since there is no identical project, there is no identical process in the world, either [6]. Many process modules need to be tailored during process design. In this paper, tailoring is defined as "*all activities to delete or modify a part of the standard process, or to add new entities to the standard process without violating the dependencies among entities considering related environment*."

Even if a process designer endeavors to tailor a standard process based on our definition, incompliances can occur without systematically guided and automated process tailoring support. Thus, we explain systematic tailoring operations preserving dependency relations in the standard process, and illustrate them with intuitive figures. When process designers tailor process modules or configure the interface mappings among modules, they must be attentive not to make circles composed of process entities. If a circle is essential in the process, it must be designed as two or more similar process modules [9]. In this paper, we assume that no circularity exists in the process.

3.2.1 Tailoring Operation: Process Entity Addition

Addition operation is classified into the standard entity addition and user-defined entity addition. Standard entity means that the entity type is already defined in the standard process, and user-defined entity type means that the entity type is newly defined by a process designer. Activity addition, artifact addition, and relation addition are possible.

Since we believe that all process entity addition reflect the rationale of a process designer, we regard that process entity addition does not violate the standard compliance. The following Algorithm 1 is used for artifact addition, and Figure 2 illustrates *detailed design* process module of ISO/IEC 12207 after applying several addition operations. In the figure, a black rectangle means an artifact, a white rectangle means an activity, and arrows mean the dependency relations among entities.

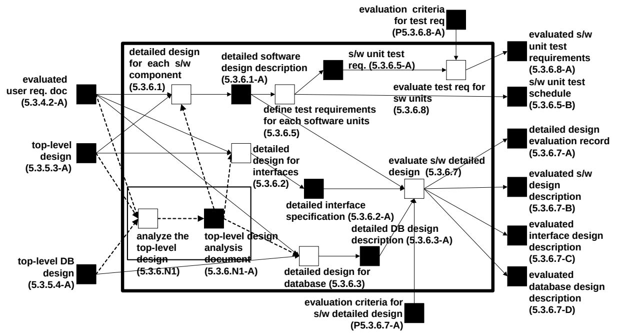

evaluation criteria for test req (P5.3.6.8-A)
evaluated user req. doc (5.3.4.2-A)
top-level design (5.3.5.3-A)
top-level DB design (5.3.5.4-A)
detailed design for each s/w component (5.3.6.1)
detailed software design description (5.3.6.1-A)
s/w unit test req. (5.3.6.5-A)
evaluate test req for sw units (5.3.6.8)
define test requirements for each software units (5.3.6.5)
detailed design for interfaces (5.3.6.2)
evaluate s/w detailed design (5.3.6.7)
detailed interface specification (5.3.6.2-A)
detailed DB design description (5.3.6.3-A)
analyze the top-level design (5.3.6.N1)
top-level design analysis document (5.3.6.N1-A)
detailed design for database (5.3.6.3)
evaluation criteria for s/w detailed design (P5.3.6.7-A)
evaluated s/w unit test requirements (5.3.6.8-A)
s/w unit test schedule (5.3.6.5-B)
detailed design evaluation record (5.3.6.7-A)
evaluated s/w design description (5.3.6.7-B)
evaluated interface design description (5.3.6.7-C)
evaluated database design description (5.3.6.7-D)

Figure 2. Process Entity Addition

Algorithm 1. Artifact Addition

```
For an artifact $a \notin AAG_{pm}.Art_{pm}$,
  fun Add_Artifact($a$, $AAG_{pm}$)
    let $\langle Act, Art, Rel \rangle = AAG_{pm}$;
    return $\langle Act, Art \cup \{a\}, Rel \rangle$;
  end;
```

3.2.2 Tailoring Operation: Process Entity Deletion

Since standard processes are written for general software project, deletion may be the most common process tailoring operation, and other process entities are affected by entity deletion. For example, when an activity is deleted, the artifacts generated from the activity also have to be deleted.

In our approach, when a process designer deletes an artifact in a process module, he has to consider whether the artifact is interface artifact or not by checking the attribute *II* and *OI*. Similarly, in the activity deletion, we have to check whether output interface or input interface is produced or used by the activity. If deleted artifact is an interface artifact, we have to remove it from the interface set of the process module, and if it is in the strong relationship with other artifacts, deletion has to be aborted. The following Algorithm 2 is used for activity deletion and the Figure 3 is the process after deleting the activity *evaluation test requirements for software units* and input interface *evaluation criteria for test requirements* from Figure 2. The attribute value *OI* of the artifact *s/w unit requirements* is still *false* after deletion, and this is obviously a syntactic error. However, proposed algorithm does not correct it automatically. Instead, process designer can find out the error by syntactic correctness checking proposed in section 4.1.

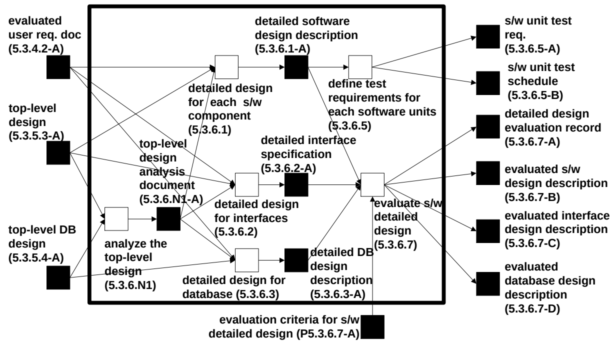

evaluated user req. doc (5.3.4.2-A)
top-level design (5.3.5.3-A)
top-level DB design (5.3.5.4-A)
detailed software design description (5.3.6.1-A)
detailed design for each s/w component (5.3.6.1)
define test requirements for each software units (5.3.6.5)
top-level design analysis document (5.3.6.N1-A)
detailed interface specification (5.3.6.2-A)
detailed design for interfaces (5.3.6.2)
analyze the top-level design (5.3.6.N1)
detailed design for database (5.3.6.3)
detailed DB design description (5.3.6.3-A)
evaluate s/w detailed design (5.3.6.7)
evaluation criteria for s/w detailed design (P5.3.6.7-A)
s/w unit test req. (5.3.6.5-A)
s/w unit test schedule (5.3.6.5-B)
detailed design evaluation record (5.3.6.7-A)
evaluated s/w design description (5.3.6.7-B)
evaluated interface design description (5.3.6.7-C)
evaluated database design description (5.3.6.7-D)

Figure 3. Process Entity Deletion

Algorithm 2. Activity Deletion

```
For an activity $a \in Act_{pm}$
  fun Del_Act($a$, $pm$)
    let $\langle N, I, O, \langle Act, Art, Rel \rangle\rangle = pm$;
      for each $b \in a.OR$ do
        if $b.SR \neq \emptyset$ then
          return $\langle N, I, O, \langle Act, Art, Rel \rangle\rangle$;
        endif
      end do
      for each $c \in a.IR$ do
        $c.CA := c.CA - \{a\}$; $Rel := Rel - \{(c, a)\}$;
      end do
      for each $d \in a.OR$ do
        $Rel := Rel - \{(a, d)\}$;
        if $d.OI = true$ then $O := O - \{d\}$; endif
        for each $e \in d.CA$ do
          $e.IR := e.IR - \{d\}$; $Rel := Rel - \{(d, e)\}$;
        end do
        $Art := Art - \{d\}$;
      end do
      $Act := Act - \{a\}$;
    return $\langle N, I, O, \langle Act, Art, Rel \rangle\rangle$;
  end
```

3.2.3 Tailoring Operation: Process Entity Splitting

A process designer can use splitting for managing process in detail. If a project is large-scale, or an activity in the standard process needs to be managed in finer activities, a process designer may split an activity into several sub-activities. In this case, the original activity will be deleted, and new activities will be added to the process module. The types of newly added activities should be decided by process designer, and the information must be maintained that the original activity is split to what activities. When an artifact is split, each split artifact has to be added as output artifact of the PA of the original artifact, and as input artifact of the CAs of the original artifact. The information that newly added artifacts are split from also has to be maintained for compliance evaluation.

In activity splitting, split activities can use the input artifacts of the original activity, but we do not know in advance which output artifact is produced from which split activity. Thus, creating the relations between split activities and output artifacts are left to the process designer. The following Algorithm 3 are used for activity splitting. In Figure

4, the activity *detailed design for each software component* is split to activities *behavioral design of software unit using MSC, make software functionality design document* and *design blocks in the system*.

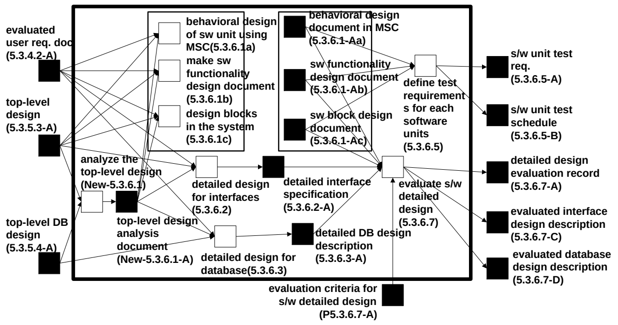

evaluated user req. doc (5.3.4.2-A)
top-level design (5.3.5.3-A)
top-level DB design (5.3.5.4-A)
behavioral design of sw unit using MSC(5.3.6.1a)
make sw functionality design document (5.3.6.1b)
design blocks in the system (5.3.6.1c)
behavioral design document in MSC (5.3.6.1-Aa)
sw functionality design document (5.3.6.1-Ab)
sw block design document (5.3.6.1-Ac)
define test requirements for each software units (5.3.6.5)
analyze the top-level design (New-5.3.6.1)
top-level design analysis document (New-5.3.6.1-A)
detailed design for interfaces (5.3.6.2)
detailed interface specification (5.3.6.2-A)
detailed design for database(5.3.6.3)
detailed DB design description (5.3.6.3-A)
evaluate s/w detailed design (5.3.6.7)
s/w unit test req. (5.3.6.5-A)
s/w unit test schedule (5.3.6.5-B)
detailed design evaluation record (5.3.7-A)
evaluated interface design description (5.3.6.7-C)
evaluated database design description (5.3.6.7-D)
evaluation criteria for s/w detailed design (P5.3.6.7-A)

Figure 4. Process Entity Splitting.

Algorithm 3. Activity Splitting

```
For an activity $a \in Act_{pm}$, and a set of activities
$R = \{r_1, r_2, ..., r_n\}$, such that $r_i \notin Act_{pm}, 1 \leq i \leq n$.
  fun Act_Split(a, R, pm)
    let $\langle N, I, O, \langle Act, Art, Rel \rangle\rangle = pm$;
        for each $r_i \in R$ do
          $\langle Act, Art, Rel\rangle$:=$Add\_Activity(r_i, \langle Act, Art, Rel\rangle)$;
          for each $b \in a.IR$ do
              $b.CA := b.CA \cup r_i$;  $r_i.IR := r_i.IR \cup b$;
              $Rel := Rel \cup \{(b, r_i)\}$;
          end do
        end do
        for each $b \in a.IR$ do
          $b.CA := b.CA - \{a\}$; $Rel := Rel - \{(b, a)\}$;
        end do
        for each $b \in a.OR$ do
          $b.PA := null$; $Rel := Rel - \{(a, b)\}$;
        end do
        $Act := Act - \{a\}$;
    return $\langle N, I, O, \langle Act, Art, Rel\rangle\rangle$;
  end
```

3.2.4 Tailoring Operation: Process Entity Merging

When a process designer apply the standard process to a small-size project or rapid application development, he may hope to manage process in higher-level to reduce workload. Merging is to transform two selected activities in the standard process into one merged activity. Thus, two original activities will be removed, and a new activity will be added. Two types of merging is possible.

The first type is when dependency relation exists between two activities *a*, *b*. In this case, we have to merge all intermediate process entities, either. To know the existence of dependency, we define dependency set in a process module as follows.

Definition 3.5 *(Total Dependency Set(TDS) of a Process Module)*

TDS_{pm} is a transitive closure of Rel_{pm}, denoted by Rel^{+}_{pm}, where

- $\forall x,y \in Art_{pm} \cup Act_{pm}, \{(x,y) | \exists i, (i \geq 1 \wedge xRel^{i}y)\}$

Let *a*, *b* the merged activities, and assume that *a* proceeds *b*. If an entity *c* satisfies the condition $(a, c) \in TDS_{pm} \wedge (c, b) \in TDS_{pm}$, then the entity is an intermediate one. In this way, we simply obtain the set of entities that satisfy this condition, and this set has to be merged with the activities *a*, *b*. For each input artifact of *a* or *b* that is not intermediate artifact of *a*, *b*, it must be an input artifact of the new activity. And, if an intermediate artifact is used by other activities that are not intermediate activities, it must be an output artifact of the new activity. The second type of merging is when no dependency relation exists between two activities. If there is no dependency relation *(a,b)* in TDS_{pm}, we only have to merge the two activities as a new activity, and connect all input and output artifacts of each activity to the new activity.

Artifact merging is not considered since an artifact can not be generated from several activities. If artifacts from two distinct activities are merged, as a result, the merged artifact has two *PA*s, and it violates our assumption. Thus, if a process designer wants to merge artifacts, he has to make a new activity that integrates the artifacts, and make new integrated artifact. In this case, the information has to be managed that the new artifact consists of the merged artifacts for module connection.

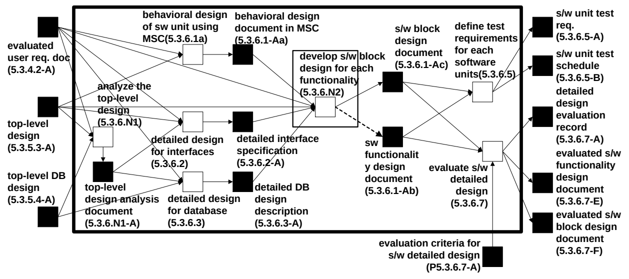

evaluated user req. doc (5.3.4.2-A)
top-level design (5.3.5.3-A)
top-level DB design (5.3.5.4-A)
behavioral design of sw unit using MSC(5.3.6.1a)
behavioral design document in MSC (5.3.6.1-Aa)
analyze the top-level design (5.3.6.N1)
top-level design analysis document (5.3.6.N1-A)
detailed design for interfaces (5.3.6.2)
detailed interface specification (5.3.6.2-A)
detailed design for database (5.3.6.3)
detailed DB design description (5.3.6.3-A)
develop s/w block design for each functionality (5.3.6.N2)
s/w block design document (5.3.6.1-Ac)
sw functionality design document (5.3.6.1-Ab)
define test requirements for each software units(5.3.6.5)
evaluate s/w detailed design (5.3.6.7)
s/w unit test req. (5.3.6.5-A)
s/w unit test schedule (5.3.6.5-B)
detailed design evaluation record (5.3.6.7-A)
evaluated s/w functionality design document (5.3.6.7-E)
evaluated s/w block design document (5.3.6.7-F)
evaluation criteria for s/w detailed design (P5.3.6.7-A)

Figure 5. Process Entity Merging

We described in the previous section that the information that original activity is split to which activities has to be preserved for evaluating standard compliance. However, contrary to the splitting, such information is not preserved in merging. User-defined type may be assigned to the merged activity by process designer, and the many dependency relations that exist before merging may be removed since he already decide not to manage the original entities in detail.

Merging is also different to applying deletion operation several times. In activity deletion, the generated artifacts have to be removed from the process. However, in merging, only intermediate artifacts are deleted. Since the merging algorithm is a bit complex, abstraction algorithm is omitted in this paper. Figure 5 illustrates that two activity *make software functionality design document* and *design blocks in the system* is merged to one activity.

4 Static Process Verification

In this section, we will describe a static verification method for the tailored process. Since there is no super process designer [6], there can be incompliances or defects in the tailored process, and they can cause project delay or replanning in extreme case. Thus, we believe that static process verification before enactment is very important. It consists of syntactic correctness checking, type conformance checking and standard compliance evaluation.

4.1 Syntactic Correctness Checking

To check the syntactic correctness, we have to check syntactic correctness for each process module and entire process, respectively. For each process module, we check whether syntactic errors exist in its AAG. In syntactic correctness checking, we do not enforce rigorous conditions on the relations between entities since we believe that rigorous constraints are not helpful in practice. Instead, a process module is said to be syntactically correct if it satisfies the following conditions.

- $\forall\ a \in Rel_{pm},\ a \in (Act_{pm}, Art_{pm}) \vee a \in (Art_{pm}, Act_{pm})$
- $\forall\ a \in Art_{pm},$
  - $a.CA = \emptyset \Rightarrow a.OI = true$
  - $a.PA = null \Rightarrow a.II = true$
  - $(a.II = true \Rightarrow a.OI = false) \vee (a.OI = true \Rightarrow a.II = false)$
  - $a.P = true \Rightarrow (a.PA = null \wedge a.II = true \wedge a.OI = false)$
- $\forall a \in Act_{pm}, a.OR \neq \emptyset \wedge a.IR \neq \emptyset$

To check syntactic correctness of the entire process, we have to check whether the attribute *D* of each unmapped *OI* (output interface) of process modules is set *true*, and whether the attribute *P* of each unmapped *II* (input interface) of process modules is set *true*. If a tailored process satisfies these two conditions, we regard the tailored process as syntactically correct process.

4.2 Type Conformance Checking

During process tailoring, a process designer may change the attributes of artifacts or activities, and he also maps *OI*s of a process module to *II*s of other modules. By type conformance checking, process designer can find out whether all interface mappings among process modules are type conformant. We assume that the artifact type hierarchy is already constructed, and the type of artifacts is one of the types in the Figure 1. If the following conditions are satisfied by all interface mappings, we regard the process as type-conformant.

- For each mapping (o_k, i_k), for $i_k \in I_{pm_a}$, and $o_k \in O_{pm_b}$,
  - $i_k.T = o_k.T$, or
  - $o_k.T$ include $i_k.T$ as a part of $o_k.T$, or
  - $o_k.T$ is a specialized type of $i_k.T$

4.3 Standard Compliance Evaluation

In ISO/IEC 12207, standard compliance is defined as *the performance of all the processes, activities, and tasks selected from ISO/IEC 12207 in the Tailoring Process for the software project* [7]. It only mentions how much the tailored process is observed in the process execution. Another definition of standard compliance is *consistency between the actual process and the normative model embedded in the standard* [4, 5]. This definition says how much the standard process is observed in the process execution.

These definitions do not mention the compliance between standard process and tailored process. When we consider the growing importance of QA and SPI activities, evaluating compliance between standard process and tailored process is also meaningful. In this paper, we define the standard compliance as *how much the tailored process preserves the dependency relations in the standard process*. If a process designer tailors standard process using proposed tailoring operations, the dependency relations in the standard may be preserved as much as possible. However, it is necessary to know how much dependency relations are preserved in the tailored process. For that, we extend the Definition 3.5 to the entire process.

Definition 4.1 *(Total Dependency Set (TDS) of Process) $TDS_{process}$ is a transitive closure of $Rel_{process}$, denoted by $Rel^{+}_{process}$, where*

- $\exists x,y \in Art_{process} \cup Act_{process}$, $\{(x,y) \mid \exists i, (i \geq 1 \wedge x Rel^{i} y)\}$
  - $Art_{process} = Art_{pm_1} \cup Art_{pm_2} \cup \ldots \cup Art_{pm_n}$.
  - $Act_{process} = Act_{pm_1} \cup Act_{pm_2} \cup \ldots \cup Act_{pm_n}$.
- $Rel_{process} = Rel_{pm_1} \cup Rel_{pm_2} \cup \ldots \cup Rel_{pm_n} \cup ICD_{processs}$.
  - *Inter-Component dependency, $ICD_{processs}$, is a set of dependency relations (o_k, i_m), for all pm_a, pm_b, $o_k \in O_{pm_a}$, $i_m \in I_{pm_b}$, where $pm_a \neq pm_b$*

In Definition 4.1, inter-component dependency means the dependency relations between process modules. The information maintained in the splitting have to be considered in building $Rel^{+}_{tailored}$, either. By regarding the type of each split activity as that of original activity, we can obtain how many dependency relations are preserved. For example, if an activity *a* is split to *b, c*, we can regard the dependency relations related to *b, c* as the dependency relations related to the original activity *a*. In this way, we can obtain the dependency relations related to *a* by searching those related to *b, c*. Since $TDS_{process}$ includes all dependency relations in a process, we can represent how much dependency relations are preserved in the tailored process by the following simple formula.

$$\frac{|Rel^{+}_{standard} \cap Rel^{+}_{tailored}|}{|Rel^{+}_{standard}|}$$

This formula only expresses how much standard dependency relations are preserved in the tailored process. In other words, it does not mean the quality of the tailored process. However, by providing the missed standard dependency relations for process designers, they can have an opportunity to review the missed relations to confirm whether the missed relations are intended or not. We believe that this information is meaningful.

5 Case Study

We applied our approach to the real-world process to demonstrate that our approach is helpful to tailor a standard process and verify the the tailored process. For this case study, we formalized ISO/IEC 12207 international software life-cycle process standard by careful inspection, and tailor the process into the organization standard process. The tailored process is currently used for a large software development company in Korea. After tailoring, we can verify the tailored process.

All tailoring operations and the static process verification techniques are implemented by enhancing SoftPM [11], a software project management tool. In SoftPM, a process is modeled as CogNet, an activity-oriented net, and it is transformed to MAM-Net based on Pr/T net [10]. Since it is activity-oriented, we should add more functionalities to reflect the characteristics of each artifact in the AAG.

5.1 Tailoring Standard Process Modules

Proposed tailoring operations was enough to tailor standard process modules for the company. In this paper, we introduce only one process module tailoring with brief explanation of the core tailoring operations in the following paragraphs.

Original *software detailed design* standard process module is very similar with the Figure 2. Under the enactment of this module, identified top-level software components are designed in detail, and the test requirements for components have to be developed. We reconstructed the target process module by applying the following tailoring operations on the standard process module, and the Figure 6 illustrates the tailored *software detailed design* process module after applying 48 tailoring operations.

- Addition: *analyze the top-level design* is added to the process
- Deletion: *definition of the test requirements for each software unit* and *update of user documentation*
- Merging: *behavioral design document in MSC*, *detailed database design description,* and *detailed interface specification* are consolidated into *final functionality design document*.

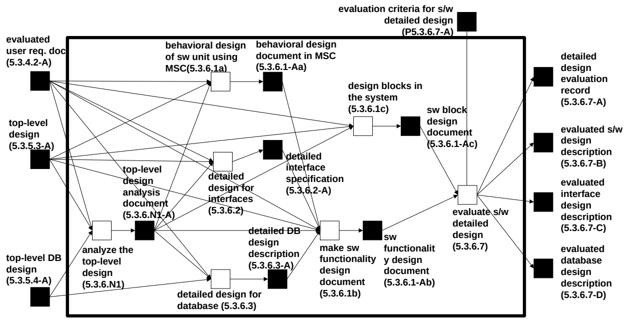
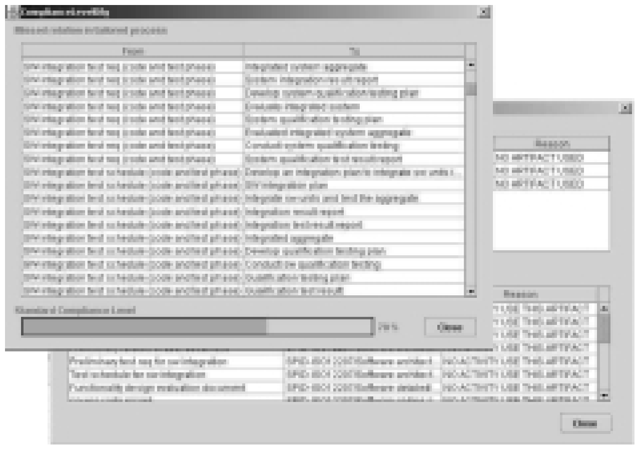

evaluation criteria for s/w detailed design (P5.3.6.7-A)
evaluated user req. doc (5.3.4.2-A)
behavioral design of sw unit using MSC(5.3.6.1a)
behavioral design document in MSC (5.3.6.1-Aa)
design blocks in the system (5.3.6.1c)
sw block design document (5.3.6.1-Ac)
detailed design evaluation record (5.3.6.7-A)
top-level design (5.3.5.3-A)
evaluated s/w design description (5.3.6.7-B)
detailed interface specification (5.3.6.2-A)
top-level design analysis document (5.3.6.N1-A)
detailed design for interfaces (5.3.6.2)
evaluated interface design description (5.3.6.7-C)
detailed DB design description (5.3.6.3-A)
evaluate s/w detailed design (5.3.6.7)
top-level DB design (5.3.5.4-A)
analyze the top-level design (5.3.6.N1)
make sw functionality design document (5.3.6.1b)
sw functionality design document (5.3.6.1-Ab)
evaluated database design description (5.3.6.7-D)
detailed design for database (5.3.6.3)

Figure 6. Tailored Detailed Design Process.

After tailoring, tailored process is verified using the developed tool. If we assume that the tailored process is properly reconstructed and there is no syntactic error, and it is 70%-compliant with the ISO/IEC 12207 standard process it is based upon. Since compliance is measured by counting the number of preserved dependency relations of the standard process, 70%-compliance means that 30% of the direct or indirect dependency relations among standard process entities are lost in the course of tailoring. We can find out that the dependency relations related to the *unit testing and software integration and integration testing* are considerably ignored in the tailored process. The result is illustrated in Figure 7.

6 Conclusion and Future Works

An authoritative standard process has been strongly demanded by industry, and several process standards are suggested by international organizations and companies. However, many process standards are described informally in


Compliance results
Missed relation in tailored process
Standard Compliance Level
70 %
Close
Close

From	To
SW integration test req (code and test phase)	Integrated system aggregate
SW integration test req (code and test phase)	System integration result report
SW integration test req (code and test phase)	Develop system qualification testing plan
SW integration test req (code and test phase)	Evaluate integrated system
SW integration test req (code and test phase)	System qualification testing plan
SW integration test req (code and test phase)	Evaluated integrated system aggregate
SW integration test req (code and test phase)	Conduct system qualification testing
SW integration test req (code and test phase)	System qualification test result report
SW integration test schedule (code and test phase)	Develop an integration plan to integrate sw units ...
SW integration test schedule (code and test phase)	SW integration plan
SW integration test schedule (code and test phase)	Integrate sw units and test the aggregate
SW integration test schedule (code and test phase)	Integration result report
SW integration test schedule (code and test phase)	Integration test result report
SW integration test schedule (code and test phase)	Integrated aggregate
SW integration test schedule (code and test phase)	Develop qualification testing plan
SW integration test schedule (code and test phase)	Conduct sw qualification testing
SW integration test schedule (code and test phase)	Qualification testing plan
SW integration test schedule (code and test phase)	Qualification test result

Reason
NO ARTIFACT USED
NO ARTIFACT USED
NO ARTIFACT USED

		Reason
		[illegible] USE THIS ARTIFACT
		[illegible] USE THIS ARTIFACT
		[illegible] USE THIS ARTIFACT
		[illegible] USE THIS ARTIFACT
Preliminary test req for sw integration	SPID-ISO4 12207 Software architect	NO ACTIVITY USE THIS ARTIFACT
Test schedule for sw integration	SPID-ISO4 12207 Software architect	NO ACTIVITY USE THIS ARTIFACT
Functionality design evaluation document	SPID-ISO4 12207 Software detailed	NO ACTIVITY USE THIS ARTIFACT

Figure 7. Verification Result Screen.

high-level, and companies have difficulty in tailoring the standard processes for their projects. Besides, incompliances can occur in the course of tailoring. Since such incompliances can delay process execution and make it difficult to apply other organizational activities, a systematic approach for tailoring standard process and verifying the tailored process is required.

In this paper, we formalized standard process by defining the notion of process module as a template process for tailoring, and by configuring the interface mappings between process modules. We also introduced standard-preserving tailoring operations classified into *addition*, *deletion*, *splitting*, and *merging*, which can be applied on process modules. Lastly, static process verification techniques are suggested. Through the proposed approach, a process designer can obtain the following benefits.

- He can reuse predefined process modules in a process design.
- He can use proposed tailoring operations for achieving tailored process minimizing the incompliances.
- He can verify the tailored process before enactment.

However, there still remains further work. Firstly, behavioral information and process entity *agent* can be modeled in our process definition. By including those information, process designer may extend the tailored process for process enactment. Secondly, we can evaluate the standard compliance between tailored process and actual process with the extended information. For this work, a run-time process monitoring technique has to be developed.

References

[1] S. Bandinelli, A. Fuggetta, and S. Grigolli. Process modeling in-the-large with slang. In *Proc. International Conference on Software Process*, 1993.

[2] Sergio Bandinelli and Alfonso Fuggetta. Software Process Model Evolution in the SPADE Environment. *Transaction on Software Engineering*, 19(12):1128–1144, Dec 1993.

[3] Sergio Bandinelli and Alfonso Fuggetta. Modeling and improving and industrial software process. *Transaction on Software Engineering*, 21(5):440–453, May 1995.

[4] W. Emmerich, A. Finkelstein, C. Montangero, S. Antonelli, S. Armitage, and R. Stevens. Managing standards compliance. *Transaction on Software Engineering*, 25(6):836–851, Nov/Dec 1999.

[5] Wolfgang Emmerich, Anthony Finkelstein, Carlo Montangero, and Richard Stevens. Standards compliant software development. In *Proc. International Conference on Software Engineering*, 1997.

[6] Watts S. Humphrey. *Managing the Software Process*. Addison-Wesley, 1989.

[7] IEEE/EIA. *Industry Implementation of International Standard ISO/IEC 12207 : 1995, Software Life Cycle Processes - Life cycle data*, 1998.

[8] SC7 ISO/IEC JTC1. Information technology - Guide for ISO/IEC 12207. Technical report, ISO/IEC, December 1998.

[9] M.I. Kellner. Connecting reusable software process elements and components. In *Proc. International Software Process Workshop*, 1996.

[10] Sang-Yoon Min, Ik-Joo Han, Wei-Jin Park, and Doo-Hwan Bae. An approach to software process management based on formal process modeling and analysis. In *Proc. APSEC*, 1997.

[11] S.Y. Min, H.D. Lee, , and D.H. Bae. SoftPM: A Software Project Management System Reconciling Formalism with Easiness. *Information and Software Technology*, 42(1):1–16, 2000.

[12] M. Verlage and J. Münch. Formalizing Software Engineering Standards. In *Proc. International Software Engineering Standards Symposium*, 1997.

[13] D. Welzel, H.-L. Hausen, and W. Schmidt. Tailoring and conformance testing of software processes: the ProcePT approach. In *Proc. International Software Engineering Standards Symposium*, 1995.