This article is downloaded from

http://researchoutput.csu.edu.au

It is the paper published as

Author: L. Inchaiwong, A. S. M. Sajeev, X. Huang and S. Ramingwong
Title: A Comparison of Software Process Models
Editor: H. R. Hamid R. Arabnia
Conference Name: 2008 International Conference on Software Engineering Research & Practice, SERP 2008
Conference Location: USA
Publisher: CSREA Press 2008
Year: 2008
Pages: 503-509
Date: 14-17 July, 2008

Abstract: Software processes make a critical contribution to successful software projects. A great challenge is to choose an appropriate process for a project. However, since process specifications vary widely in their quality and level of detail, selecting the most appropriate process could be very confusing and misleading. Therefore, a systematic study of process models is needed. Software Process Analysis Method (SAM) is created to assist developers in comparing and analyzing software process models.

Author Address: xhuang@csu.edu.au

URL: http://www.informatik.uni-trier.de/~ley/db/conf/serp/serp2008.html#InchaiwongSHR08
http://www.world-academy-of-science.org/worldcomp08/ws/conferences/serp08

CRO identification number: 8775
A Comparison of Software Process Models

Lachana Inchaiwong*
Chiang Mai University
Chiang Mai
Thailand
Tel. +66 5394 2024
Fax. +66 5394 2072
lachana@eng.cmu.ac.th

A.S.M. Sajeev
University of New England,
Armidale, NSW 2351
Australia
Tel. +61 2 6773 2285
Fax. +61 2 6773 3312
sajeev@turing.une.edu.au

Xiaodi Huang
Charles Sturt University,
Albury, NSW 2640
Australia
Tel. +61 2 60519652
Fax. +61 2 6051 9878
xhuang@csu.edu.au

Sakgasit Ramingwong
University of New England
Armidale 2351
NSW, Australia
Tel. +61 2 6773 3488
Fax. +61 2 6773 3312
sramingw@turing.une.edu.au

Abstract—Software processes make a critical contribution to successful software projects. A great challenge is to choose an appropriate process for a project. However, since process specifications vary widely in their quality and level of detail, selecting the most appropriate process could be very confusing and misleading. Therefore, a systematic study of process models is needed. Software Process Analysis Method (SAM) is created to assist developers in comparing and analyzing software process models.

Keywords—software process model, process specification, process analysis method, process comparison, process selection.

I. INTRODUCTION

The significance and the diversity of software development lifecycles and methodologies have created a situation where a thorough study and analysis of various processes is needed to provide researchers and developers a better understanding and realization of process characteristics and their differences. Software process selection can ultimately lead the project to either success or failure.

Over the years, a large variety of software processes has been developed and applied by software engineers. Waterfall model, arguably the best-known software process, is now a forty-year-old concept. Due to uncountable scholarly feedback and criticism, this classic software process has resulted in a number of variations. In the same way, many other better-known processes such as Spiral model have more than one variations and applicability [1] [2].

Generally, a software process contains information of activities, objectives, conditions, time, and other related information. This information is usually demonstrated by using a model and associated descriptive text. Several software process modeling approaches have been proposed by researchers. Many of them are variations of existing software processes designed to enhance their abilities to cope with various derivative aspects [3] [4] [5] [6] [7] [8] whereas there are others which are entirely new concepts [9] [10]. Although the approaches are different, all attempts share the same purpose – to provide structure and discipline to software development activities.

Choosing a software process is not difficult. However, choosing the most appropriate process can be extremely challenging. The Waterfall model, even though has been subjected to a large amount of criticisms, is still in favor in several software organizations. Although the required software process characteristics is known, due to their great variety and variation, selecting the most appropriate software process is often difficult. An systematic analysis tool will greatly assist software engineers to choose an appropriate software process.

II. RESEARCH PROBLEMS

There are several problems in software process analysis. Firstly, the terminology used by different process developers can be confusing. It is not very surprising to see a term in one process being used for a totally different concept in another process. In the same way, it is also common to see a same concept being described using different terms by two different modelers. For example, user requirements are called “Feature” in Feature Driven Development while it is called “User Stories” in Extreme Programming [11] [12].

Another serious problem in software process analysis is that processes are defined at different levels of detail. A software process with five steps could be even more complex than a ten-step process. This is because each step in the first software process may consist of several sub-processes of which some may be very detailed.

Descriptions of software processes are often expressed in textual English and/or using diagrams. Often, there is
insufficient explanation of the notations used in the diagrams. These make comparison of processes difficult. In order to properly analyze software processes, standardization is required. Once the notation is standardized, an efficient software process analysis method will be able provide insights into software processes.

III. PROCESS DICTIONARY

As stated earlier, it is not uncommon for different software processes to use different terms to mean the same thing, or the same term to mean different concepts. Such terminological ambiguity can include vital key components in the software process such as activities, objects and other artifacts. This could generate confusion. For instance, for a software engineering, the term “Spike” may not come across as something to do with software processes unless they are familiar with Extreme Programming where it stands for a simple program developed to explore potential solution to tough design questions. A new process specification in future may use the term for something completely different such as a software bug or an unexpected obstruction.

A process dictionary is a key component in software process standardization. It contains collections of terms used in software process models and their connections to other related terms. The purpose of the process dictionary is not only to introduce a set of terms that can be universally used across all software processes but also to minimize the inessential repetition of terms. The process dictionary assists software engineers to precisely define process terms and map related terms defined in various software processes into common ones. Therefore, a systematic and balanced comparative analysis of software processes can be made. Table 1 is an example of a process dictionary.

### Table 1: A process dictionary

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Granularity</th>
<th>Description</th>
<th>Synonym</th>
<th>Related</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acceptance test</td>
<td>Artifact</td>
<td>Atomic</td>
<td>A test written by the customer, (or QC on the customer's behalf) that tests the entire system to ensure that a specific piece of functionality is present and functions correctly.</td>
<td>Customer test,</td>
<td>Test</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Functional test</td>
<td></td>
</tr>
<tr>
<td>Acceptance testing</td>
<td>Activity</td>
<td>Non-Atomic</td>
<td>Formal testing conducted to determine whether or not a system satisfies its acceptance criteria and to enable the customer to determine whether or not to accept the system.</td>
<td>Customer testing,</td>
<td>Formal testing</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Functional testing</td>
<td></td>
</tr>
<tr>
<td>Coding</td>
<td>Activity</td>
<td>Non-Atomic</td>
<td>(1) In software engineering, the process of expressing a computer program in a programming language. (2) (IEEE Std 1002-1987 [91]) The transforming of logic and data from design specifications (design descriptions) into a programming language.</td>
<td>Code, Programming</td>
<td>Implementation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Document</td>
<td>Activity</td>
<td>Atomic</td>
<td>(1) A medium, and the information recorded on it that generally has permanence and can be read by a person or a machine. Examples in software engineering include project plans, specifications, test plans, user manuals. (2) To create a document as in (1).</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The process dictionary, $D$, includes the definition of process terms and their relation to other terms. It initially contains terms taken from IEEE Glossary of Software Engineering Terminology [13] and IEEE/EIA 12207.0-1996 (Standard for Information Technology-Software life cycle processes) [14]. New terms can be added by the process analyst as appropriate.

The process dictionary is a collection of entries, $e$. Each entry involves a unique term referred as common term $t_c$. Each term is related to other terms with different degrees of similarity. In other words, a term is associated with a set of synonyms, $S_e$, and a set of related terms, $L_e$. In most cases, a term has only one description. However, a term can have more than one description if its meaning is different in different process models. A set of meanings of the term is referred by $\text{Meaning}(t_e)$. $T$ is a set of all terms.

Determining similarity and related terms is complicated and can be confusing, especially with a large collection of terms. Automatic determination can be performed by computers. However, in order to maximize the accuracy, human justification is still needed at the final stage. The following semantic is used for determining relationship and measurement of the term.

$$ e = (t_c, \text{Meaning}(t_e), S_e, L_e), \text{ where }$$

$$ \forall e_1, e_2 \in D : t_{e1} \neq t_{e2}$$

$$ t_c \subseteq T$$

$$ S_e \subseteq T$$

$$ L_e \subseteq T$$

Synonyms and related terms to term $t_e$ are determined by the degree of similarity between $\text{Meaning}(t_e)$, $\text{Meaning}(S_e)$ or $\text{Meaning}(L_e)$. Since $\text{Meaning}$ is a set, at a time, only one meaning from each set is considered. The higher the degree of similarity is, the higher the probability that two terms are related.
The term “Closeness” is used for referring to quantification of term-similarity. Theoretically, for two software process terms, \(t_1, t_2 \in T\), closeness \(C\), or \(C(\text{Meaning}(t_1), \text{Meaning}(t_2))\), indicates the closeness in meanings between two terms. The value of \(C(\text{Meaning}(t_1), \text{Meaning}(t_2))\) is between 0 and 1. Table 2 describes the meanings of closeness.

### Table 2

<table>
<thead>
<tr>
<th>(C(\text{Closeness}))</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(t_1) and (t_2) has no relationship</td>
</tr>
<tr>
<td>1</td>
<td>(t_1) and (t_2) are identical</td>
</tr>
<tr>
<td>0 &lt; (C &lt; 1)</td>
<td>(t_1) and (t_2) are related but not identical</td>
</tr>
</tbody>
</table>

In order to automatically quantify the closeness of software process terms, an open source software called ReqSimile is adapted. ReqSimile is a Java application based on research led by Johan Natt och Dag [15]. The similarity measurement is based on Cosine similarity. Although ReqSimile is originally designed for tackling problems on requirement sets, it can be efficiently adapted to perform the process terminology comparison. In this research ReqSimile’s database is altered in order to suit the content of the previously described process dictionary. Upon selection of a term in a process model, ReqSimile can efficiently determine the closeness to all terms in the process dictionary.

The preparation of the analysis is based on the following rules:

- Each process term is verified for its existence in the process dictionary. If it exists and its meaning is identical to the meaning defined in the dictionary, no update is required.
- If the term exists in the dictionary but there are no suitable meanings, a new meaning is added to an existing term.
- If the term is not in the dictionary and cannot be substituted by any terms in the dictionary, it will be added to the dictionary as a new term.
- Then, the term’s connection to other terms in the dictionary is determined.

The algorithm for developing the process dictionary is illustrated in Figure 1.

```
for each process model, p
  for each term, \(t_0\) in p
    let the meaning of \(t_0\) in p be \(m\)
    if there exists an entry \(e\) in \(D\) such that
      \(t_0\) equals \(e\)
      if there is an element \(me\) in \(\text{Meaning}(e)\) such that \(C(m, me)\) equals 1
        \(D = D \cup \{t_0\}\) to add \(t_0\) as a synonym for entry \(e\)
        break; // to explore relations with other terms
    else
      \(D = D \cup \{t_0\}\) to add \(t_0\) as a new term to the dictionary
      continue;
```

Fig. 1 The algorithm for developing the process dictionary.

### A. Root terms

A concept call root terms is introduced to establish the baseline for comparison of process terminology. Root terms set the foundation for process terms in the process dictionary. All terms in the process dictionary are either directly or indirectly related to at least one of the root terms, either by having a close meaning or by being a part of the root terms. Thirteen root terms are derived from the development process defined in IEEE/IEA12207 international standard [17]. An additional root term, supporting processes, is added to represent supporting activities which are present in many software processes. For analysis purposes, these fourteen root terms are categorized into six groups, i.e., Planning, Specifying, Designing, Coding and Testing, Delivering, and Supporting. Table 3 displays all root terms used in this research.

### Table 3

<table>
<thead>
<tr>
<th>Group</th>
<th>Root terms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Planning</td>
<td>1. Process implementation</td>
</tr>
<tr>
<td>Specifying</td>
<td>2. System requirements analysis</td>
</tr>
<tr>
<td>Designing</td>
<td>3. Software requirements analysis</td>
</tr>
<tr>
<td>Coding &amp; Testing</td>
<td>4. System architectural design</td>
</tr>
<tr>
<td></td>
<td>5. Software architectural design</td>
</tr>
<tr>
<td></td>
<td>6. Software detailed design</td>
</tr>
<tr>
<td>Delivering</td>
<td>7. Software coding and testing</td>
</tr>
<tr>
<td></td>
<td>8. Software integration</td>
</tr>
<tr>
<td></td>
<td>9. Software qualification testing</td>
</tr>
<tr>
<td></td>
<td>10. System integration</td>
</tr>
<tr>
<td></td>
<td>11. System qualification testing</td>
</tr>
<tr>
<td>Supporting</td>
<td>12. Software installation</td>
</tr>
<tr>
<td></td>
<td>13. Software acceptance report</td>
</tr>
<tr>
<td></td>
<td>14. Supporting processes</td>
</tr>
</tbody>
</table>

The definition of root terms is as follows:

Let \(r\) be a root term and \(R\) be the set of all root terms

\[
R = \{\text{Process implementation, System requirements analysis, System architectural design, Software requirements analysis, Software architectural design, Software detailed design, Software coding, Software testing, Software integration, Software qualification testing, System integration, System qualification testing, Software installation, Software acceptance support, Supporting Processes}\}
\]

A root term is defined based on the following condition:
1. It is not close to any root terms, that is:
   \[ \forall r_1, r_2 \in R \mid C(r_1, r_2) = 0 \]
2. Every term is directly or indirectly connected to at least one root term
   \[ \forall t \in T \mid \exists r \in R \mid 0 < C(t, r) \leq 1 \]

IV. SOFTWARE PROCESS ANALYSIS METHOD (SAM)

Software process analysis method (SAM) is introduced to promote better understanding of software processes. This includes their similarities, differences and relations. The result gained from SAM can be efficiently used for assisting in process selection, process tailoring, process adoption, process re-design or other related areas.

SAM involves three main steps: (1) Elaboration, (2) Normalization, and (3) Abstraction. These essential steps are explained in following subsections.

A. Elaboration

Two approaches are commonly used for defining software processes, i.e., top-down and bottom up. In a top-down approach, the highest level is firstly described and then refined into smaller details. On the other hand, in a bottom-up approach, the individual detailed elements are defined first and then joined to form a higher level. In many cases, a combination of both approaches is used.

In SAM’s elaboration stage, the process definition is mapped to cover all details specified in the base references. A base reference is ideally the standard reference for the process definition. However, if a standard reference is not available, which is normally the case, the most available authoritative reference(s) is chosen. In this stage, Order, a step-wise software process decomposition notation, is used for elaborating the process specification [18]. The elaboration usually follows the top-down approach. The software process is elaborated until no further elaboration is possible. The final elaboration is determined either when all details in the base references are captured and/or when each element in the elaborated model reach atomic state. An atomic element is the smallest element which cannot be further divided.

B. Normalization

In general, a process term describes a process activity in a natural language phase. Each term has at least one meaning. In addition, the process term may have relationships to others terms in the process dictionary. For example, the terms “software requirement specification” and “write user stories” are very closely related. Obviously, using only a common logic might not be able to specify the relationship of these two terms. Fortunately, SAM can efficiently judge the relationship of both terms by systematically comparing their descriptions.

As stated previously, in order to be able to compare processes on equal footing, each process element needs to be defined using the same standard terminology. This can be done by using the process dictionary.

After the elaboration is finished, the normalization stage begins. In this stage, SAM, assisted by Reqsimile, automatically determines and replaces non-standard terms by their corresponding standard terms from the previously defined process dictionary. For example the term “user stories” defined in XP is replaced by “user requirements” and the term “sprint” used in Scrum is replaced by “iteration”.

The normalization stage ends when all elaborated specific process terms are replaced with dictionary terms. However, in some cases, the descriptions of the process terms can be extremely brief and vague. In such cases, human intervention is needed to improve the accuracy of the replacement.

C. Abstraction

Once every term is normalized in the normalization stage, all similar activities are combined to provide an abstract view of each process model. Through the abstraction step, SAM establishes the baseline for process comparison by relating normalized items to their root elements.

The normalized terms are further abstracted into root terms. This allows the process engineer to recognize the root of the activities. Thus, various process models can be compared.

In this stage, detailed abstract models of each software process are created. These abstract models are flow-chart like diagrams consisting of six nodes, i.e., Planning,
Specifying, Designing, Coding & Testing, Delivering and Supporting, as defined in the previous section. Multiple occurrences of the same root term are compressed to one node. The frequency of occurrence is indicated by the thickness of the node. Similarly, the connections between activities are compressed and their frequencies are also indicated by the thickness of the arrow connecting each node. The filled circles indicate the starting and ending points of the process. Figure 2 is an example of the abstract model created by SAM’s abstraction stage.

The abstract model provides some valuable insight of the process nature. From the abstract model, it can be easily seen that some processes focus heavily on specifying while some other processes do not concentrate on this node at all. The case study in the next section provides a more concrete example of this stage.

Fig. 3. A top-level view of the Waterfall model in Order notation

Fig. 4. Waterfall’s Elaboration: Preliminary Program Design in Order notation

V. CASE STUDY

In this case study, SAM is used for analyzing the classic Waterfall model. The base reference of the Waterfall model is [19]. This base reference describes the process in top-down perspective. Figure 3 illustrates the top level view of the Waterfall process in Order notation, as derived from the base reference.

A. Elaboration

This top level of the Waterfall model displays eight modules. Four of them can be further elaborated. Figure 4 shows an example of the second level elaboration from the preliminary program design module. In this case, the elaboration ends after the second turn since all activities reach atomic level.

B. Normalization

After the elaboration is completed, all final terms are normalized based on the process dictionary. Table 4 displays the process terms and normalized terms.

C. Abstraction

The normalized terms are further abstracted in this stage. Table 5 exhibits the abstracted terms of the Waterfall model.

With these abstracted terms, an abstract model of the Waterfall model is created as shown in Figure 5.
According to Figure 5, the main emphasis of Waterfall process is designing. Specifying and Coding & Testing are the second most important stages of this process. Interestingly, planning does not receive much attention as other development phases. Furthermore, the dotted supporting node indicates that the Waterfall model does not define any supporting processes at all.

**TABLE 4**

<table>
<thead>
<tr>
<th>Waterfall process terms</th>
<th>Normalized terms</th>
</tr>
</thead>
<tbody>
<tr>
<td>System requirements specification</td>
<td>Requirements specification</td>
</tr>
<tr>
<td>Software requirements specification</td>
<td>Software requirements specification</td>
</tr>
<tr>
<td>Document system overview</td>
<td>Document</td>
</tr>
<tr>
<td>Design, Specify and Allocate Data</td>
<td>Design</td>
</tr>
<tr>
<td>Processing Modes</td>
<td></td>
</tr>
<tr>
<td>Define operating procedures</td>
<td>Design</td>
</tr>
<tr>
<td>Preliminary Design Review</td>
<td>Preliminary design review</td>
</tr>
<tr>
<td>Requirements Analysis</td>
<td>Requirement analysis</td>
</tr>
<tr>
<td>Software design and documentation</td>
<td>Design</td>
</tr>
<tr>
<td>Test planning</td>
<td>Develop plan</td>
</tr>
<tr>
<td>Test planning Documentation</td>
<td>Document</td>
</tr>
<tr>
<td>Critical Software Review</td>
<td>Critical design review</td>
</tr>
<tr>
<td>Coding</td>
<td>Coding</td>
</tr>
<tr>
<td>Visual Code Inspection</td>
<td>Inspection</td>
</tr>
<tr>
<td>Integration</td>
<td>Integration</td>
</tr>
<tr>
<td>Final Software Acceptance Review</td>
<td>Review</td>
</tr>
<tr>
<td>Software Installation and Documentation</td>
<td>Installation and checkout phase</td>
</tr>
<tr>
<td>Software System Diagnostic test</td>
<td>Testing</td>
</tr>
<tr>
<td>Functional Enhancement</td>
<td>Operation and maintenance phase</td>
</tr>
</tbody>
</table>

**TABLE 5**

<table>
<thead>
<tr>
<th>Normalized terms</th>
<th>Root terms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements specification</td>
<td>System Requirements analysis</td>
</tr>
<tr>
<td>Software requirements specification</td>
<td>Software Requirements analysis</td>
</tr>
<tr>
<td>Document</td>
<td>System architectural design</td>
</tr>
<tr>
<td>Design</td>
<td>Software architectural design</td>
</tr>
<tr>
<td>Design</td>
<td>Software architectural design</td>
</tr>
<tr>
<td>Preliminary design review</td>
<td>Software architectural design</td>
</tr>
<tr>
<td>Requirement analysis</td>
<td>System requirements analysis</td>
</tr>
<tr>
<td>Design</td>
<td>Software detailed design</td>
</tr>
<tr>
<td>Develop plan</td>
<td>Process implementation</td>
</tr>
<tr>
<td>Document</td>
<td>Process implementation</td>
</tr>
<tr>
<td>Critical design review</td>
<td>Software detailed design</td>
</tr>
<tr>
<td>Coding</td>
<td>Software coding</td>
</tr>
<tr>
<td>Inspection</td>
<td>Software coding</td>
</tr>
<tr>
<td>Integration</td>
<td>Software Integration, System</td>
</tr>
<tr>
<td>Review</td>
<td>Software Acceptance Support</td>
</tr>
</tbody>
</table>

Fig. 5. An abstract model of the Waterfall model

**VI. CONCLUSION**

Software Process Analysis Method (SAM) is a systematic framework for analyzing the nature of software processes. It comprises of three stages: elaboration, normalization and abstraction. The elaboration stage involves the decomposition of software processes until each activity reaches the atomic stage. Then, the normalization stage removes the confusion of specific terminologies by replacing them with standard dictionary terms. Finally, the abstraction stage categorizes the normalized terms into root terms and compresses them in order to produce a final abstract model. The abstract model reveals the nature of the software process. The information obtained from the abstract model can be used for further comparative analysis of software processes.

In the future, it is possible to minimize a major drawback of SAM in the normalization stage, that is, the occasional need for manual intervention to validate normalized terms. With a more elaborate process dictionary database and by including antonym relationship or artifact behaviors, the accuracy of the automatic normalization of terms is very
likely to improve. Additionally, it is also possible to adapt SAM to the use of formal methods in order to support a wider participation of the software community.

REFERENCES


