Annals of Software Engineering

Volume 14

Process-Based Software Engineering

Co-Editors

Yingxu Wang and Antony Bryant

2002
Table of Contents

Editors’ Introduction: Process-Based Software Engineering: Building the Infrastructures
Yingxu Wang and Antony Bryant

Three Process Perspectives: Organizations, Teams, and People
Watts S. Humphrey

eR&D - Effectively Managing Process Diversity
Christof Ebert and Jozef DeMan

Testing Processes of Web Applications
Filippo Ricca and Paolo Tonella

Integrating Process and Project Management for Multi-Site Software Development
Keith C. C. Chan and Lawrence M. L. Chung

Integrating Process Support and Knowledge Management for Virtual Software Development Teams
Frank Maurer and Harald Holz

Internet Software Engineering: A Different Class of Processes
Balasubramaniam Ramesh, Jan Pries-Heje, and Richard Baskerville

Automatic Verification of Static Policies on Software Process Models
Rodrigo Q. Reis, Carla A.L. Reis, Heribert Schlebbe, and Daltro J. Nunes

The Real-Time Process Algebra (RTPA)
Yingxu Wang

Software Evolution and Software Evolution Processes
Meir M. Lehman and Juan F. Ramil

A Comparative Review of Process-Centered Software Engineering Environments
Selma Arbaoui, Jean-Claude Derniame, Flávio Oquendo, and Hervé Verjus

Brian Henderson-Sellers

Process-Centered Software Engineering Environments: A Brief History and Future Challenges
Volker Gruhn

A Knowledge-Based Software Process Simulation Model
Noriko Hanakawa, Ken-ichi Matsumoto, and Koji Torii

Commitment Nets in Software Process Improvement
Pekka Abrahamsson
Editors’ Introduction

Process-Based Software Engineering: Building the Infrastructures

Yingxu Wang a and Antony Bryant b

a Dept. of Electrical and Computer Engineering, University of Calgary
2500 University Drive, NW, Calgary, Alberta, Canada T2N 1N4
Tel: (1) 403 220 6141, Fax: (1) 403 282 6855
Email: wangyx@enel.ucalgary.ca

b School of Information Management, Leeds Metropolitan University
Beckett Park, Leeds LS6 3QS, UK
Tel: (44) 113 283 7422, Fax: (44) 113 283 7599
Email: a.bryant@lmu.ac.uk

Abstract

A recent trend in software engineering is the shift from a focus on laboratory-oriented software engineering to a more industry-oriented view of software engineering processes. This complements preceding ideas about software engineering in terms of organization and process-orientation. From the domain coverage point of view, many of the existing software engineering approaches have mainly concentrated on the technical aspects of software development. Important areas of software engineering, such as the technical and organizational infrastructures, have been left untouched. As software systems increase in scales, issues of complexity and professional practices become involved. Software development as an academic or laboratory activity, has to engage with software development as a key industrialized process.

This expanded domain of software engineering exposes the limitations of existing methodologies that often address only individual sub-domains. There is, therefore, a demand for an overarching approach that provide a basis for theoretical and practical infrastructures capable of accommodating the whole range of modern software engineering practices and requirements. One approach is provided by Process-Based Software Engineering (PBSE); part of the more general trend towards a focus on process.

This paper provides a review of process techniques for software engineering and a high-level perspective on PBSE. Typical approaches and techniques for the establishment, assessment, improvement and benchmarking of software engineering process systems are introduced in this paper, and many are developed further in other contributions to this volume.

Keywords: Software engineering, software engineering processes, process-based software engineering (PBSE), software process modeling (SPM), software process establishment (SPE), software process assessment (SPA), software process improvement (SPI), and software process benchmarking (SPB).

1. Introduction

The concept of the software engineering process embodies a set of sequential practices that are functionally coherent and reusable for software engineering organization, implementation, and management. It is usually referred to as the software process, or simply the process. As such it is part of a more general trend that focuses on process, pushing structure and product into the background [Bryant 2000; Wang and King 2000].

Software engineering is a discipline that has emerged from computer science and is based on interdisciplinary theoretical and empirical methodologies. Initial approaches developed thus far have concentrated on technical aspects of software engineering, such as programming methodologies, software development
models, automated software engineering, and formal methods. While a cutting-edge approach, Process-Based Software Engineering (PBSE), has been developed in the last decade for integrating the modern domain of software engineering.

This expanded view of the domain exposes the limitations of conventional approaches, methodologies, and tools; but this is not to imply that the wealth of experience that they embody should be jettisoned. On the contrary, we would advocate the development of an inclusive and integrative approach that offers a suitable theoretical and practical infrastructure capable of accommodating both new demands and existing expertise: Hence the process-oriented view.

The software process approach towards software engineering encompasses systematic, organizational and managerial infrastructures for software engineering. It is necessary to expand the horizons of software engineering in this way because of the rapidly increasing complexity and scale demanded by software products. The need to ensure software quality and to increase productivity also provide impetus for PBSE.

The software process was originally devised and offered as a software management method [Humphrey and Sweet 1987; Humphrey 1988; Gilb 1988], a quality assurance approach [Evans and Marciniak 1987; ISO 9001 1989], or as a set of software development techniques [Curtis et al. 1987; Fayad 1997]. The reorientation of the software process to a software engineering process system reflects recent trends in seeking an ideal means for organizing and describing the whole process framework and infrastructure of software engineering.

Two events in 1987, the development of CMM and of ISO 9000, marked a significant development in the emergence of the software engineering process as a fully-fledged discipline. The software engineering process deals with foundations, modeling, establishment, assessment, improvement, benchmarking, and standardization of software processes. Generally, a process may be described as a set of linked activities that take an input and transform it to create an output. The software engineering process as a system is no different; it takes a software requirement as its input, while the software product is its output.

This paper explains the infrastructure of software engineering from a process perspective. Section 2 explores the concept of PBSE, and industrial practice based on PBSE. Section 3 introduces a software engineering reference model (SEPRM) and discusses the SEPRM process and capability models as well as describing an algorithm. The domain of PBSE is examined in Section 4, which covers software engineering process system establishment, assessment, improvement, and benchmarking. Technologies for formally describing a software engineering process system are introduced in Section 5 in order to develop rigorous and quantitative process technologies. Finally Section 6 provides an overview of the structure of PBSE and the configuration of this special volume on PBSE with 14 selected papers, which present cutting-edge research results and best practices on “Empirical Process Systems of Software Engineering” and “Theories and Environments of Software Engineering Processes.”

2. Process Models: The Infrastructure of Software Engineering

The technical and organizational infrastructures of software engineering rely on the software engineering processes. The processes of software engineering are complex systems as described by various process models and standards such as CMM [Humphrey and Sweet 1987; Humphrey 1988, 1995; Paulk et al. 1993, 1995], ISO 9001 [ISO 9001 1989], BOOTSTRAP [Koch 1993; Haase et al. 1994; Kuvaja et al. 1994], ISO/IEC 15504 [ISO/IEC 2000], and SEPRM [Wang et al. 1996, 1998, 1999a, b; Wang and King 2000]. These models collected a set of processes ranging from 18 to 51. This section comparatively explores current process models and the relationships among them. Then, the concept of PBSE is introduced.

2.1 Process Models as Infrastructures

The Software Engineering Institute (SEI) Software Capability Maturity Model (SW-CMM or CMM) was initially developed as an assessment model for software engineering management capabilities. As such it was expected that it would provide useful measures of organizations bidding or tendering for software contracts.

However, it was soon realized that the concept of process for software engineering had more utility than that of capability assessment. Software development organizations may use the process model as an infrastructure for internal process organization and improvement. As a result of this deeper understanding, new
practices in PBSE have been emerging in the last decade. This may be considered as one of the important inspirations arising from CMM and related research.

In the software industry, software development is commonly perceived as a one-off activity. On the other hand, one of the most interesting finding in software engineering practices is that the processes of software development are relatively stable, repeatable, and reusable. Therefore, software engineering processes can be adopted as the *infrastructure* for software engineering. This leads to the development of the concept and technology known as *PBSE*.

### 2.2 Practical Requirements for a Process Reference Model

In attempting a comparative analysis of the major existing process models, it is found that an intermediate *reference model* is quite useful for simplifying the many-to-many mappings into a many-to-one projection, and for reducing the complexity of mutual mapping of multiple models.

Existing work in one-to-one mapping among current process models is illustrated in Figure 1. Usually, conventional mappings were carried out at process level and were unidirectional [Wang et al. 1997a, b, 1999b]. For more accurate mapping, it is necessary that the mapping is conducted at the base process activity (BPA) level. However, terminology referring to the BPA differs from model to model, examples being: key practice (KP) in CMM, management issue (MI) in ISO 9001, quality system attribute (QSA) in BOOTSTRAP, and base practice (BP) in ISO/IEC 15504. All these elements of process or practices are generally referred to as BPA.

![Diagram](image)

**Figure 1. One-to-one mapping between current process models**

Because of structural differences between the current process paradigms and variations among their BPA domains, the comparison of two models needs mapping twice, once in each direction. This has been described as the *mapping asymmetry* between process models [Wang et al. 1997a, b]. Generally, for *n* models, the complexity in mutually mapping each other, *Cₙ*, is in the order of *n*², i.e.:

\[
C_n \triangleq O(n^2) = n \times (n-1)
\]

(1)

For example, for *n*=4 as shown in Figure 1, the mapping complexity is *Cₙ = n \times (n-1) = 4 \times (4-1) = 12*.

However, where an intermediate reference model is adopted as shown in Figure 2, the complexity of mapping *n* models via the reference model, *Cᵣ*, can be significantly reduced to:

\[
Cᵣ = O(n)
\]

\[
= n
\]

(2)

In the case of Figure 2, *Cᵣ = n = 4*. The efficiency has increased by three times by the use of the reference model based approach. Generally, the larger the *n*, the more effective is the reference model approach.

Another reason for requiring a reference model in mutually mapping and analyzing multi-models is that the fundamental BPAs defined in different process models result in a partial joint domain. Thus, some of the
mappings at the BPA level would not exist, i.e. some mappings of BPAs would result in an empty set. However, with a reference model that defines a superset of BPAs to cover the entire process domain that all current models form, the mapping will never be empty. This is a necessary and practical requirement in multi-model mapping.

![Figure 2. The role of a software process reference model](image)

**2.3 Industrial Practices on Process-Based Software Engineering (PBSE)**

For *modeling* a process system, processes are elicited and integrated from the bottom, up. Processes in the development subsystem are first analyzed and modeled. Corresponding to the development processes, the management processes are then deployed as measures to support and control the development processes. The third step is to design the organization processes, which are the top-level management processes oriented to the whole software development organization, and which are applicable to all software engineering projects within the organization.

It is generally considered that there would be a number of parallel development and management processes for individual projects within a software development organization. For the purpose of *controlling* a process system, software engineering processes are implemented and activated top down, from the organization level to the project level. Therefore, the relationship between the *organization*, *management*, and *development processes* can be further refined [Wang and King 2000] as shown in Figure 3.

![Figure 3. Practices in process-based software engineering](image)
processes and reflects the benchmarked best practices in the software industry. At project level, a number of parallel development and management processes may exist based on the individual project’s tailored process model (PTPM) and which are derived models of the OPRM reference model. In Figure 3 the process reference model, OPRM, is the key for empirical PBSE. If an OPRM is well established in an organization, the PTPMs at project level can easily be derived.

At the top level, a software development organization may adopt an existing international standard or an established process model as its OPRM; or, it can develop a specific organization-oriented OPRM based on the existing models and the organization’s own practices and experiences in software engineering. The OPRM plays a crucial role in the regulation, coordination, and standardization of an organization’s software engineering practices.

At project level, the OPRM reference model could, and usually should, be tailored or adapted to a specific project according to the nature of the project, taking into account application domain, scope, complexity, schedule, experience of project team, reuse opportunities identified and/or resources availability, and so on. For a PTPM of an individual project, the management and development processes should be one-to-one designed and synchronized. Tailoring of a PTPM from a comprehensive OPRM makes the software project leaders’ tasks dramatically easier. Using this approach, project organization and conduct can be effectively performed within an organization’s unified software engineering process infrastructure.

3. The Software Engineering Process Reference Model (SEPRM)

The software engineering process reference model (SEPRM) [Wang et al. 1996, 1998, 1999a, b; Wang and King 2000] identifies a superset of processes and BPAs that covers the domains of current process models and new areas for software engineering environment and supporting tools. The philosophy of SEPRM is to provide a comprehensive and integrated software engineering process system reference model. SEPRM demonstrates a unified process system framework for PBSE, in which all current process models can be located.

SEPRM develops a basis against which process capability levels between existing process models can be systematically and quantitatively compared. It also allows transformation between process capability levels within different process models, and it enables software development organizations to relate their process capabilities to a range of different process models.

The following subsections describe the process model, process capability model, and evaluation method of SEPRM.

3.1 The SEPRM Process Model

SEPRM models a software engineering process system as 3 process subsystems, 12 process categories, 51 processes and 444 base process activities. A high level hierarchical structure of the SEPRM framework is shown in Figure 4.

Details of 444 BPAs, and their configuration in the SEPRM reference model can be found in [Wang and King 2000], including the relationships between the defined BPAs and their counterparts in current process models.
The Software Engineering Process Reference Model
SEPRM

PS.1
Organisation Process Subsystem

PS.2
Development Process Subsystem

PS.3
Management Process Subsystem

PC.1
Organisation structure

PC.2
Organisational process

PC.3
Customer service

PC.4
Software engineering methodologies

PC.5
Software development

PC.6
Software development environment

PC.7
Software quality assurance

PC.8
Project planning

PC.9
Project management

PC.10
Contract and requirement management

PC.11
Document management

PC.12
Human resource management

Figure 4. Hierarchical structure of SEPRM
3.2 The SEPRM Process Capability Model

Parallel to the SEPRM process dimension, the process capability model describes another dimension of SEPRM that provides an assessment framework for each process defined in the process model.

The SEPRM process capability model consists of a practice performance scale, a process capability scale and a process capability scope. A practice performance rating scale for the BPAs in SEPRM is defined in Table 1. It employs a four-level scale for evaluating a BPA’s existence and performance. The rating thresholds provide a set of quantitative measurement for rating a BPA’s performance with the scale.

<table>
<thead>
<tr>
<th>Scale</th>
<th>Description</th>
<th>Rating threshold</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 (F)</td>
<td>Fully adequate</td>
<td>90% - 100%</td>
</tr>
<tr>
<td>3 (L)</td>
<td>Largely adequate</td>
<td>70% - 89%</td>
</tr>
<tr>
<td>1 (P)</td>
<td>Partially adequate</td>
<td>35% - 69%</td>
</tr>
<tr>
<td>0 (N)</td>
<td>Not adequate</td>
<td>0 – 34%</td>
</tr>
</tbody>
</table>

Table 1
Performance Rating Scale of the BPAs

There are three types of process capability scales: the pass-threshold-based, process-management-oriented, and process-oriented. The SEPRM process capability model is designed for directly rating and characterizing the performance of a process within context, rather than to indirectly evaluate the management maturity level for a process.

SEPRM develops a six-level software process capability model as shown in Table 2, with a set of defined criteria for rating the capability of a process.

<table>
<thead>
<tr>
<th>Capability Level (CL[I])</th>
<th>Description</th>
<th>Process Capability Criteria</th>
<th>Manner of individual Activities</th>
</tr>
</thead>
<tbody>
<tr>
<td>CL[0]</td>
<td>Incomplete</td>
<td>C[0,1] No process system reference model</td>
<td>C[0,3] Ad hoc</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C[0,2] No defined and repeatable process activities</td>
<td></td>
</tr>
<tr>
<td>CL[1]</td>
<td>Loose</td>
<td>C[1,1] There are defined processes at some extent</td>
<td>C[1,3] Varying</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C[1,2] Three are limited process activities defined and conducted</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>C[2,2] There are relatively complete process activities defined and aligned to organization reference model</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>C[3,2] There are complete process activities derived from organization reference model</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>C[4,2] - There are completed process activities derived from organization reference model</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Performances of processes are monitored</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>C[5,2] - There is a completed derived process model</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Performances of processes are quantitatively monitored and fine-tuned</td>
<td></td>
</tr>
</tbody>
</table>

Table 2 shows that, in SEPRM, a process as an independent unit is assessed in the organization, project and individual contexts against the six level process capability criteria. In order to relate the process capability criteria to the performance of BPAs in a process, there is an additional threshold for assessing a process. This is the average performance of the BPAs. Thus, based on both the software process capability model and the BPA performance threshold, an SEPRM process capability scale is described in Table 3.
### Table 3

The SEPRM Process Capability Scale

<table>
<thead>
<tr>
<th>Capability Level (CL[i])</th>
<th>Description</th>
<th>Process Capability Criteria</th>
<th>BPA Average Performance Threshold</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Organization Scope</td>
<td>Project Scope</td>
</tr>
<tr>
<td>CL[0] Incomplete</td>
<td></td>
<td>C[0,1] No</td>
<td>C[0,2] No</td>
</tr>
</tbody>
</table>

It can be seen in Table 3 that there are four criteria that a process has to fulfill at each capability level. The first three are oriented to a process as a whole; while the last one is oriented to BPAs contained in a process. Therefore, the capability of a software development organization to operate a given process is determined by the maximum level \(i\) that a process achieved for fulfilling all four criteria for that level.

The SEPRM process assessment results are reported at the six levels plus a decimal value. This means it has the potential to distinguish the process capability at tenth-sublevels. This approach enables a software development organization to fine-tune its process system in continuous process improvement.

### 3.3 The SEPRM Process Capability Determination Methodology

Using the formal definitions of the SEPRM process model and process capability model as developed in Sections 3.1 and 3.2, we can now consider how to apply the latter to the former for the assessment of process capability at practice, process, project and organization levels. This activity is called process capability determination.

The SEPRM process capability determination methodology [Wang and King 2000] can be described as an algorithm as shown below:

**Algorithm 1. The SEPRM process capability determination algorithm**

Assume:
- \(N_{PC}(SUBSYS)\) - Number of process categories in a process subsystem
- \(N_{PROC}(SUBSYS,PC)\) - Number of processes in a category
- \(N_{BPA}(SUBSYS,PC,PROC)\) - Number of BPAs in a process
- \(BPA(SUBSYS,PC,PROC)\) - A BPA index
- \(CL\) - A capability level
- \(PCL_{proc}(SUBSYS,PC,PROC)\) - A process capability level
- \(PCL_{proj}\) - Capability level of a project

Input: Sample indicators of BPA and processes existence and performance

Output: A process profile: \(PCL_{proc}[SUBSYS,PC,PROC]\), and a project process capability level: \(PCL_{proj}\)

Begin

// Step 1: Initialization
// Define numbers of processes in each process subsystem and category according to [Wang and King 2000] ...

// Define numbers of BPA's in each process [Wang and King 2000] ...

// Step 2: Practice performance rating

for SUBSYS := 1 to 3 do // the process subsystem index
    for PC := 1 to Npc(SUBSYS) do // the process category index
        for PROC := 1 to Nproc(SUBSYS, PC) do // the process index
            begin
                PP(SUBSYS, PC, PROC) := 0;

                for BPA := 1 to NBPA(SUBSYS, PC, PROC) do
                    // The BPA index
                    begin
                        // Assess a BPA according to Table 1, and
                        // record performance rating in BPA(SUBSYS, PC, PROC)
                        case BPA(SUBSYS, PC, PROC)
                            F: // Fully adequate
                                PP(SUBSYS, PC, PROC) :=
                                    PP(SUBSYS, PC, PROC) + 5;
                            L: // Largely adequate
                                PP(SUBSYS, PC, PROC) :=
                                    PP(SUBSYS, PC, PROC) + 3;
                            P: // Partially adequate
                                PP(SUBSYS, PC, PROC) :=
                                    PP(SUBSYS, PC, PROC) + 1;
                            N: // Not adequate
                                PP(SUBSYS, PC, PROC) :=
                                    PP(SUBSYS, PC, PROC) + 0;
                            NA: // Does not apply
                                PP(SUBSYS, PC, PROC) :=
                                    PP(SUBSYS, PC, PROC) + 5;
                        end;
                    end;
            end;
end;

// Step 3: Process capability determination

for SUBSYS := 1 to 3 do // the process subsystem index
    for PC := 1 to Npc(SUBSYS) do // the process category index
        for PROC := 1 to Nproc(SUBSYS, PC) do // the process index
            // 3.1 Assess each process against the six level
            // process criteria as defined in Table 3
            CLproc(SUBSYS, PC, PROC) :=
                max { i \ (C[i,j] are fulfilled) } ^ j=1,2,3;

            // 3.2 Assess mean BPA performance according to Table 1
            CL_bpa(SUBSYS, PC, PROC) := PP(SUBSYS, PC, PROC) / NBPA(SUBSYS, PC, PROC);
// 3.3 Determine process capability levels
\[
CL(SUBSYS, PC, PROC) := \\
\min \{CL_{PROC}(SUBSYS,PC,PROC)+0.9, \\
CL_{BPA}(SUBSYS,PC,PROC)\};
\]

// 3.4 Save process capability profile
\[
PCL_{proc}(SUBSYS,PC,PROC) := CL(SUBSYS,PC,PROC);
\]

// Step 4: Project capability determination
\[
k := 51; \quad \text{// Number of PROCs defined in SEPRM}
CL := 0;
\]
for \( SUBSYS := 1 \) to 3 do \( // \) the process subsystem index
for \( PC := 1 \) to \( N_{PC}(SUBSYS) \) do \( // \) the process category index
for \( PROC := 1 \) to \( N_{proc}(SUBSYS, PC) \) do \( // \) the process index
\[
\text{// Calculate cumulated process capability value}
CL := CL + PCL_{proc}(SUBSYS,PC,PROC);
\]
\[
\text{// Derive capability level of the project}
PCL_{proj} := CL / k; \quad \text{// Calculate project capability level}
\]
End

An SEPRM assessment according to Algorithm 1 is carried out in four steps:

(a) Initialization: This step is designed to specify the numbers of BPAs defined in SEPRM. For obtaining a detailed configuration of BPAs in the SEPRM process model, reference may be made to Appendix C in [Wang and King 2000].

(b) BPA performance rating: In this step, all BPAs for each process are rated according to the definitions of practice performance scale in Table 1. The basic function of this step is to count the total values of the rated BPAs within individual processes.

(c) Process capability determination: This step first derives both the process capability ratings by the process criteria and the BPA performance criteria according to the definitions of Table 3. Next, the capability level of the process is determined by taking into account of both the first three criteria (the qualitative score) and the fourth criterion (the quantitative score). Then, a process capability profile of an SEPRM assessment is created.

(d) Project process capability determination: In the final step, the algorithm derives a process capability level for a software project based on all processes’ capability levels derived in Step 3. The project capability level will be reported with the addition of a process capability profile.

SEPRM establishes a comprehensive and unified process system reference model. The development of SEPRM was based on the great inspiration derived from existing process models and the plentiful experience of empirical software engineering. From this we have gained improved understanding on software engineering and on software process system modeling as a key for organization and implementing software engineering. SEPRM’s process model is supported by a set of industrial benchmarking data, and its process capability model is independently operational at levels of organization, project and individual software engineer. SEPRM enables a derived process capability level to be transformed into other process models; and further, it allows, for the first time, capability levels from different process models to be related and compared [Wang and King 2000].

4. Process-Based Software Engineering

In the literature of software engineering process research, it has been assumed that a process system should have already existed in a software development organization so that a process assessment and improvement
project could be carried out directly. However convenient this assumption is, it is not true that the majority of software organizations have formal, definable processes.

In reality, a process assessment project starts by the mapping of a software organization’s existing processes to a process model that has been chosen for the assessment. The usual cases are that a software development organization has only some loose and informal practices, rather than a defined and coherent process system. This scenario leads to the observation that rigorous PBSE has to start from process establishment rather than process assessment in a software development organization. Therefore, the right order of events in achieving software engineering process excellence in an organization is first, process establishment; second, process assessment; and then process improvement as shown in Figure 5.

![Figure 5. Process-based software engineering](image)

4.1 Software Engineering Process System Establishment

An initial and fundamental step in process-based software engineering is process system establishment. The major aim of process establishment is to build up a software engineering process reference model for the software development organization. When a process system is established and experienced, improvement can be initiated effectively via process assessment and benchmarking.

4.1.1 Procedure to Derive a Software Project Process Model

This subsection explores three basic steps for deriving a software project process model.

(a) Select and reuse a process system reference model at organization level

The most efficient way to establish a process system is to reuse a standard or well-accepted process model. In selecting an existing process model as an organization’s reference model, one of the key issues is that the reference model should be reasonably comprehensive in order to enable an easy derivation of working process models at project level. Another issue is that the reference model should be able to serve many purposes in software engineering such as multi-type process assessment, improvement, training, and internal standardization. The third issue is the flexibility of the reference model, i.e. the selected reference model should allow incorporation of the host organization’s experience and special needs into the reference model and derived models.

When an organization’s process system is determined, the next step is to uphold it as the organization’s official and unified software engineering platform. Based on this, various process models should be derived for different projects.

(b) Derive a process model at project level

Before commencing a new project, the first thing that a project manager needs to do is to derive the project’s process model as the infrastructure for the project. The project process model will serve as a blueprint
for organizing all activities that are going to be enacted within the scope of the project, including technical, managerial, organizational, customer, and supporting activities.

A checklist of factors for consideration in deriving a project process model from the reference model is shown in Table 4. When all factors are weighted by high (H), Medium (M), or low (L), a rating for what kind of project process model is needed can be determined according to Expressions 3 and 4.

Assume that $S_i$ is the $i$th weight for factor $i$ and $n$ is the number of total factors; the average score, $S$, or the level of requirement for a derived model is defined as:

$$S = \frac{1}{n} \sum_{i=1}^{n} S_i$$

According to the average score $S$, the type of derived model determined by the weighted factors can be estimated as follows:

$$S > 3, \text{ the need is for a complete project process model}$$

$$S = 3, \text{ the need is for a medium project process model}$$

$$S < 3, \text{ the need is for a light project process model}$$

For instance, applying Expression 4 to the weights of the ten factors as shown in Table 4 results in an estimated average score $S = 3.6$. According to Expression 4, the project process model has to be a relatively complete model that covers almost related process areas modeled in the reference model.

Table 4

Determining Type of Derived Process Models for a Project

<table>
<thead>
<tr>
<th>No.</th>
<th>Project Factor</th>
<th>Weight</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Importance</td>
<td>H</td>
<td>S1 = 5</td>
</tr>
<tr>
<td>2</td>
<td>Difficulty</td>
<td>M</td>
<td>S2 = 1</td>
</tr>
<tr>
<td>3</td>
<td>Complexity</td>
<td>L</td>
<td>S3 = 5</td>
</tr>
<tr>
<td>4</td>
<td>Size</td>
<td>H</td>
<td>S4 = 3</td>
</tr>
<tr>
<td>5</td>
<td>Domain knowledge requirement</td>
<td>M</td>
<td>S5 = 3</td>
</tr>
<tr>
<td>6</td>
<td>Experience requirement</td>
<td>M</td>
<td>S6 = 5</td>
</tr>
<tr>
<td>7</td>
<td>Special process needed</td>
<td>L</td>
<td>S7 = 1</td>
</tr>
<tr>
<td>8</td>
<td>Schedule constraints</td>
<td>H</td>
<td>S8 = 3</td>
</tr>
<tr>
<td>9</td>
<td>Budget constraints</td>
<td>L</td>
<td>S9 = 5</td>
</tr>
<tr>
<td>10</td>
<td>Other process constraints</td>
<td>H</td>
<td>S10 = 5</td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td></td>
<td>S = 3.6</td>
</tr>
</tbody>
</table>

Note: H = High (5), M = Medium (3), L = Low (1).

Note the factors shown in Table 4 are examples for demonstrating how the type of project process model can be determined in a formal way. It is by no means exhaustive. Therefore, readers may add, delete, and/or modify the factors in order to make them suitable for their specific projects.

(c) Apply the derived project process model

When a project process system model is derived, the next step is to accept, as a common platform, the process model at both project and individual levels, and to apply the project process model to all activities within the project scope.

It can be seen that the reference model approach to implement software engineering provides project managers with a means to derive and organize a project process model in a consistent and transparent manner. It also provides software engineers and others in a software project with a clear picture of their roles, interactions, and relationships to each other.

4.1.2 Methods for Deriving a Software Project Process Model
In establishing a process model for a software project, three types of methods may be introduced. They are process model tailoring, extension, and adaptation; ordered increasingly according to their technical difficulty in applications.

(a) **Process model tailoring** is a model customization method for making a process model suitable for a specific software project by deleting unnecessary processes.

Model tailoring is the simplest method to derive a project process model from a comprehensive organizational process reference model. The only technique is to delete what is not needed in order to establish a specific software project based on an understanding of both the reference model and the nature of the project.

(b) **Process model extension** is a model customization method for making a process model suitable for a specific software project by adding additional processes.

Model extension requires a project manager capable of integrating new processes, adopted from either process models or best practices repositories, into the current project process model or organizational process reference model. When new processes are introduced, a validation phase is needed for monitoring their fitness and performance in the whole process system.

(c) **Process model adaptation** is a model customization method for making a process model suitable for a specific software project by modifying, updating, and fine-tuning related processes.

Model adaptation is useful when a project manager is experienced with respect to one process reference model and prepared to monitor the performance of adapted processes during a project life span. All of the above three approaches for process model derivation and establishment can be used individually or together to derive an effective project process model for software engineering.

### 4.2 Software Engineering Process System Assessment

It is believed that if one cannot measure a process system, one cannot improve it. Therefore, *software process assessment (SPA)* is critical for process improvement. Various methodologies for SPA have been developed in the last decade. This section describes the integrated SPA framework, and demonstrates that current process models, such as CMM, ISO 9001, BOOTSTRAP, ISO/IEC 15504, and SEPRM, can be perfectly fitted into this framework.

#### 4.2.1 Process Assessment Methods from the Viewpoint of Reference Systems

From the viewpoint of reference systems there are four types of assessment methods - model-based, standard-based, benchmark-based, and integrated (model-and-benchmark-based) assessment.

(a) **Model-based assessment** is an SPA method by which a software development organization is evaluated against a specific process and capability model, and according to a specific capability determination method provided by the model.

Model-based assessment is a kind of *absolute* assessment approach. Using this approach, a software development organization is evaluated against a fixed process framework and a defined capability scale. The assessment result reports a capability level of a software development organization against the capability scale of the model. CMM and BOOTSTRAP are examples of model-based assessment methodologies.

(b) **Standard-Based Assessment** is an SPA method by which a software development organization is evaluated against a specific process and capability model defined by a standard, and according to a specific capability determination method provided in the standard.

Standard-based assessment is a special type of model-based assessment method. It also provides an absolute assessment approach by which a software development organization’s process capability is rated against a defined capability scale. ISO/IEC 15504 and partially ISO 9001 are examples of standard-based assessment methodologies.

(c) **Benchmark-Based Assessment** is an SPA method by which a software development organization is evaluated against a set of benchmarks of software processes, and according to a specific capability determination method.

Benchmark-based assessment is a kind of *relative* assessment approach. By this approach a software development organization is evaluated against a set of benchmarks. Thus, the assessment result associated with a software development organization’s capability level may be presented in three relative levels: below, equal, or above the benchmark of each process.
(d) **Integrated Assessment** is a kind of composite model-based and benchmark-based SPA method in which a software development organization is evaluated against both a benchmarked process model and a capability model, and according to a specific capability determination method provided in the model.

The integrated assessment method inherits the advantages of both absolute and relative SPA methods as described in this section. Using the integrated assessment method, a software development organization can be evaluated against both a benchmark and an absolute capability scale at the same time. The SEPRM model is such an integrated SPA model. Another advantage of the integrated assessment method is its ability to provide a quantitative guide for software process improvement.

4.2.2 **Process Assessment Methods from the Viewpoint of Model Structures**

From the viewpoint of model framework structures, there are three types of assessment methods. They are: checklist-based assessment, 1-D process-based assessment, and 2-D process-based assessment, as illustrated in Figure 6.

Figure 6 shows that a 2-D process model allows all processes to be performed and rated at any process capability level. A 1-D process model is a special case of 2-D models, where a group of processes are defined and rated at a certain capability level. For example, according to the 1-D process model, processes 7 – 13 in Figure 6 can only be performed, and therefore rated at level 3 or below. Similarly, the checklist-based process model is a simpler 1-D process model, where all processes are defined and rated at a single level with equal importance.

(a) **Checklist-based assessment** is an SPA method that is based on a pass/fail checklist for each practice and process specified in a process model.

A checklist-based assessment model is the simplest assessment methodology. This kind of method is only suitable for SPA. It is not much help in step-by-step process improvement. The ISO 9001 model provides a checklist-based assessment method.

(b) **1-D process-based assessment** is an SPA method that determines a software development organization’s capability from a set of processes in a single process dimension.

The 1-D assessment is an extension of the checklist-based assessment. This type of model is suitable for process improvement in project or organization scopes while, at the same time, being relatively weak in detailed process scope simply because processes have been grouped and pre-allocated at specific capability levels as shown in Figure 6. CMM and BOOTSTRAP are examples of 1-D assessment models.

An issue presenting in such methods is that there are no widely accepted criteria prescribing how a set of software processes are grouped and mapped onto different capability levels. In principle, the processes defined in a model would be practiced at any capability level. That is, software processes in practice have no inherited capability levels; only the software development organizations and the people who are performing the processes can be measured by capability levels.

(c) **2-D process-based assessment** is an SPA method that employs both process and capability dimensions in a process model, and derives process capability by evaluating the process model against the capability model.

The 2-D assessment method enables every process in the process dimension to be performed and evaluated against the capability dimension at all levels. This is a flexible approach to software process assessment,
although effort spent in a 2-D process assessment would be much higher than that of a 1-D or checklist assessment. This type of model is suitable for process improvement from process scope to project and organization scopes because it provides precise measurement for every process at all the capability levels. ISO/IEC 15504 and SEPRM are examples of 2-D assessment models.

Conventionally, 1-D methods were considered to have provided a process dimension in process assessment. By comparing this with the 2-D assessment methods described above and in Figure 8, it may be predicted that there is another kind of 1-D process assessment model which implements only the capability dimension, while leaving the process dimension open for a software development organization or the process model providers to design and implement. This would provide a level of flexibility in software process assessment and standardization.

4.3 Software Engineering Process System Improvement

Software engineering process system improvement is the goal of process assessment, acting on issues found in an assessment and enhancing the effectiveness of processes in the process system. This section attempts to describe major philosophies in software process improvement (SPI) and alternative SPI methodologies.

4.3.1 Software Process Improvement Philosophies and Approaches

There are various philosophies underpinning SPI. Key categories of SPI philosophy are goal-oriented process improvement, benchmark-based process improvement, and continuous process improvement. This subsection discusses philosophies behind the process improvement methodologies. The usability of various SPI approaches and their relationships are also commented upon.

(a) Goal-oriented process improvement is an SPI approach by which process system capability is improved by moving towards a predefined goal, usually a specific process capability level.

This approach is simple, and is the most widely adopted philosophy in software engineering. For example, ISO 9001 provides a pass/fail goal with a basic set of requirements for a software process system. CMM, ISO/IEC 15504, and SEPRM provide a 5/6-level capability scale that enables software development organizations to set more precise and quantitative improvement goals.

(b) Benchmark-based process improvement is an SPI approach by which process system capability is improved by moving towards an optimum combined profile according to software engineering process benchmarks, rather than a maximum capability level.

This is a realistic and pragmatic philosophy for process improvement. It is argued that in order to maintain sufficient competence, a software organization does not need to push all its software engineering processes to the highest level because it is neither necessary nor economic. This philosophy provides alternative thinking to the idea “the higher the better for process capability” as is presented in the goal-oriented process improvement approach.

Using the benchmark-based improvement approach, an optimized process improvement strategy identifies a sufficient (the minimum required) and economic target process profile, which provides an organization with sufficient margins of competence in every process. It does not necessarily set them all at the highest level of a capability scale.

(c) Continuous process improvement is an SPI approach by which a process system’s capability is required to be improved all the time, and toward ever higher capability levels.

This is considered an oriental philosophy that accepts no top limits or discrete goals because “ideal” standards are continuously changing. It is this assumption that change is normal that is in tune with modern management theory. Continuous process improvement has been proven effective in engineering process optimization and quality assurance. Using this approach, SPI is a continuous, spiral-like procedure. The Deming Circle, plan-do-check-act, is a typical component of this philosophy.

In continuous process improvement there is no end to process optimization, and all processes are supposed to be improved all the time. There is a criticism that the goals for improvement are not explicitly stated in this philosophy. Therefore, when adopting continuous process improvement, top management should make clear the current goals, as well as the short, middle, and long-term ones.

Generally, goal-oriented methodologies will still constitute the mainstream in SPI. While 2-D process models provide more precise process assessment results, and the benchmark-based process models provide
empirical indications of process attributes, benchmark-based improvement, will gain wider application. Also, the continuous process improvement approach will provide a basis for sustainable long-term strategic planning.

4.3.2 Software Process Improvement Methodologies

The above discussion on the philosophies for process improvement yields the basis for an investigation of possible software process improvement methodologies. There are two basic SPI methods – assessment-based and benchmark-based process improvement. The former improves a process system from a given level in a defined scale to a next higher level; the latter provides improvement strategies by identifying gaps between a software development organization’s process system and a set of established benchmarks. In addition, a combined approach may be adopted.

(a) Model-Based Improvement is an SPI method by which a process system can be improved by basing its performance and capability profile on a model-based assessment.

Using this idea, the processes inherent in a software development organization are improved according to a process system model with step-by-step suggestions. CMM and BOOTSTRAP are examples of such a model-based process improvement methodology.

(b) Standard-Based Improvement is an SPI method in which a process system can be improved by basing its performance and capability profile on a standard-based assessment.

Using this approach, the processes inherent in a software development organization are improved according to a standardized process system model. ISO/IEC 15504 provides a standard-based improvement method. However, it is noteworthy that ISO 9001 is probably not suitable because it lacks a process improvement model and a step-by-step improvement mechanism as analyzed in Section 4.3.1.

(c) Benchmark-Based Improvement is an SPI method in which a process system can be improved by basing its performance and capability profile on a benchmark-based assessment.

Benchmark-based improvement is a kind of relative improvement approach. Using this approach, the processes inherent in a software development organization are improved according to a set of process benchmarks. It provides an optimized and economical process improvement solution. SEPRM is the first benchmarked model for enabling benchmark-based process improvements [Wang et al. 1998, 1999a].

(d) Integrated Improvement is a combined model-based and benchmark-based SPI method in which the process system can be improved by basing its performance and capability profile on an integrated model-based and benchmark-based assessment.

The integrated process improvement method inherits the advantages of both absolute and relative SPI methods. Using the integrated improvement method, the processes of a software development organization are improved according to a benchmarked process system model. SEPRM is designed to support integrated model- and benchmark-based process improvement.

4.4 Software Engineering Process Benchmarking

A benchmark for a software process system is a set of statistical reference data that represents the average performance and industrial norms of all processes in software engineering practices. Software process benchmarking is one of the important methodologies in software process engineering. Benchmarking is useful in both process assessment and improvement.

A few process benchmarks, such as the European process benchmarks developed by IBM [IBM 1996] and the Swedish national software engineering process benchmarks [Wang et al. 1999c], have been developed in later 1990s. These process benchmarks covered a small set of processes. To extend the work, a comprehensive set of process benchmarks has been developed recently [Wang et al. 1999a; Wang and King 2000] for the software engineering process reference model (SEPRM), which provides a superset of 51 software engineering process benchmarks characterized by 444 BPAs as shown in Figure 7.

The establishment and availability of software engineering benchmarks provide a new possibility for benchmark-based process assessment and improvement. Although the conventional goal-based process assessment and improvement technologies have been widely accepted, its philosophy of “the higher the better” has been questioned in practice. Specifically it has been found that the determination of target capability levels for a software organization in the goal-based approach would be virtual, infeasible, and sometimes overshoot [Wang and King 2000]. Benchmark-based process improvement provides a new approach to adaptive and
relative process improvement based on a philosophy of “the smaller the advantage, the better” [Wang and King 2000]. According to the benchmark-based process improvement method, the target capability levels of a software organization may be set related to the benchmarks of the software industry or a specific sector, rather than a virtual higher capability level according to a goal-based process model.

Observing the benchmarks of the SEPRM process reference model as shown in Figure 7, the target capability levels for benchmark-based process improvement can be divided into three categories: the basic level, the competitive level, and the advanced level.

<table>
<thead>
<tr>
<th>Process Capability Level</th>
<th>Benchmark</th>
<th>Basic</th>
<th>Competitive</th>
<th>Advanced</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>0.5</td>
<td>1.5</td>
<td>2.5</td>
<td>3.5</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>1.5</td>
<td>2.5</td>
<td>3.5</td>
<td>4.5</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

Figure 7. The SEPRM software engineering process capability benchmarks

The basic level is the minimum level of process capability that a software organization should achieve in order to develop quality software according to the SEPRM benchmarks. It is suitable as a target for initial software organizations that are in the early stages of software process establishment and improvement.

The competitive level is an average level of process capability that ordinary software organizations have reached in software development according to the SEPRM benchmarks. It is suitable as a target for the established software organizations that pursue a stable software engineering process system and systematic process improvement.

The advanced level is the highest level of process capability that has been achieved by the top 10% of software development organizations according to the SEPRM benchmarks. It is suitable as a target for the experienced software organizations that aim at optimizing existing process system and producing high quality software for complicated and/or mission-critical systems.

The key value of the software engineering benchmarking technologies is the establishment of the industrial norms and the quantitative measurement of common and best practices in different regions. On the basis of the benchmarks, software organizations are able to determine their current positions in a region, and to compare their practices against peers in the same sector. One of the major application areas of software engineering benchmarking is benchmark-based process improvement. Another application area of benchmarking is to enable software development organizations to compare and better manage their process improvement activities through benchmarking analysis.

5. Formal Description of Software Engineering Process Systems

Conventionally, software engineering process systems are designed and described by natural languages. There is a requirement to seek more accurate and precise technologies for software process system description. The Real-Time Process Algebra (RTPA) [Wang 2002a, b] has been developed to support formal description of process system architectures and behaviors. A process can be a single meta-process or a complex one that is built upon meta-processes by using a set of process combination rules – the process relations.
There are 16 meta-processes and 16 process relations developed in RTPA. The former identify a set of essential elements for modeling a software system; the latter provide a set of algebraic process relations for building complex processes. Based on the RTPA notations, any architecture and behavior of software process systems can be sufficiently described.

Using RTPA we are able to describe formally a process model based on a coherent set of mathematical notations. The formal description is useful for providing precise and accurate definitions of the structure and interrelationship of a process system, and to avoid ambiguity inherent in natural language descriptions. For instance, the algorithm of the CMM model can be formally described below.

\[ \text{CMM Algorithm} \triangleq \text{CMMAlgorithm.Architecture} \]
\[ \| \text{CMMAlgorithm.StaticBehaviors} \]
\[ \| \text{CMMAlgorithm.DynamicBehaviors} \]

\[ a. \text{ CMM Algorithm Architecture} \]
\[ \text{CMMAlgorithm.Architecture} \triangleq \begin{align*}
\| \text{Input : ST} \\
\| \text{Output : N} \\
\| \text{CLM : ST} \\
\| \text{Events : S} \\
\| \text{Status : BL} \\
= \text{InputPars: (\langle Level : N | 1 \leq \text{Level N} \leq 5,} \\
\quad \langle \text{NKP[level N]} : N | \text{NKP[1]}N=0, \text{NKP[2]}N=62, \text{NKP[3]}N=50, \\
\quad \text{NKP[4]}N=12, \text{NKP[5]}N=26,} \\
\quad \langle \text{KP[level N, iN]} : N | 1 \leq \text{KP N} \leq 4, 1 \leq iN \leq \text{NKP[level N]} > \\
\quad \text{)} \\
\| \text{CMM_PCL : N | 1 \leq \text{CMM-PCL N} \leq 5} \\
\| \text{CLM: (\langle \text{RKP[level N, iN]} : N | 1 \leq \text{RKP N} \leq 4, 1 \leq \text{Level N} \leq 5, 1 \leq iN \leq} \\
\quad \text{NKP[level N]},} \\
\quad \langle \text{SAT KP[level N]} : N | 1 \leq \text{Level N} \leq 5,} \\
\| @\text{CMMAlgorithmS} \\
\| CMMResultBL
\end{align*} \]

\[ b. \text{ CMM Algorithm Static Behaviors} \]
\[ \text{CMMAlgorithm.StaticBehaviors} \triangleq \text{KP_PerformanceRating} \]
\[ \| \text{CMM_PCL_Determination} \]
\[ \text{KP_PerformanceRating } \{(I:: \text{NKP[1]}N=0, \text{NKP[2]}N=62, \text{NKP[3]}N=50, \text{NKP[4]}N=12, \text{NKP[5]}N=26, \{\text{KP[level N, iN]}}\}; \\
\{(O:: \{(\text{SAT KP[level N]}, \{\text{RKP[level N, iN]}})}\) \triangleq \]

\[ {(// \text{ CMM key practice performance rating}} \]
\[ \frac{\text{R}}{\text{level}=2} (\text{SAT KP[level N]} := 0) \]
\[ \rightarrow nN := \text{NKP[level N]} \]
\[ \rightarrow \frac{\text{R}}{\text{level}=2} (\text{KP[level, l]S := 'satisfied'}} \]
\[ \rightarrow \frac{\text{R}}{\text{level}=2} (\text{KP[level, l]S := 'not satisfied'}} \]
\[ \rightarrow \frac{\text{R}}{\text{level}=2} (\text{KP[level, l]S := 'deosn't apply'}} \]
→ ↑ (SATKP[level]|N)

| ? KP[level, i]|S := ‘don’t know’
→ RKP[level, i]|N := 1
)


{O:: CMM_PCLN, ¤CMMResultBL}) ≜ △

{ // CMM process capability level assessment
   → PCLN := 1 // CMM initial level
   → PCLN := 2 // CMM repeatable level
   → PCLN := 3 // CMM defined level
   → PCLN := 4 // CMM managed level
| ?~
   → PCLN := 5 // CMM optimizing level
)
CMM_PCLN := PCLN
¤CMMResultBL:= T
)

c. CMM Algorithm Dynamic Behaviors

CMMAlgorithm.DynamicBehaviors ≜
{
   {O:: {{SATKP[level]|N}, {RKP[level]|N, i|N}}})
   {O:: CMM_PCLN, ¤CMMResultBL})
)

RTPA is a contribution towards the development of an essentially small set of formal notations with reasonably expressive power for software system specification and refinement. RTPA has been developed as an expressive, easy-to-comprehend, and language-independent notation system; and a specification and refinement method for process system description and specification. RTPA can also be used to describe dynamic process deployment and process dispatch in executing a process model in a real environment.

6. Structure of this Special Volume on Process-Based Software Engineering

The theme of this special volume of Annals of Software Engineering is ‘empirical and innovative software engineering process systems’. This volume explores software engineering process models, systems, theories, and environments, with particular emphasis on innovative processes and empirical practices in engineering large-scale software development – and against a complex understanding of what the term software
engineering implies [Bryant 2001]. A structure of the subject areas on PBSE as modeled in this volume is as follows:

- **Domain of software engineering**
  - Foundations of software engineering (SE)
  - SE methodologies
  - SE infrastructures
  - SE organization
  - SE project management

- **Empirical process systems of SE**
  - SE process models
  - SE process standards
  - Best practices in SE processes
  - Enterprise SE process structures
  - Interrelationship of existing process models
  - Experience with SE processes

- **Innovative processes of SE**
  - New SE processes and practices
  - Incorporation of new technologies into SE processes
  - Integration of existing SE process systems
  - Internet-based processes in SE
  - Reuse processes in SE
  - Component-based processes in SE
  - Visualization processes of SE
  - Formal-method-based processes of SE
  - Domain knowledge representation in SE processes

With the above framework of PBSE, we present two sets of articles on “Empirical Process Systems of Software Engineering” and “Theories and Environments of Software Engineering Processes,” as shown in Table 5, in this special volume of Annals of Software Engineering. Fourteen papers have been selected from contributions of outstanding researchers and practitioners in software engineering among 44 submissions.

### Table 5
Structure of ASE Volume 14 on Process-Based Software Engineering

<table>
<thead>
<tr>
<th>Empirical Process Systems of Software Engineering</th>
<th>Theories and Environments of Software Engineering Processes</th>
</tr>
</thead>
</table>
We hope that readers of *Annals of Software Engineering* will benefit from the papers presented in this Special Volume on *Process-Based Software Engineering* and be able to share the latest research results and empirical best practices in software engineering with the authors.

7. Conclusions

Process technologies have been perceived as the latest trend in building the infrastructure of software engineering. A comprehensive review on process techniques for software engineering has been presented. The concept of process-based software engineering (PBSE) and the industrial practice of PBSE have been explored. A unified software engineering reference model (SEPRM) has been introduced, and roles of the process reference model have been analyzed. Empirical practices of PBSE in the software industry have been demonstrated. The domain of PBSE, encompassing software engineering process system establishment, assessment, improvement, and benchmarking, have been examined. Then a formal approach for software engineering process system description has been introduced.

An overview of the structure of PBSE and the configuration of selected papers of this special volume have been described. They present cutting-edge research results and best practices on “Empirical Process Systems of Software Engineering” and “Theories and Environments of Software Engineering Processes.” In this special volume, software engineering processes have been perceived as a coherent system rather than individual activities. Alternative approaches and techniques on the establishment, assessment, improvement and benchmarking of software engineering process systems have been investigated and presented from different perspectives.

Acknowledgments

This special volume on PBSE of *Annals of Software Engineering* is the result of the combined efforts and contributions of many people. We would like to thank all authors who submitted interesting papers to this volume. We acknowledge the guidance of the Editor-in-Chief Osman Balci, and Susan Lagerstrom-Fife and Sharon Palleschi at Kluwer Academic Publishers for their professional support. We thank Graham King for providing reference materials. The following people have participated in the reviewing processes:


We would like to thank the reviewers for their enormous effort in helping to maintain and uphold the quality of this special volume that covers a wide range of fundamentally significant issues on software engineering process models, systems, theories, and environments.

References


