

# SOFTWARE PROCESS REUSABILITY IN AN INDUSTRIAL SETTING

by

Craig R. Hollenbach

Masters Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

MASTERS OF COMPUTER SCIENCE

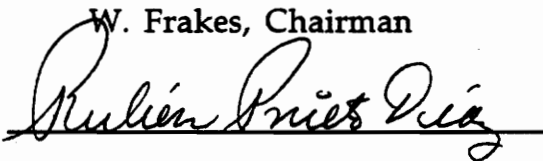
APPROVED:



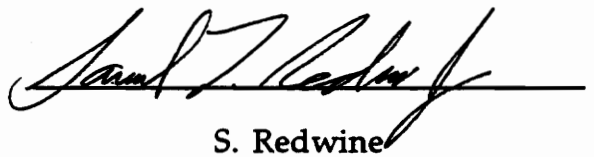
W. Frakes, Chairman



C. Fox



R. Prieto-Diaz



S. Redwine

October 1995

Blacksburg, Virginia

c.2

LD  
5655  
V855  
1995  
H654  
c.2

# **Software Process Reusability In An Industrial Setting**

by Craig R. Hollenbach

## **ABSTRACT**

Many software organizations are actively pursuing software process maturity. One of the cornerstones of software process improvement is the definition of processes in key process areas [Paulk 93a]. As development organizations define processes to increase their project's maturity, a method is needed to make the defined processes reusable by the organization and to tailor these reusable processes for reuse on other projects.

The thesis is that the development and implementation of a method to construct reusable corporate process definitions and to tailor them to specific projects will significantly increase the corporate level of process reuse.

Therefore, this thesis investigates how to pragmatically and systematically standardize and replicate project-specific processes in an industrial software setting. Process reusability and attributes of process reusability are discussed and a systematic and standardized method for process reuse is presented.

The utility of the methodology is demonstrated by its application to software process definition activities at PRC Inc., where the level of process reuse both before and after its application has been measured. Process reuse increased from 41% in 1994 to 55% in 1995. PRC also saw a 10 to 1 improvement in time to define a project-specific process.

## ACKNOWLEDGMENTS

I would like to first acknowledge Dr. Bill Frakes, my advisor. It is a tribute that my initial healthy respect for him has grown even stronger throughout my many months under his tutelage.

I also want to thank my colleagues at PRC, especially Jonathan Addelston, Cora Carmody, Caz Caswell, Jude Franklin, Judy Herndon, Mary Peterson, and Ralph Young for their cooperation, endorsement, and support.

My thanks also go to my parents who modeled for me hard work and attention to detail and provided unquestioned support.

I am most happy to finish for Aledra, my help-mate, and Haley and Nathan, my children, since they endured the absence of their husband and father at home and suffered along with me the long stretches of work.

Finally, I thank the great I AM, who in a very personal sense ceaselessly encouraged and supported me throughout this thesis process.

## TABLE OF CONTENTS

1. Introduction.....	1
1.1 Problem Definition.....	2
1.2 Current Situation.....	3
1.3 Analysis and Thesis.....	3
1.4 What is a Reusable Process?.....	4
1.5 Organization of the Remaining Thesis Chapters .....	6
2. Literature Review .....	8
2.1 Domain Engineering, Analysis, and Implementation.....	8
2.1.1 Definitions.....	9
2.1.2 Common Domain Analysis Process.....	11
2.1.3 Comparisons of Domain Analysis Methods.....	14
2.1.3.1 Bailin's Domain Analysis in KAPTUR.....	14
2.1.3.2 Lubars' Domain Analysis in IDeA.....	16
2.1.3.3 SEI's Feature-Oriented Domain Analysis (FODA) .....	16
2.1.3.4 SPC's Domain Analysis in the Synthesis Environment.....	17
2.1.3.5 Prieto-Diaz's Domain Analysis for Reusability.....	17
2.2 Software Processes .....	18
2.2.1 Software Process Context.....	18
2.2.2 Basic Process Artifacts .....	19
2.2.3 Process Engineering Actions .....	22
2.3 Process Modeling .....	23
2.3.1 Levels of Process Models.....	24
2.3.2 Essential Process Model Entities.....	24
2.3.3 Process Model Relationships and Behaviors.....	25
2.3.4 Perspectives in Process Representation .....	26
2.3.5 Process Modeling Paradigms.....	26
2.3.6 Issues in Process Modeling .....	29
2.4 Software Reuse.....	30
2.5 Software Process Reuse Techniques .....	32

3. Process Reusability.....36

3.1 Definitions.....36

3.2 Process Reusability.....37

3.3 Process Domains .....37

3.4 Methods of Defining Process Variability .....41

4. Software Process Reuse Methodology .....44

4.1 Introduction.....45

4.2 Process Notation at PRC.....46

4.3 Context for Process Definition and Reuse.....52

4.4 Process Definition Process.....55

4.4.1 Customers.....55

4.4.2 Process Interfaces.....56

4.4.3 Process Activities .....58

4.5 Process Tailoring Methodology .....63

4.6 The Storage and Retrieval of Reusable Processes.....65

5. Implementations of the Software Process Reuse Methodologies.....71

5.1 Definition of Reusable Configuration Management Processes.....71

5.2 Tailoring of a Reusable Peer Review Process .....82

6. Case Study.....92

7. Conclusions.....103

7.1 Conclusions.....103

7.2 Future Research .....103

References.....105

Vita.....110

**LIST OF TABLES**

Table 1-1. Examples of the 3C's model .....5

Table 2-1. Comparison of Domain Analysis Methods.....15

Table 2-2. Comparison of Process Modeling Paradigms.....27

Table 2-3. The 3C's Reference Model .....30

Table 6-1. Process Reuse in 1994-1995 by Business Unit .....95

## LIST OF FIGURES

Figure 1-1. A Process Reusability Example.....	6
Figure 2-1. Two Domain Analysis Viewpoints.....	10
Figure 2-2. Structure of Process Concepts .....	18
Figure 2-3. Process Definition Artifacts and Actions.....	20
Figure 3-1. The Three Categories of KPAs.....	39
Figure 3-2. A Configuration Management Process Domain .....	41
Figure 3-3. An Example of Process Parameterization .....	42
Figure 3-4. An Example of Process Abstraction.....	43
Figure 4-1. Process Notation at PRC - Part 1 of 3 .....	48
Figure 4-1. Process Notation at PRC - Part 2 of 3 .....	49
Figure 4-1. Process Notation at PRC - Part 3 of 3 .....	50
Figure 4-2. Formats for Variations.....	51
Figure 4-3. Process Definition Life Cycle.....	54
Figure 4-4. Process Tailoring Methodology .....	66
Figure 4-5. PAL Administration.....	70
Figure 5-1. Configuration Identification Process Definitions.....	77
Figure 5-2. CM200 Configuration Identification Process - Part 1 .....	78
Figure 5-2. CM200 Configuration Identification Process - Part 2 .....	79
Figure 5-2. CM200 Configuration Identification Process - Part 3 .....	80
Figure 5-2. CM200 Configuration Identification Process - Part 4 .....	81
Figure 5-3. Corporate Peer Review Process - Part 1.....	84
Figure 5-3. Corporate Peer Review Process - Part 2.....	85
Figure 5-3. Corporate Peer Review Process - Part 3.....	86
Figure 5-3. Corporate Peer Review Process - Part 4.....	87
Figure 5-4. Tailored Peer Review Process - Part 1 .....	88
Figure 5-4. Tailored Peer Review Process - Part 2 .....	89
Figure 5-4. Tailored Peer Review Process - Part 3 .....	90
Figure 5-4. Tailored Peer Review Process - Part 4 .....	91
Figure 6-1. Defined Processes by Business Unit by Year .....	94
Figure 6-2. Process Definitions in Level 2 Key Process Areas by Reuse Type.....	98



Figure 6-3. Process Definitions in Level 3 Key Process Areas by Reuse  
Type.....99

Figure 6-4. Percentage of Process Reuse by Business Unit .....102

## **1. Introduction**

Many software organizations are actively pursuing software process maturity [Humphrey 89]. Often they use measures like the Capability Maturity Model (CMM) for Software [Paulk 93a] to structure their process improvement initiatives. While the CMM calls for organizational process definitions for use on software development projects, there is very little help to show organizations how to leverage their current process knowledge from projects into process definitions that can be reused across the organization, i.e., to make the process definitions reusable. Furthermore, even though some organizations discuss tailoring the CMM or organizational process definitions, they offer no method to do the actual tailoring. [Ginsberg 95] [McDaniel 95]

At the 6th International Software Process Workshop, [Kumagai 91] summarized a consensus viewpoint by saying that "process descriptions MUST be reusable. It should be possible to describe processes or process fragments in general terms. It should be possible to instantiate general descriptions for specific cases." A search of the literature shows that these goals have not yet been realized. Although there are a few instances of replicating processes within organizations [Fagan 86] [Gale 90], there are even fewer documented instances of a methodical process reuse program applied across an entire organization. Only a very small number of studies have addressed the actual construction of reusable software processes, leading one to believe that systematic process reuse is either trivial or unrealized. The author believes the latter to be the case.

The thesis is that the development and implementation of a method to construct reusable corporate process definitions and to tailor them to specific projects will significantly increase the corporate level of process reuse.

Some benefits of reusable processes are an increased ability to transfer process knowledge between projects in a time- and cost-effective

manner, to reduce training costs, to plan project activities, and to adjust project performance based on process metrics, in a continuously improving process-oriented environment.

Therefore, this thesis investigates how to pragmatically and systematically standardize and replicate project-specific processes in an industrial software setting. Process reusability is discussed and a systematic and standardized method for process reuse is presented.

The utility of the methodology is demonstrated by its application to software process definition activities at PRC Inc., where the level of process reuse both before and after its application has been measured. Initial results show that process reuse increased from 41% in 1994 to 55% in 1995. PRC also saw a 10 to 1 improvement in time to define a project-specific process.

## **1.1 Problem Definition**

In 1991 PRC began to work with individual projects to increase their software maturity. The company centralized their software process improvement efforts into one group of full-time staff in the company's Technology Center. The group used the Capability Maturity Model (CMM) for Software [Paulk 93a] to guide its process improvement efforts. The model measures organizational maturity according to five levels. Organizations begin at level 1 and improve by becoming proficient in key process areas (KPAs) defined for the next level. Each KPA defines goals and commitments, abilities, activities, verifications, and measurements to meet the goals. Activities describe repeatable actions that often call for documented procedures. To comply with these CMM-required activities, organizations define software processes.

Based in part upon the experience of a pilot project, the group of full-time staff developed approximately 300 corporate process descriptions in the level 2 key process areas, plus the peer review key process area. The group organized these process descriptions into a set of process manuals. Although the pilot project used these corporate process descriptions, there was little interest among other projects in software process improvement and process adoption.

In 1993, PRC formed the "Phoenix" team. Composed of eleven of PRC's most important software projects and assisted by the Technology Center group, PRC tasked the team with improving the maturity of the projects using both the company's quality improvement techniques [Arthur 93] and CMM-based methods and principles. The Phoenix projects defined their own processes, often without reference to the previously defined corporate processes. To maximize process maturity, they tried to replicate "best of breed" (BoB) processes from the most mature projects to more immature projects. This failed almost totally. The team determined that they needed a more methodical approach to process reuse, one that abstracted away project-specific details so that other projects could reuse the processes.

## **1.2 Current Situation**

To examine the 1994 level of process reuse more closely, the PRC SEPG surveyed representatives from the Phoenix team. This survey showed that of the possible coverage across the thirteen level 2 and 3 KPAs, only 27% of the KPAs had instances of process definition. Eleven percent of the KPAs had processes that were derived from corporate ones in some unspecified way and only two percent of the KPAs had instances of processes that were derived from a "BoB" process.

## **1.3 Analysis and Thesis**

From this survey, several root causes were identified. First, PRC did not have a consistent approach to developing processes that could be reusable within PRC. Second, PRC did not have a consistent approach to tailoring the corporate processes to the specific needs of business units or projects. Third, adequate training based on the corporate processes did not exist. Finally, the PRC process community needed a better mechanism to access "soft copy" of these process assets.

From these root causes, the thesis for this study was constructed:

The development and implementation of a method to construct reusable corporate process definitions and to tailor them to

specific projects will significantly increase the corporate level of process reuse.

#### 1.4 What is a Reusable Process?

[Over 94] defines a process as "a logical organization of people, procedures, and technology into work activities designed to transform information, materials, and energy into a specified result." We define process reuse as the usage of one process description in the creation of another process description. It is not multiple executions of the same process on a given project.

What makes a process reusable? To answer that question it is important to examine what makes software reusable. Significantly, the research community has not definitely answered that question even though it has researched it for over ten years; there is no foolproof external indicator that can determine the reusability of a software module [Frakes 96]. The best answer to date is simply that the software module is reusable if it is reused. The same can be said for software processes; if reused, they are reusable.

What is the difference between porting and reusing processes? Porting refers to moving a whole process system to a new environment or platform; reuse refers to using a process in a different system [Frakes 95].

The 3C's model of reuse design provides a framework that has been found effective in the design of reusable assets [Latour 91]. The model indicates three aspects of a reusable component -- its *concept*, its *content*, and its *context*. The concept specifies the abstract semantics of the component, the content specifies its implementation, and the context specifies the environment necessary to use the component.

Table 1-1 shows an example of the 3C's model applied to a software component and to the reusable process notation described in Chapter 4. For a software component, the concept might correspond to an abstract data type (ADT), whose implementation might be a C module using linked lists. This

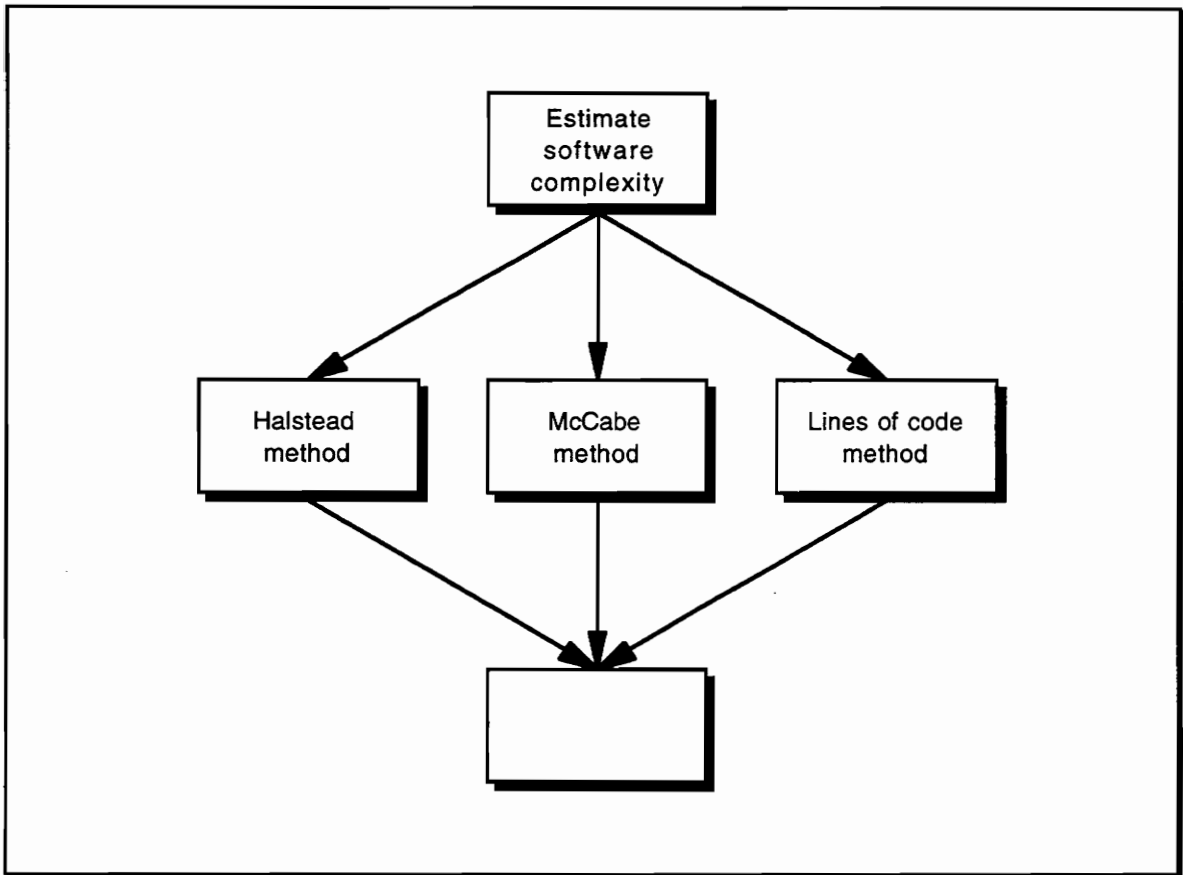
component's context might require a Sun workstation running UNIX, and an ANSI C compiler.

**Table 1-1. Examples of the 3C's model**

<b>3 C's Aspect</b>	<b>Code</b>	<b>Process</b>
<b>Concept</b>	Abstract Data Type (ADT)	Informal specification of: <ul style="list-style-type: none"><li>• General information</li><li>• Customer description</li><li>• Interface description</li></ul>
<b>Content</b>	Implemented in C with link lists	Procedural description (graphic & text)
<b>Context</b>	Operating system = UNIX, hardware = Sun, compiler = ANSI C	Contextual description

Analogously, a reusable process will have for its concept an informal specification of: the general information, the customer description, and the interface description. The content will include the procedural description using a textual and graphical representation. The reusable process context will be defined in the contextual description.

Generally speaking, a process is more reusable when it can be used in various situations without changing its concept. The concept in this case is the portion of the process description that defines the customer and associated requirements, the interfaces, and other general information. Figure 1-1 shows an example of a reusable "estimate software complexity" process. In the course of the process, a complexity measure is needed; there are three methods used to measure complexity: a Halstead measure, a McCabe measure, and a "lines of code" measure. The use of any of these methods should not affect the concept of the process; indeed, the degree to which it affects the concept is the degree to which it is less re-usable.



**Figure 1-1. A Process Reusability Example**

## **1.5 Organization of the Remaining Thesis Chapters**

This chapter provides an introduction to the problem to be studied and its associated thesis. It gives a summary description of a reusable process and describes how the rest of the document is organized.

Chapter Two surveys the literature on domain analysis, software process definition and modeling, and software and process reuse. Principles from the literature are used for material in Chapters Three and Four.

Chapter Three defines process and process reuse, discusses what process reuse is, defines process domains, and describes methods of process variability.

Chapter Four describes the PRC process reuse methodology. First, a process notation used at PRC is described. Within a process framework, the context

for process reuse is discussed and the two parts of the methodology are presented. The first part, the process definition methodology, describes how to create a reusable process definition. The second part, the process tailoring methodology, describes how to tailor a reusable process to the specific characteristics of a project. Finally, the method for retrieving processes and process assets at PRC is provided.

Chapter Five describes an implementation of the process definition methodology and the process tailoring methodology at PRC. The process definition methodology is applied to the configuration management domain and the process tailoring methodology is applied to the tailoring of a reusable peer review process.

Chapter Six describes a case study within PRC and reviews the results.

Chapter Seven provides conclusions and areas for further study.



## **2. Literature Review**

This section surveys the literature on software processes and reuse and categorizes it into several subareas that are described below:

- 2.1 Domain Analysis and Implementation - This section defines domain terminology, describes a common domain analysis process, and surveys a set of specific domain analysis techniques. This section lays the foundation for Section 3.3, Process Domains. The common domain analysis process is used as the basis for the PRC Process Definition Methodology as described in Section 4.4.
- 2.2 Software Process Definition - This section defines basic software process terms and concepts that orient the reader to Section 4.3, the Context for the PRC Process Definition Methodology.
- 2.3 Process Modeling - This section describes the status of process modeling and surveys process modeling paradigms. It also provides a context for PRC's approach to modeling and to process notation, as described in Section 4.2.
- 2.4 Software Reuse - This section presents the 3 C's reference model that provides the reuse foundation for PRC's process notation and process definition methodology.
- 2.5 Software Process Reuse - This section surveys the few instances of research on process reuse.

### **2.1 Domain Engineering, Analysis, and Implementation**

A domain is a set of related systems. Domain Engineering is the "process of analyzing a domain and creating reusable assets in the domain." [Frakes 93] Domain engineering has two parts, domain analysis and domain implementation. Domain analysis examines a set of application systems within a domain to discover and model their commonalties and variabilities.

[Prieto-Diaz 87] Domain implementation constructs reusable assets and new systems within the domain based upon the results of the domain analysis.

This section describes fundamentals of domain engineering, with special emphasis on the process of domain analysis. It begins with a set of definitions of domain analysis and then describes a domain analysis methodology that is common to most domain analysis approaches. Finally, five representative approaches are examined.

### **2.1.1 Definitions**

To understand the various domain analysis methods and the commonality between them, four terms are defined: problem domain, domain analysis, domain model, and task specification. Two distinct viewpoints of these domain analysis definitions are briefly compared: problem-centric and systems-centric analysis.

Problem Domain - [Arango 94] defines a problem domain as a set of real-world information that has two qualities:

- a) "deep or comprehensive relationships among the items of information are suspected or postulated with respect to some class of problems, and
- b) the problems are perceived as significant by the members of the community."

Domain Analysis, Domain Modeling - Activities whose purpose is to impose a coherent organization upon domain data in order to reduce complexity and promote better understanding of the domain.

Task Specification - A rigorous description of a task that describes the structure and associated activities of the task, including goals, operators or functions, methods, implementation details, and rationale information.

Domain Model - A formal system that is the output of domain analysis and used to obtain and derive information about a problem domain.

[Arango 94] describes two viewpoints of these definitions: a problem-centric viewpoint or a systems-centric viewpoint. As shown in Figure 2-1, a problem-centric viewpoint sees the problem domain as a set of problems. Domain analysis is then a way to devise a theory of problems that predicts, explains, and derives useful facts about the problem set. The output of problem-centric domain analysis is a formal system of terms, relationships

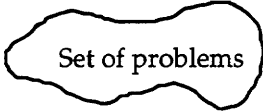

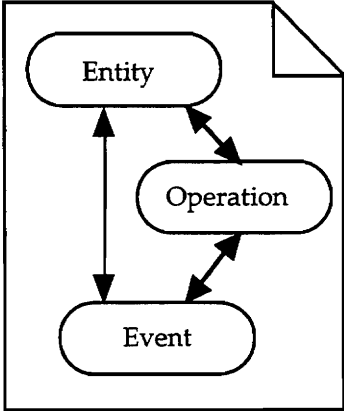
	Domain Viewpoint	
	Problem-Centric	Systems-Centric
Problem Domain		
Domain Analysis	<div>An activity to reduce the complexity of a domain by imposing a coherent organization</div>	
Domain Model	<div>Formal system of: <ul style="list-style-type: none"> <li>- Terms</li> <li>- Relationships between terms</li> <li>- Rules to compose terms into expressions</li> <li>- Rules to reason about terms</li> <li>- Rules to map problem domain to model and vice versa</li> </ul> </div>	
Purpose of Domain Model	Theory of problems used to predict, explain, and derive facts	Taxonomy of system components for reuse

Figure 2-1. Two Domain Analysis Viewpoints

between terms, and rules to 1) compose terms into expressions, 2) reason about terms, and 3) map the problem domain to the model and the model to the problem domain.

A systems-centric viewpoint sees the problem domain as a set of applications from which to derive a taxonomy of system components, used primarily for reuse. A domain model in this context defines entities, operations, and events, and the relationships between them in order to organize the domain into a set of common architectures and components. Systems-centric domain analysis can be dynamic, adding components to the taxonomy for reuse. This thesis and the case study at PRC take a systems-centric viewpoint that provides a framework for reusable processes that the organization can update as it continuously learns and improves.

### **2.1.2 Common Domain Analysis Process**

[Arango 94] defines a common domain analysis process derived from his study of domain analysis approaches. It is based upon an "abstract and classify" paradigm. The five generic activities of the common process are:

- 1) Domain characterization and project planning - This step analyzes the feasibility of the domain analysis from business and technical points of view. If found feasible, data about the domain is identified and the domain analysis is planned. The step is composed of five substeps. These substeps are not strictly sequential; information from one substep may require that an earlier substep be revisited.
  - 1.1) Select domain - Traditional business and risk analysis techniques determine the feasibility of the domain analysis. Organizations use these techniques to decide whether the project is right for the company at this time, whether they have selected the right domain, whether there is sufficient return on investment, whether the right expertise is available within the company, and whether the domain is mature enough for analysis.

- 1.2) Describe domain - This activity defines the scope and contents of the domain and sets boundaries on the domain analysis effort.
  - 1.3) Identify relevant data - By identifying the data, the domain analyst can decide if enough relevant data exists, what the sources of information are, and whether there is sufficient access to the data.
  - 1.4) Create inventory of data - Inventorying the data is an optional activity that prepares for the subsequent collection of the data itself.
  - 1.5) Plan the project
- 2) Data collection - This activity collects raw data that the domain analyst can later filter, clarify, abstract, and organize. The analyst uses several approaches that described below. Each approach delivers a different type of information. Their associated information acquisition techniques are useful, but not exclusive, to domain analysis.
    - 2.1) Recover abstractions - The domain analyst retrieves detailed information of existing applications regarding relevant components (e.g., user interface, architectures), behavior, and original system designs and rationales. The analyst may use reverse engineering to recover the abstractions.
    - 2.2) Review literature
    - 2.3) Elicit knowledge from experts - Experts can identify underlying principles, design rationales, and system pitfalls to avoid. They can also validate information from other sources.
    - 2.4) Develop scenarios - Scenarios explain how the user community and other interfacing systems typically use the systems.
  - 3) Data analysis - In this activity, the domain analyst verifies the data for correctness, consistency, and completeness, and describes the reusable modules using a six step process, listed below. Step one identifies

important domain events, entities, and operations, and the relationships between them. Step two organizes and modularizes relevant data using either object-oriented or functional and data decomposition. Steps three through six capture recommendations. The specific domain analysis techniques employed are not unique to domain analysis; rather, the unique nature of domain analysis stems from the fact that the focus of domain analysis is a set of applications whereas the focus of other analyses is a single application.

- 3.1) Identify entities, events, operations, and relationships - The domain analyst describes the domain of systems in terms of major data units, functions performed on these, outside events that affect them, and relationships between and within all three.
  - 3.2) Modularize information - The domain analyst modularizes the information by either function and data decomposition or object-oriented analysis, and identifies and records design decisions.
  - 3.3) Analyze similarities - The analyst identifies similarities in order to allow consolidation of application commonalties.
  - 3.4) Analyze variations - The analysis suggests ways to enumerate, parameterize, or encapsulate the variations.
  - 3.5) Analyze combinations - Combinations suggest structural or behavioral schemas and/or architectures.
  - 3.6) Analyze tradeoffs - Tradeoffs suggest different ways to decompose architectures to satisfy incompatible requirement sets.
- 4) Classify - Classification is the primary modeling activity in domain analysis. It captures and explicitly states the structure of information for classes of applications. Various methods are employed to classify the information on different attributes: asset type and feature (Bailin KAPTUR), features (FODA), features and decisions (SPC Synthesis), and facets (Prieto-Diaz).

- 4.1) Cluster descriptions - Information retrieval clustering algorithms are sometimes used to group the descriptions.
  - 4.2) Abstract descriptions - A generalization of the all the descriptions within each group is composed, highlighting the most relevant common features.
  - 4.3) Classify descriptions - When new descriptions are available, they are assigned to a cluster or the clusters are reorganized to include the new descriptions.
  - 4.4) Generalize descriptions - Hierarchies are formed to meaningfully relate the abstract descriptions together.
  - 4.5) Construct vocabulary - A domain vocabulary, or language, is optionally constructed of relevant domain jargon and formalized to aid in the classification process.
- 5) Evaluation of domain model - Validation criteria are not included in the individual methods and therefore are not part of Arango's common process. Model validation is discussed as an important step in the specific methodologies; none, however, have procedures for validating the domain model.

### **2.1.3 Comparisons of Domain Analysis Methods**

This section samples five representative domain analysis methods. As shown in Table 2-1, each method has a particular focus and unique strengths. The domain analysis process used by each method is a variation on the common process described in Section 2.1.2.

#### **2.1.3.1 Bailin's Domain Analysis in KAPTUR**

KAPTUR is both a tool environment and a domain analysis process. [Bailin 91] It views reusable domain assets in terms of their features, which represent the constituent systems, objects, and functions within the domain. The term feature is used in a different sense than it is in the FODA, Synthesis,

**Table 2-1. Comparison of Domain Analysis Methods**

<b>Method</b>	<b>Author</b>	<b>Focus</b>	<b>Strengths</b>
KAPTUR	Bailin	Describes domain model graphically and textually based on systems, objects, and functions	<ul style="list-style-type: none"> <li>• Validation of domain model</li> <li>• Capture of design rationales</li> </ul>
IDeA	Lubars	SW design construction by building abstract design schemas	<ul style="list-style-type: none"> <li>• Cyclically analyzes similar problems in same and other domains</li> </ul>
FODA	SEI	Uses existing SW engineering modeling techniques	<ul style="list-style-type: none"> <li>• Documented examples</li> <li>• Validation steps</li> <li>• Well defined process</li> <li>• Includes functional, operational, presentation components</li> </ul>
Synthesis	SPC	Integrates domain analysis with 2167A life cycle model	<ul style="list-style-type: none"> <li>• Excellent documentation</li> <li>• Includes lessons learned</li> <li>• Repository of domain knowledge</li> </ul>
Domain Analysis for Reusability	Prieto-Diaz	Populates SW libraries of reusable components using faceted classification schemes & SW engineering modeling techniques	<ul style="list-style-type: none"> <li>• Domain dictionary, language, &amp; model</li> <li>• Combines IR &amp; SW engineering techniques, both top-down &amp; bottom-up</li> </ul>

or Organon techniques, where features refer only to the functional characteristics of the system. KAPTUR provides assistance to both the producers of KAPTUR features, i.e., the domain analysts/implementors, and the consumers of KAPTUR features, i.e., the component reusers. KAPTUR provides assistance to the producers by 1) assisting in the formation of architectural drawings and 2) entering and classifying textual information, all



at differing levels of abstraction. The domain analysis process does not vary significantly from the common process defined in 2.1.2. Two strengths of the KAPTUR process are in the validation of the domain model and the capture of design rationales.

#### **2.1.3.2      Lubars' Domain Analysis in IDeA**

IDeA is a design environment that assists in software design construction, and supports reuse of abstract designs represented as semi-formal design schemas. [Lubars 91] Lubars uses domain analysis to populate the design reuse libraries with schemas for new domains. While the IDeA process is similar to the common process, it also specifies steps for cyclically analyzing results from individual problems with other similar problems in the same domain and then in other domains. Lubar's approach is therefore closer to the idea of an evolutionary domain analysis process than the other approaches; however, it does fall short of specifying how to determine similarities and under what conditions to stop the cyclical process.

#### **2.1.3.3      SEI's Feature-Oriented Domain Analysis (FODA)**

The feature-oriented domain analysis (FODA) approach was developed by the Software Engineering Institute (SEI) to discover a set of common features within related software systems that could be represented in a usable format with maps to specific instances of those features. [Kang 90] Based upon sound modeling techniques taken from the software engineering arena like hierarchical decomposition and entity-relationship models, the FODA approach comes close to being a union of other approaches. This approach has several strengths. First, the documentation contains detailed examples of the FODA approach; secondly, the approach contains the unique feature of having several validation steps; thirdly, the approach clearly defines the goals, inputs, outputs, and internal steps of each activity in the process; and finally, FODA decomposes the notion of a feature into functional, operational, and presentation components. Features are aggregated into hierarchies using a CONSISTS-OF relationship.

#### **2.1.3.4 SPC's Domain Analysis in the Synthesis Environment**

The Synthesis project at the Software Productivity Consortium (SPC) is designed to provide its member companies with a methodology that is integrated into the DoD-mandated 2167A standard. [Jaworski 90] The methodology uses domain analysis during the system and software requirements definition phases to construct a reusable set of system components, drawn from a larger family of similar systems. Like FODA, it has excellent documentation and draws on results from several analyses. The Synthesis environment uses a repository to store all domain knowledge which in turn is the basis for the domain modeling process. The SPC has chosen to use a variant of object-oriented analysis to formalize domain requirements.

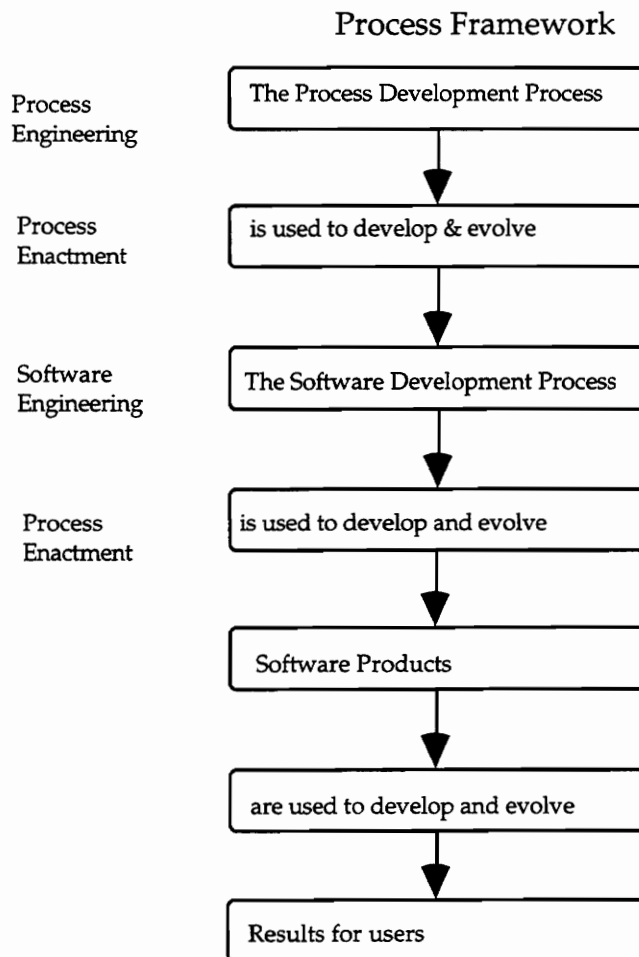
#### **2.1.3.5 Prieto-Diaz's Domain Analysis for Reusability**

The purpose of Prieto-Diaz' domain analysis approach is to populate libraries of reusable software components, organizing them using a faceted classification scheme. [Prieto-Diaz 87] He draws heavily on information retrieval. [Prieto-Diaz 91a] Prieto-Diaz' approach uses dataflow diagrams to explicitly define activities, inputs/outputs, relative sequence, & opportunities for parallelism. His approach has three parts: preparation, domain analysis, and work product generation. The domain analysis activity produces three outputs: a taxonomic classification of definitions for domain objects, functions, and relationships; a domain language; and a domain model using a faceted classification scheme based on vocabulary in the domain language. These structures provide reasoning capabilities based on the relationships IS-A( $x,y$ ), PART-OF( $x,y$ ), and INSTANCE-OF( $x,y$ ) and their subsequent retrieval through faceted descriptors. Prieto-Diaz has extended this approach to couple a top-down analysis using standard structured analysis techniques with existing bottom-up faceted classification techniques.

## 2.2 Software Processes

### 2.2.1 Software Process Context

Since software process terms and their meanings are often dependent upon their context, [Feiler 92] presents a context for software development process architectures and related process definitions, as shown in Figure 2-2.



**Figure 2-2. Structure of Process Concepts**

The scope of this thesis is the first function, the process development process, which is used to develop and evolve a software development process, which

in turn is enacted on a specific software project. A subpart of this function is the development of reusable process definitions.

[Feiler 92] states that to be widely valued and used, process definitions must make higher-quality software easier and more economical to produce. These process definitions must be useful to practitioners and reasonably economical to produce.

Figure 2-3, taken from [Feiler 92], illustrates the basic process artifacts and actions involved in the process development function. The next two sections describe these artifacts and actions respectively.

### **2.2.2 Basic Process Artifacts**

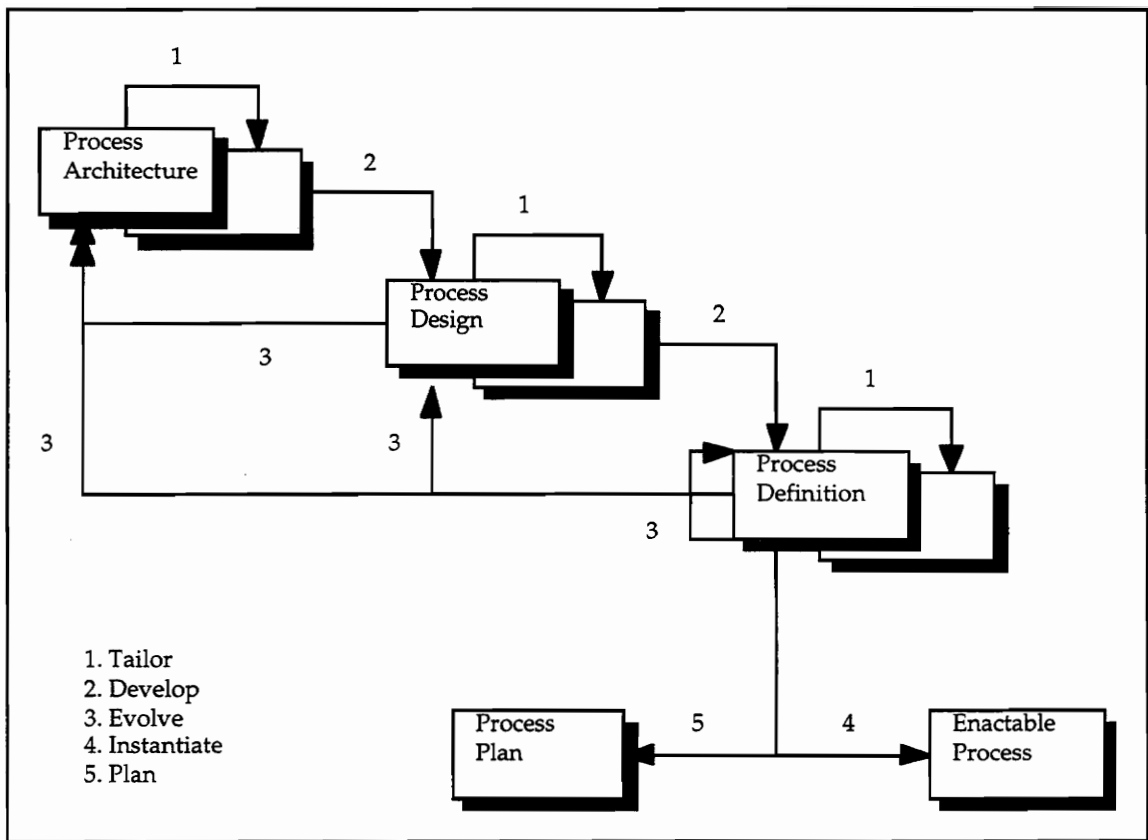
[Feiler 92] defines six basic process artifacts from the "develop and evolve a software development process" function. They are:

- Process Architecture
- Process Design
- Process Definition
- Process Plan
- Enactable Process
- Process Model

**Process architecture** - a conceptual framework for consistently incorporating, relating, and tailoring process elements into enactable processes.

Process architectures provide space for process designs. They are useful in specifying how a process must relate to other processes, either existing or future. Indeed, an essential characteristic of a process architecture is its ability to indicate whether a given process is compatible with an architecture or not.

Process architectures are sets of designs with their peculiar characteristics and rules. They define standards for the structures and interfaces within the architecture. For instance, one can build an architecture using the ETVX



**Figure 2-3. Process Definition Artifacts and Actions**

(Entrance criteria, Task, Validation, Exit criteria) process model [IBM], object-oriented models, etc.

[Redwine 93] points out other uses for process architectures, including enumerating components; describing relationships, evolution paths, and reuse variations; providing guidance on process selection, adaptation, and composition; and providing compatibility across various implementations.

[Radice 85] states that an orderly evolution of process architectures is desired and planned. As a baseline, the initial process architecture must: 1) ensure repeatable and simple paradigms for software development at all levels, 2) contain a self-improvement process based on statistical quality control, 3) require a validation mechanism to ensure the quality of the produced products, 4) be based on the best, proven, available knowledge within the

software industry, 5) address the entire software life cycle, and 6) be independent of tools.

**Process design** - an embodiment of a process architecture that establishes the architectural options and parameters, the existing elements to be reused, the structure and behavior of new elements, and the relationships among these elements.

Process designs may apply to a specific project, a single organization, or a group of organizations. They are produced to meet specific goals. A process design includes 1) process definition and instantiation standards and interfaces, 2) overall process structure, and 3) functions and relationships of the process elements.

A process design may include reusable process definitions and partially/fully populated process elements. It specifies selection choices to be made during process development.

Process designs differ from process architectures in that process designs add specificity to an existing process architecture in terms of reusable processes, rules for new processes, interface and instantiation standards, etc. An example of a process design would be a requirements management (RM) process design which would include a set of reusable RM processes, rules for adding new RM processes, and descriptions of how RM processes interface.

**Process definition** - an implementation of a process design in the form of a partially ordered set of process steps that is enactable.

[Feiler 92] states that each process step may be further decomposed, and that process steps may be enacted concurrently. [Feiler 92] adds that a process definition is complete or fit for enaction when its levels of abstraction are refined fully.

**Process plan** - a specification of the resources necessary for the enactment of a process definition, the relationships of these resources to process steps, the products produced by these steps, and any constraints on enactment or

resources. Process plans guide the instantiation and use of processes while project plans guide the design, development, evolution, and tailoring of processes (or products).

Resources for a process plan include humans, computers, time, and budgets. Process plans show relationships of resources to process steps in order to meet process objectives. A project plan contains work packages which include: 1) a process definition at some level of abstraction, and 2) one or more process plans.

**Enactable process** - an instance of a process definition that includes all the elements required for enactment.

An enactable process consists of 1) process definition, 2) required process inputs, 3) assigned enactment agents and resources, 4) an initial enactment state, 5) an initiation agent, and 6) continuation and termination capabilities. A process that lacks any of these six parts is not enactable [Feiler 92].

**Process model** - an abstract representation of a process architecture, design, or definition.

[Feiler 92] states that process models can be used where the use of a full complete process is undesirable or impractical. They can be analyzed, validated, or used to simulate processes. They also assist during process analysis, aiding in process understanding or predicting process behavior.

### 2.2.3 Process Engineering Actions

There are also actions that are performed on the process definition artifacts. [Feiler 92] They are:

**Tailor** - the act of adapting process architectures, designs, or definitions to the unique needs and characteristics of an enactment environment.

**Develop** - the act of creating enactable processes. Development transforms an initial process architecture into a process design that implements the architecture and then into a set of enactable process definitions. Each

transformation step may include planning, architecture, design, and validation tasks.

**Evolve** - the act of changing existing process architectures, designs, and definitions to increase their productivity, quality, and ability to meet new needs. Evolution of process assets is performed in the spirit of continuous improvement.

**Instantiate** - the act of creating enactable processes from process definitions and process plans by applying the planned personnel and resources to the enactment of the process definition using existing inputs and an initial enactment state.

**Plan** - the act of developing a process plan for the enactment of a process definition. Process planning includes specifying the required personnel and resources, the relationship of this process to other processes, the specific output of the process, and any constraints.

## **2.3 Process Modeling**

[Curtis 92] defines a process model as "an abstract description of an actual or proposed process that represents selected process elements that are considered important to the purpose of the model and can be enacted by a human or machine."

The objective of process modeling is to assist in the study and improvement of the organization's methods and products by defining the organization's processes and relating them to each other, to the people who enact the processes, and to the work products that are produced.

[Curtis 92] defines five uses of process models. Process models:

1. Facilitate human understanding and communication
2. Support process improvement
3. Support process management



4. Automate process guidance
5. Automate execution support

### **2.3.1 Levels of Process Models**

[Humphrey 89] describes three levels of software process models: the U, or Universal, process model; the W, or Worldly, process model; and the A, or Atomic, process model. The U model describes the software process in its most global sense. It describes the life cycle of software development and its constituent phases or stages and is useful for initial software planning. The W model describes the sequence of tasks and "who does what when." It is most directly applicable to the day to day work of software practitioners. The A model describes all the information necessary to fully automate the process and can be enormously detailed; it is of interest primarily to the process programmer.

To PRC, the W model is most important. The U model is useful for initial project planning but by itself does not provide enough detail to be useful during project execution. The A model on the other hand is much too detailed; at this stage of PRC's process maturity, process programming is not viewed as a viable alternative. Therefore, efforts are concentrated on making human-enactable processes. This perspective determines a very pragmatic approach to process definition and modeling. Much of the process programming research surveyed below is of interest to PRC but does not immediately affect PRC's software process improvement priorities.

### **2.3.2 Essential Process Model Entities**

Process models describe processes using at least three essential entities [Armitage 93]:

1. Agent - an actor (human or machine) who performs a process element [Curtis 92]. [Curtis 92] further defines an agent's roles as a coherent set of process elements to be assigned to an agent as a unit of functional

responsibility. A single agent can perform multiple roles, and a single role can be performed by multiple agents.

2. Artifact - a product created or modified by the enactment of a process element [Curtis 92].
3. Activity - the work performed during the process [Armitage 93].

[Curtis 92] defines a process as "one or more agents acting in defined roles to enact the process elements that collectively accomplish the goals for which the process was designed." A process usually manipulates (transforms) an artifact or "coordinates dependencies with other agents involved in the same or a related process." Processes can be 1) planned as a part of a larger process, 2) assigned a to role, 3) allocated resources, and 4) monitored.

### **2.3.3 Process Model Relationships and Behaviors**

[Armitage 93] describes two aspects of process models that are orthogonal to process entities: relationships and behaviors.

Relationships occur both within and among entity classes. Examples of relationships within entity classes are the relationship between activities and sub-activities, or the relationship between an artifact and the artifact from which it is derived. Examples of relationships between entity classes are those present when activities are performed by agents, activities use and produce artifacts, agents own artifacts, and agents perform activities to produce artifacts. Decomposition is a common relationship where an entity is divided into finer grained units.

Behavior of a process entity or relationship is dynamic in nature. Indeed, a process can be viewed as the set of entity and relationship behaviors over time. Examples of behavioral information are when and under what circumstances an activity can begin or complete; when and under what circumstances an artifact is produced or acquired, or it's state is changed; and when and under what circumstances an agent can begin or end work, or make decisions. Behavior is represented in many ways. Some examples of

behavioral representation are: entry/exit criteria, states and transitions, events and triggers, pre/post conditions, decision tables, and rules.

#### **2.3.4 Perspectives in Process Representation**

Process models capture information about processes that is desired by the user community, usually answering the traditional questions of "what, who, when, where, how, and why." Process modeling languages and representations provide one or more perspectives to these questions.

Four common process modeling perspectives are:

- 1) Functional - stresses "what" process elements and information entities are involved.
- 2) Behavioral - stresses "when" process elements are executed and "how" they are executed, including feedback, iteration, complex decision-making conditions, and entry/exit criteria.
- 3) Organizational - stresses "where" and "by whom" process elements are performed. Process participants are often termed agents. The organizational view includes physical transfer and storage mediums.
- 4) Informational - stresses details of the informational "what"s, including information structure and internal relationships.

The hypothesis that all 4 perspectives are needed is yet untested [Curtis 92]. [Curtis 92]'s view of the state of the practice is that modeling is performed, but not much rigor is practiced in the modeling techniques. As a result, it is difficult to analyze the properties of these perspectives.

#### **2.3.5 Process Modeling Paradigms**

At this time, process modeling is fluid and includes several techniques. These techniques can be loosely categorized into the five paradigms [Curtis 92] that are shown in Table 2-2 and described below.

**Table 2-2. Comparison of Process Modeling Paradigms**

Paradigm	Derivation	Feature	Example
Programming	Software Engineering	Software development tools and techniques	APPL/A by Osterweil
Functional	Mathematics	Set of math functions define process and relationships between inputs and outputs	HFSP by Katayama
Plan	Artificial Intelligence	Rules define actions based on satisfaction of pre-conditions	GRAPPLE by Huff and Lesser
Petri-Net	Simulation	Describes interaction between roles within a process	ProcessWeaver by Cap Gemini
Quantitative	System Dynamics	Applies feedback and control techniques to social and industrial situations	Abdel-Hamid and Madnick

*Programming Models:* Programming models use software programming tools and techniques to model processes. An example is Ada Process Programming Language based on Aspen (APPL/A) developed by Osterweil and his colleagues. [Sutton 90] This model enables representation of programmable, persistent relations using Aspen, a software engineering data model that captures relations between software objects in an entity-relationship form. It employs triggers to propagate updates to relations and predicates to express constraints on relations. APPL/A includes both procedural and declarative capabilities and supports multiple process representations.

Programming models are very procedural in nature and, at this time, appear to have problems scaling to large process-based environments.

*Functional Models:* Functional models use declarative, textual languages to represent processes as "sets of mathematical functions depicting relationships among inputs and outputs" [Curtis 92]. Each function is hierarchically decomposed until atomically automated or manual operations are

encountered. These models have capabilities to run processes concurrently or stipulate sequence, iteration, and synchronization. They often use meta-operators to allow dynamic control of behavior. One example of a functional model is the HFSP (Hierarchical and Functional Software Process) description and enaction language by Katayama. HFSP uses a process enaction mechanism to allow activity scheduling.

*Plan-Based Models:* Plan-based models are derived from artificial intelligence research. They provide mechanisms in which operators represent possible actions; these actions are selected based on the satisfaction of their associated preconditions. GRAPPLE, developed by Huff and Lesser, is an example tool.

*Petri-Net Models:* Petri-net models are based on a structure of roles utilized within a process and their interaction. Motivation to use this modeling technique seems to be based on Anatole Holt's work in applying petri nets to modeling coordination in the workplace [Curtis 92]. These models are designed to aid in representing and executing structured tasks (i.e., those planned from known dependencies). Petri-net models have been useful in restructuring organizations units based upon role rather than the traditional reporting relationships.

*Quantitative Models:* Most quantitative models are based on system dynamics, which "apply feedback and control system techniques to social and industrial phenomena" [Curtis 92]. These systems are designed to model the observed behavior of social systems. Exemplar work done by Abdel-Hamid and Madnick seeks to answer why managers chronically underestimate resources needed during a software project.

Most of the process model approaches listed above pertain to A (atomic) level process models, with the quantitative models being the notable exception. Process models at the Universal or U level are generally more functional in nature. An example of a Worldly or W level functional process model is the Process Definition Information Organizer Template System, developed for the US government-sponsored STARS project [Ett 95]. PRC uses a similar approach, as defined in Section 4.1.

### 2.3.6 Issues in Process Modeling

The following issues arise in determining the process modeling approach to be employed:

*Formality:* [Curtis 92] states that the level of formality depends upon the purpose of the process model or upon whether the agent enacting the process is human or computer. Computers seem to require more formality than humans whereas flexibility is a more necessary requirement for processes enacted by humans. Human results are more varied, but they have ability to interpret ambiguous directions. Formal process languages are usually used to define processes that will be fully automated. Although formal specifications can provide assurance of internal consistency of processes, they cannot show whether it is valuable and true to external users. [Redwine 91] discusses a set of process properties that are amenable to rigorous definition while noting that the complexities of software development organizations compound the difficulty of defining many other process properties. Organizations like PRC who implement W (Worldly) level process models are choosing a less formal model.

*Granularity and Precision:* [Kumagai 91] defines granularity as an abstraction of definitional detail about the system or item being described; for example, super-types define an abstraction of the information appearing in an instance of any of the child subtypes. Granularity depends upon the purpose of the model and the knowledge and capability of the agents. Automated processes require small granularity whereas manual ones, like those at PRC, can tolerate more abstraction.

*Scriptiveness and Fitness:* Process models describe activities in at least three different senses: prescriptive, descriptive, and proscriptive. Prescriptive modeling relates how the process ought to be performed; it characterizes the “to be” process model. Descriptive modeling relates how the process is currently performed; it characterizes the “as is” model. Proscriptive modeling relates behavior that is not allowed; it characterizes the legal values of the model. Often organizations use descriptive models as a beginning baseline of

process improvement. Prescriptive models describe a general, common process or the target process to be eventually used. The proscriptive models exercise control over the allowable steps in a process and often complement prescriptive and descriptive models. PRC uses prescriptive models initially for corporate processes and descriptive models for project processes. As the domain engineering of processes continues, PRC expects the corporate processes to take on a more descriptive nature.

## 2.4 Software Reuse

[Weide 91] presents a model of software structure, called the "3C's Reference Model." A working group at the Reuse in Practice Workshop in July 1989 initially developed it and further elaborated it during 1990 [Latour 91]. The "3C's Reference Model" defines and distinguishes three main ideas:

**Table 2-3. The 3C's Reference Model**

Concept	A statement of what a piece of software does, factoring out how it does it; an abstract specification of functional behavior.
Content	A statement of how a piece of software achieves the behavior defined in its concept, e.g., the code to implement a functional specification.
Context	Aspects of the software environment relevant to the definition of concept or content that are not explicitly part of the concept or content; additional information needed to write a behavioral specification (e.g., mathematical machinery and other concepts), or an implementation (e.g., other components).

The 3 "Cs" (concept, content, and context) can be applied to software, as shown in Table 1-1. An example of a software concept is an abstract data type for implementing stacks. An example of a software content is the implementation of the stack abstract data type in C using linked lists. An

example of software context would be the operating system, hardware, and compiler necessary to execute the stack implementation.

[Weide 91] uses this reference model to address reusability issues. The model provides an environment in which to study reuse but does not mandate reuse per se. The model facilitates reuse discussion because 1) it separates concept from content, 2) it allows a concept to be implemented many different ways (i.e., it has many different contents), and 3) the reuse of both the concept and the content are separated from the domain in which it is reused.

Of the three "C"s, context is perhaps the hardest to comprehend, yet it has significant impact on the reusability of the component. Context, or the environment in which the abstract component is used, controls the manner in which the behavior and performance of the component are adapted. Without adaptability to its context, components will proliferate and finally clog the user with an overabundance of variations on a single theme. Therefore, it is imperative that components be designed to allow behavioral adaptations (at the concept level) and performance adaptations (at the content level). [Weide 91] envisions a natural index of components based on the concept-content relationship within a domain.

[Weide 91] suggests two types of context: fixed and parameterized. Fixed context is a principle upon which the concept or content of a component is based; it offers no choice to the client. An example of fixed context is the mathematical principles upon which stacks are based. Parameterized context is a mechanism that allows the user to choose one of several concept or content alternatives. An example of parameterized context is the choice of data types that populate a stack. Since components designed for reuse should be as adaptable as possible, parameterized context should be used whenever possible. Two mechanisms to promote parameterized context are genericity and inheritance.

Genericity is the property that allows a component definition to act as template for a family of reusable abstract components. The client is



responsible for creating an instance of the template by substituting an actual type for a formal type parameter item.

Inheritance is a mechanism that allows a new component to be a variation of an existing one. A new component is an extension of its parent(s) in that it provides the same operations and possibly additional operations. Inheritance provides three types of concept, content, and context differentiation. It allows differentiation between: 1) concept and content, 2) concept and conceptual context, and 3) content and implementation context.

First, inheritance allows parent components to 'defer' the implementation of operations to its inherited children, thus providing a mechanism to separate the declaration of the operations (concept) from their implementation (content). Second, since the child inherits and hence has visibility into the functionality of its parent, it can extend the functionality (i.e., concepts) of its parent; this kind of inheritance is type or specification inheritance. Third, in a manner similar to specification inheritance, a child also can extend or replace the implementation details (i.e., content) of the parent component; this kind of inheritance is code or implementation inheritance.

PRC bases its process notation and process reuse methodologies on the 3 C's model. The notation, described in Section 4.2, separates process attributes into its conceptual, content, and contextual parts. As detailed in Sections 4.4 and 4.5, the methodologies design and tailor a process starting with its concept and next its content, using the context as a way to assess the viability of the reusable asset to the project environment.

## **2.5 Software Process Reuse Techniques**

Although software process reuse is a need faced by a multitude of software businesses, there is a dearth of articles describing how to create and tailor reusable software processes. [Bechtold 94] and [Krasner 92] describe a general process, and [Castano 93] describes a detailed method, for creating reusable processes.

[Bechtold 94] discusses five levels of abstraction: process definitions at the organizational, business area, program, project and project plan levels. Using a top-down approach, process definitions are constructed at the organizational level and refined at each subsequent lower level until the processes are ready to be enacted on a project. Typically, organization-level processes are policy statements; business-level processes focus on product line issues; program-level processes are concerned with life cycle models and partially ordered process activities; and project processes are detailed and enactable. Project-level processes are developed in two phases. The first phase, preliminary definition, defines the life cycle, various options for process steps, and the relationships between the steps. This phase focuses on product-specific requirements, standards, methods, and risks. The final definition phase selects and tailors the chosen process specifications, identifying the produced work products, and choosing methods for each process step.

[Bechtold 94] also briefly discusses using a bottom-up approach to process definition. Projects build processes that are easily implemented and have a higher potential for acceptance and success. After several implementations of the process, the process definitions are abstracted by removing project-specific details.

[Krasner 92] describes the development of generic process models for the Software Process Management System (SPMS), part of the STARS project. Process engineers build the generic model from process requirements of the organization, its process measurement and improvement goals, existing process and product components, and organizational constraints. [Krasner 92] does not elaborate on its five-step process tailoring method, composed of: edit constraints, edit process components, edit product components, edit input/output relationships, and edit model.

[Castano 93] addresses the reuse of process behaviors in an object-oriented environment and the creation of generic process specifications (i.e., process descriptions). Although the specific approach does not address the business needs of our target environment, the work contains some interesting principles. The following discussion outlines the methodological steps

[Castano 93] proposes and discusses the principles that apply to more industrial, non-object-oriented environments.

The context for [Castano 93]'s approach is the F-ORM object-oriented model developed by his colleagues. [Castano 93] defines two classes: the resource and process classes. Resource classes define agents and their characteristics. Resource roles represent the behavioral actions that the resource takes during the life cycle. Process classes describe interactions between resource classes. Process roles describe the communication or coordination behaviors that occur between resources. [Castano 93] uses bipartite graphs to rigorously represent a given process.

The [Castano 93] approach is a three-step process. The three steps are:

1. Classify existing conceptual schema - The analyst classifies existing process specifications (conceptual schemas) using information retrieval indexing criteria and clustering techniques. Schema objects within a conceptual schema are weighted on the basis of their properties, hierarchies, and relationships. Those that exceed a given threshold become schema descriptors. Schema descriptors are then compared, with pairs that exceed a threshold becoming candidates for domains or subdomains within the universe of examined schemas. The clustering of these schema descriptors into hierarchies facilitates the identification of a community of schemas from which to draw reusable schemas.
2. Design reusable resources - This phase has two parts. First, the semantic affinity of all schema descriptors is calculated. Semantic affinity is mathematically determined based upon the common properties, hierarchies, and relationships that are shared by two of the schemas. Again, a threshold value determines those schemas that are grouped together into affinity sets. Second, an application engineer (or domain engineer) evaluates each object in the affinity set in order to define generic resource classes. For this part, [Castano 93] has defined a function that examines and identifies those objects that have common properties and

domains. The application engineer uses these objects to determine generic resource classes.

3. Define reusable processes - The definition of reusable processes is based upon the common patterns of message passing between two objects, i.e., process behavior. This activity has four parts. First, the application engineer identifies key resources, which, although not necessary, help to reduce the number of comparisons that follow. Second, the application engineer defines the order and sequence of the message passed during the process, called coordination constraints. Third, based upon these constraints, the application engineer determines generic messages. To aid him in this task, [Castano 93] has developed guidelines for determining compatible messages, which are then used as a source for determining generic messages. Fourth and finally, the application engineer constructs the generic process class, composed only of those messages and resources that are common to the generic resources and messages.

Some software reuse principles from [Castano 93] are:

1. Consider using information retrieval techniques to cluster processes into domains.
2. Examine inputs and outputs to determine if two processes are compatible.
3. Examine resources, both roles and output products, to determine process compatibility.

### **3. Process Reusability**

This chapter describes the principles of process reusability that are used for the thesis and its case study. The sections of the chapter are:

- 3.1 Definitions - Definitions of process and process reuse are given.
- 3.2 Process Reusability - This section discusses what makes a process reusable.
- 3.3 Process Domains - Process domains are defined and explained. This section applies the domain analysis research described in Section 2.1 to processes, providing a rationale for the use of domain analysis techniques in the PRC process definition methodology defined in Chapter 4.
- 3.4 Methods of Defining Process Variability - Different ways are described to define and organize process variability. These methods are used in extensions to the graphical notation described in Section 4.2.

#### **3.1 Definitions**

What is a process? [Over 94] defines a process as:

*"**Process:** a logical organization of people, procedures, and technology into work activities designed to transform information, materials, and energy into a specified result."*

Processes that may provide fruitful study are typically those that are widely used, easily tailorable, and humanly enactable. The organization of the people, procedures, and technology, and the detail used to describe it, may vary according to the levels of expertise and training, and the criticality of the process's intended use.

What is process reuse? We define process reuse as:

**"Process Reuse:** *the use of an existing process description in the creation of another process description.* "

Process reuse can occur during the definition of the project processes as well as during the execution of those defined processes. This thesis has chosen to not address process reuse in the sense of multiple executions or enactments of the same process on a given project. Instead, the focus is on process design using reusable processes.

### **3.2 Process Reusability**

What makes a process reusable? To answer that question it is important to examine what makes software reusable. Significantly, that question has not been definitely answered even though it has been researched for over ten years; there is no foolproof objective indicator that can determine the reusability of a software module [Frakes 96] [Conte 86]. The best answer to date is simply that the software module is reusable only if it is reused. The same can be said for software processes; if reused, they are reusable.

### **3.3 Process Domains**

An understanding of process domains provides the foundation for the use of domain engineering in constructing reusable processes. The following discussion provides that understanding.

A problem domain is a set of related systems. [Frakes 93] According to [Arango 89], a body of information is a domain if:

- Deep or comprehensive relationships among the items of information are known or are suspected with respect to some class of problems,
- There is a community that has a stake in solving the problems,
- The community seeks (software-intensive) solutions to these problems, and

- The community has access to knowledge that can be applied to solving the problems.

The following paragraphs show that key process areas meet these requirements for a problem domain, as long as the third criterion that stipulates software-intensive solutions is broadened to include defined, repeatable solutions. This should be acceptable since the cited criteria were placed in the context of software systems modeling. [Prieto-Diaz 91b]

The software development life cycle can be decomposed into key process areas (KPA's). As shown in Figure 3-1, the CMM defines three categories of KPA's: engineering, managerial, and organizational. [Paulk 93b] Of these three, the first two pertain to a single software development life cycle; the third, organizational KPA's, support several software development efforts and therefore are outside the context of a single software life cycle. The engineering KPA's represent those activities we normally associate with the software development life cycle, such as software product engineering and peer reviews. Managerial KPA's represent processes that support the engineering process. Examples of managerial KPA's are requirements management, project management (including both project planning and project tracking and oversight), configuration management, subcontract management, and quality assurance. Organizational processes are activities that support several software development efforts and therefore are appropriate at higher organizational levels. Examples of organizational KPA's are organizational process focus, organizational process definition, and the organizational training program.

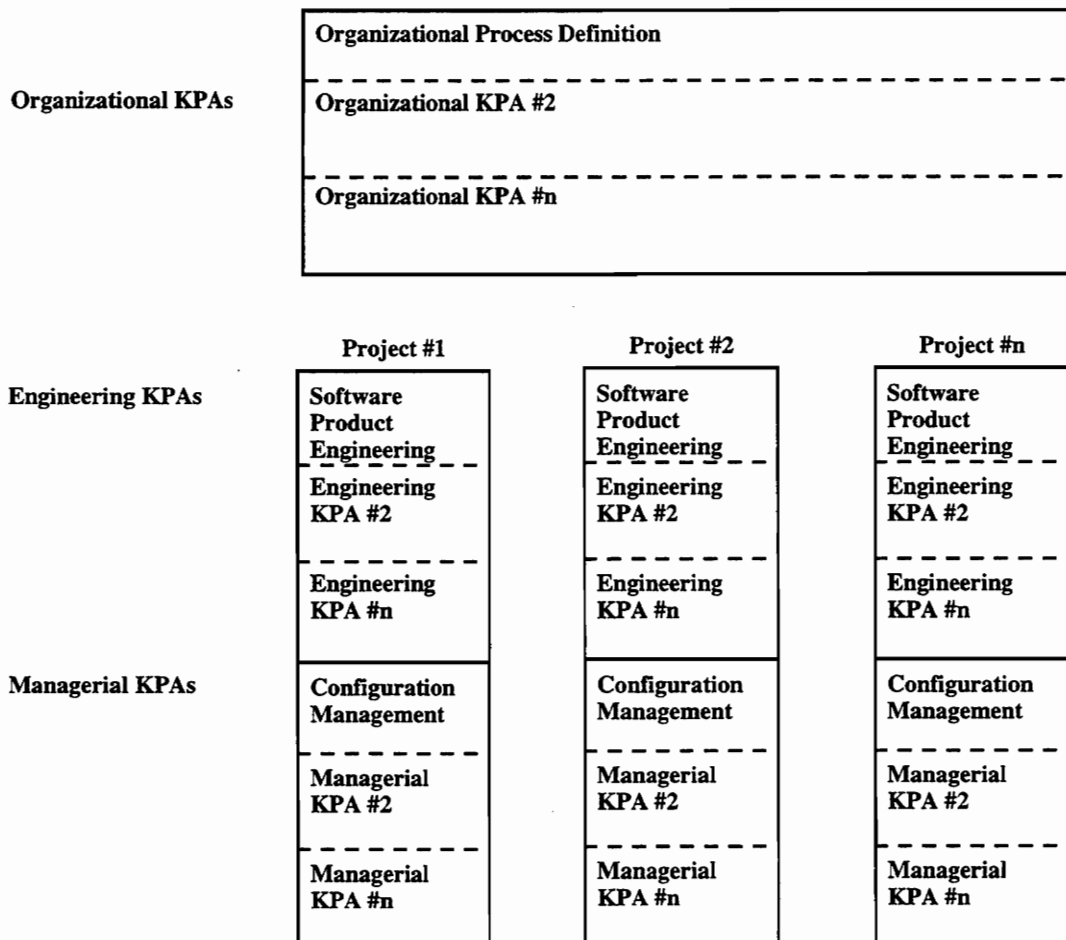
Software life cycles are composed of a set of KPA's:

$$SLC = \{K1, K2, ..., Kn\}$$

where SLC is a given software life cycle and K1 - Kn represent the KPA's of the software life cycle.

Since processes are hierarchical, processes within each of the three KPA process categories can be further decomposed into other processes. For

instance, configuration management can be decomposed into four subprocesses: configuration identification, configuration change control, configuration status accounting, and configuration audits. Each of these subprocesses can in turn be further decomposed. Likewise, software product engineering can be further decomposed into processes for requirements analysis, software analysis and design, unit code and test, software integration and test, and software maintenance. The CMM calls the lowest level of process decomposition a process element.



**Figure 3-1. The Three Categories of KPAs**

The fact that each KPA is composed of a set of processes can be expressed as follows:



$$K = \{P1, P2, \dots, Pn\}$$

where K is a given key process area and P1 - Pn represent the processes or process elements that make up the KPA.

Therefore, the KPAs organize the software development life cycle into process subsystems or domains, as shown below.

$$K = \{P1, P2, \dots, Pn\}$$

where K is a key process area and P1 - Pn are software life cycle processes or process elements within that KPA.

$$SLC = \{K1, K2, \dots, Kn\}$$

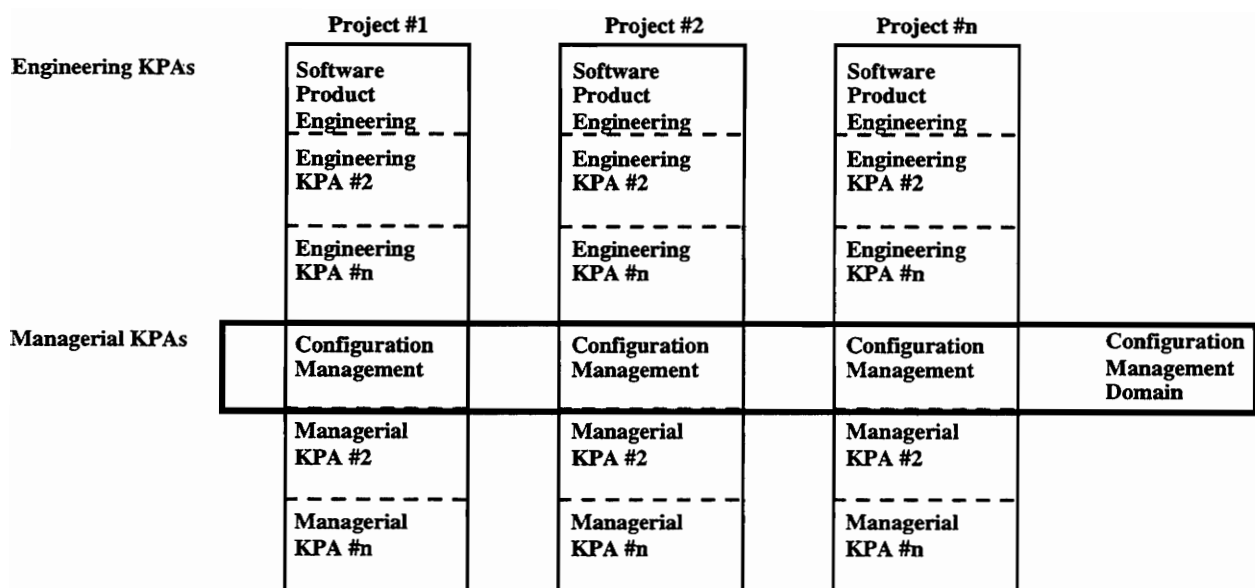
where SLC is a given software life cycle and K1 - Kn are KPA subsystems within that software life cycle.

$$D = \{SLC1K1, SLC2K1, \dots, SLCnK1\}$$

where D is a given KPA domain and SLC1K1 - SLCnK1 are related KPA subsystems drawn from the universe of software systems.

Some examples may benefit the reader. Consider the configuration management (CM) KPA. A domain of configuration management processes exists, composed of CM processes in all software systems, as shown in Figure 3-2. It would contain processes related to configuration identification, configuration change control, configuration status accounting, and configuration audits. The domain could be further decomposed into sub-domains based on system size or development platform complexity. It is reasonable to assume that high degrees of similarity exist between configuration management subsystems from multiple application systems.

Likewise, a domain of software product engineering processes is possible, where all processes devoted to creating application software products would be studied. Of course, the scope of this domain is too large to be meaningfully studied; further decomposition into smaller sets of systems, or application domains, is necessary to arrive at practical results.



**Figure 3-2. A Configuration Management Process Domain**

Using this definition of process domains, processes within a given process domain can potentially be reused by another project:

$$Dx-Sc-Pd = R(Dx-Sa-Pb)$$

where R is the reuse function that reuses a process Pb from Domain Dx used in System Sa to create a similar process Pd in another System Sc.

The potential for reuse increases as the commonalty between the reused and target systems increases; that is, reuse increases as the scope of the process domain narrows.

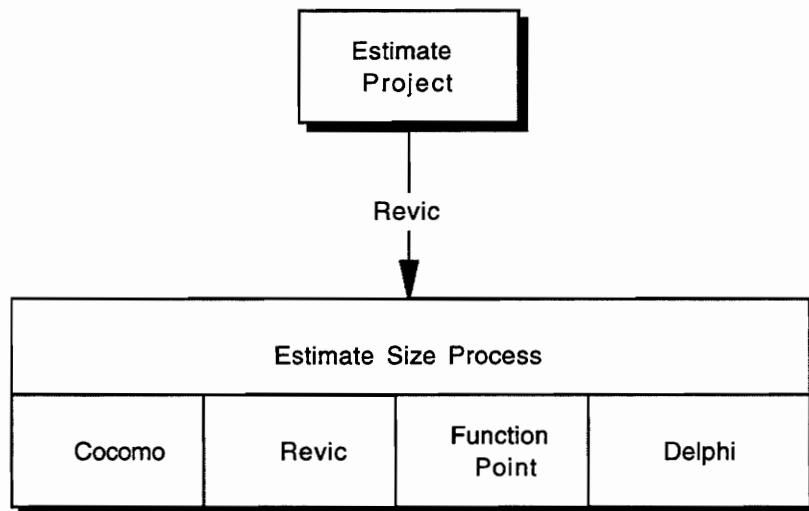
### 3.4 Methods of Defining Process Variability

To create reusable processes, the domain engineer selects a process domain, collects processes and process data, and categorizes the processes to find logical groupings of common processes. The domain engineer is really defining the common and variant elements of the processes with a process domain. This

section describes methods that the domain engineer uses to express the process domain variations.

Methods of defining and organizing variability in processes are similar to the methods used in software. [Frakes 93] defines these methods as:

- Enumeration: Process variations are described one at a time. The reuser chooses which process variant most closely represents his target process. Enumeration is the most elementary type of defining variability.
- Parameterization: Parameterization knows more about the set of process variations than enumeration; the set and range of variations are known in enough detail that the reuser can select the desired variant by naming it as a parameter to a given process. For example, during project planning, estimates of software size are often needed. There are a number of valid estimating techniques that could be used, such as Cocomo, Revic, function points, or delphi methods. As Figure 3-3 shows, an estimating process would be “called” with a parameter specifying which estimating method to use.



**Figure 3-3. An Example of Process Parameterization**

- Abstraction/inheritance: Abstraction focuses on creating generic processes rather than the reuse of existing processes, as in enumeration and parameterization. The generic processes are composed of the features that are common to the set of specific implementations. The generic process can be instantiated in a new environment, inheriting the common features plus adding features that are specific to that environment, like the hardware and software platforms, client requirements, standards, metrics needs, and project size and scope issues. The generic processes hide the details of the implementation, allowing the user to focus on the *what* not the *how* of the process.

An example of abstraction is shown in Figure 3-4. A generic baseline process was created that defines how baselines are created, validated, and promoted. The generic baseline process is used for the four major baselines created during a standard development project. Projects inherit the generic baseline process and apply it to their project environment, adding detail regarding hardware platform, operating system, and the project configuration management toolset.

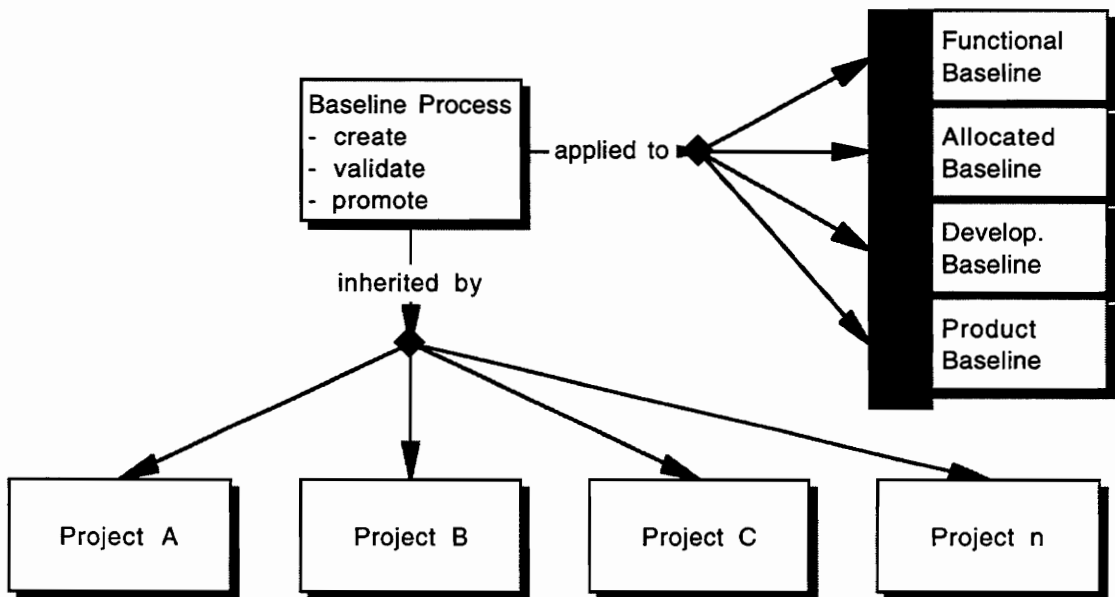


Figure 3-4. An Example of Process Abstraction

#### **4. Software Process Reuse Methodology**

This chapter describes how to create, store, access, and tailor (or instantiate) a reusable software process at PRC. The software process reuse methodology has two parts: a process definition methodology and a process tailoring methodology. The sections of the chapter are:

- 4.1 Introduction - The introduction describes the benefits of the software process reuse methodology and the PRC personnel that use it.
- 4.2 Process Notation at PRC - This section presents the standard PRC process notation in both its textual and graphic forms and proposes extensions for graphing reusable process relationships.
- 4.3 Context for Process Definition and Reuse - Based on the process engineering actions defined in Section 2.2.5, this section provides the context for the PRC process definition and tailoring methodologies.
- 4.4 Process Definition Methodology - This section describes how to create a reusable process definition using the process notation described in Section 4.2 and principles from domain analysis (Sections 2.1 and 3.3) and software reuse (Section 2.4). Section 5.1 describes an implementation of the methodology.
- 4.5 Process Tailoring Methodology - This section describes how to instantiate, or tailor, a reusable software process at PRC. The methodology is based upon the 3C's model defined in Section 2.4. Section 5.2 describes an implementation of the methodology.
- 4.6 Storage and Retrieval of Reusable Processes - The section closes with a discussion of the storage and retrieval of reusable software processes at PRC, using world-wide-web technologies.

A short word of explanation on terminology: the term "methodology" is used here as a synonym for "process" to reduce overloading of the term.

#### **4.1 Introduction**

The purpose of the process definition methodology is to create reusable processes so that projects within PRC can cost-effectively tailor them to their own environment, needs, and requirements. Standardized processes aid in transferring process knowledge between projects, reducing training costs, planning common project activities based on process data, and increasing productivity and quality in a continuously improving process-oriented environment.

PRC uses W (Worldly) level process descriptions, as defined in Section 2.3. Humans enact these processes; no process programming is performed. The process definition methodology is based upon a systems-centric viewpoint of domain engineering whose goal is the categorization of reusable software processes, and upon the common domain analysis approach outlined by [Arango 94]. The methodologies are strongly influenced by the 3 C's model described in Section 2.4; the process notation is grouped by concept, content, and context, and the methods use these groupings to sequentially perform the process design and tailoring.

Within PRC, staff dedicated full time to software process improvement use the process definition methodology, working together with subject matter experts and training personnel. There is considerable effort to use personnel of individual projects, especially if the project currently has a process improvement program. This involvement seems to be beneficial to both the full-time and the project staff; the team interaction enables a product that is more usable by the projects and the full time staff benefit from a wealth of experiences, lessons learned and implemented processes, essential when generating an abstract description from the specific examples.

## 4.2 Process Notation at PRC

For processes to be reusable, organizations need a way to express common and variant elements within a process. Frameworks provide one mechanism to accomplish this. Figure 4-1 shows PRC's framework for process definition. Several portions of the framework reflect the integration of PRC's software process improvement and quality improvement programs; see [Arthur 93] for details on this approach to quality.

This framework has several sections. The first three sections, known as the general information, customer description, and interface description sections, specify the process concept, or the *what* specification of the process. These sections describe *what* the process is, *what* it does, and *what* its interface is to other processes. The process concept follows the 3 C's model described in Section 2.4.

The next section, the procedural section, specifies the process content, or the *how* of the process, and implements the content part of the 3 C's model. This section describes how the process is performed by specifying the people who do the process and their interactions, the tasks and the order of their execution, and the tools and other necessary resources. The procedural description is both textual and graphical.

The graphical notation uses flowcharting symbols. The flowchart describes the process, its customers, and the indicators used to measure compliance in meeting customer requirements. The graph is divided vertically into columns; each column represents a participant in the process. Process steps for each participant are placed in the associated column. The graph can also be divided into horizontal bands that represent sequential steps of the process. Each process starts with an oval symbol describing the customer's need. The steps in meeting those needs are described using rectangles for processes and diamonds for decisions. Boxes with double side bars represent a collection of sub processes that is defined graphically in a separate diagram. The arrows represent control flows. The graph does not capture data flows. Circular symbols denote Indicators, described below.

The context description describes the context in which the process can execute. The process context defines domain, organizational, project, and managerial criteria for application of the process.

The final section, the measurement description, spans both the *what* and *how* specification roles. The quality indicators measure how well the process meets its goal and are therefore part of the *what* specification. The process indicators measure how well individual steps perform. Because the process indicators measure steps, they are part of the *how* specification.

The standard symbols can be augmented to represent reusable processes as shown in Figure 4-2. Reusable processes and process elements use rectangles drawn with dashed lines; non-reusable, or project-specific, processes and process elements use solid lines. Enumeration is described using a branch point that splits into the various enumerated process choices and then converges back to a single point. Parameterization is described using a single process box from which diverge a set of parameterized units or methods. The parameters are labeled on the arrow from the single box to the associated method. Lines from the various methods converge back to a single point before control passes to the next process element. The abstraction and instantiation paradigm is described by using a dashed rectangular box from which other process boxes diverge, each box representing an inherited process. If the box uses solid lines, it represents an instantiated process. If the box uses dashed lines, the inherited process is still reusable but has inherited characteristics from the abstract process.

These additional symbols can be grouped to show more complex arrangements between reusable processes. For instance, enumeration and abstraction can be combined to show the set of choices to be made at a given process step.



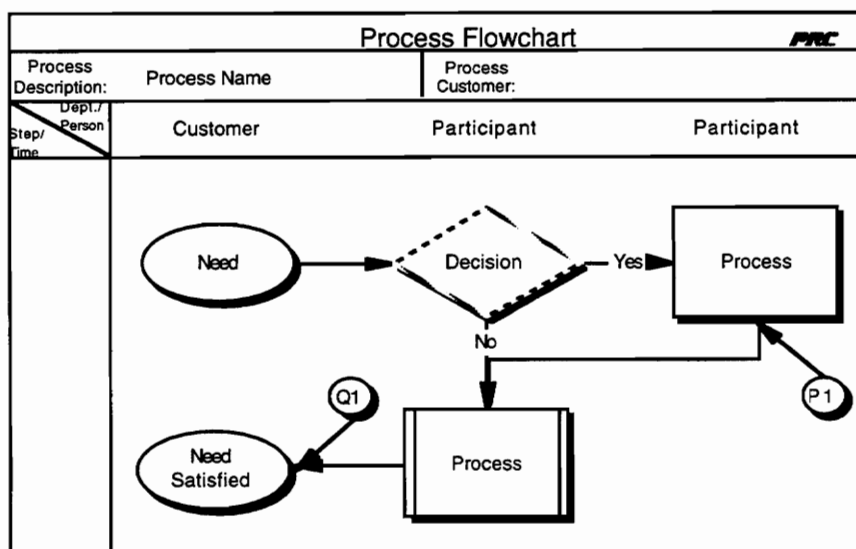
<b>Name</b>	Name the process/sub process that is described within the document.
<b>GENERAL INFORMATION</b>	
<b>Process ID</b>	Unique process identifier
<b>Process Purpose</b>	Provide a brief description of the purpose and objective of the activity.
<b>Standards</b>	Identify the applicable process and product standards, including the SEI CMM KPA reference.
<b>Related Processes</b>	Identify processes that are related to this process, especially if this process is part of a set that is normally viewed as a whole.
<b>Version Number</b>	Configuration management version number used in PAL
<b>CUSTOMER DESCRIPTION</b>	
<b>Customer</b>	Identify the internal and external groups who benefit directly (receive products/services) from the results (outputs) of this process.
<b>Customer Requirements</b>	List each of the legitimate requirements that have been negotiated and agreed to with the identified. These requirements should follow the RUMBA criteria in that they should be Reasonable, Understandable, Measurable, Believable, and Acceptable.
<b>INTERFACE DESCRIPTION</b>	
<b>Entrance Criteria</b>	Identify the criteria that must be satisfied before the activity can be initiated. The criteria might say how to tell when a process can be started, for example at the conclusion of another activity or process.
<b>Inputs</b>	Identify the work products that are used at any point in the process.
<b>Outputs</b>	Identify the work products that are produced during the process.
<b>Exit Criteria</b>	Identify the criteria that must be satisfied before the activity can be considered complete. Exit criteria summarize the salient measurable tasks of the process.
<b>PROCEDURAL DESCRIPTION</b>	
<b>Responsibilities</b>	Describe the groups that participate in the process.

**Figure 4-1. Process Notation at PRC - Part 1 of 3**

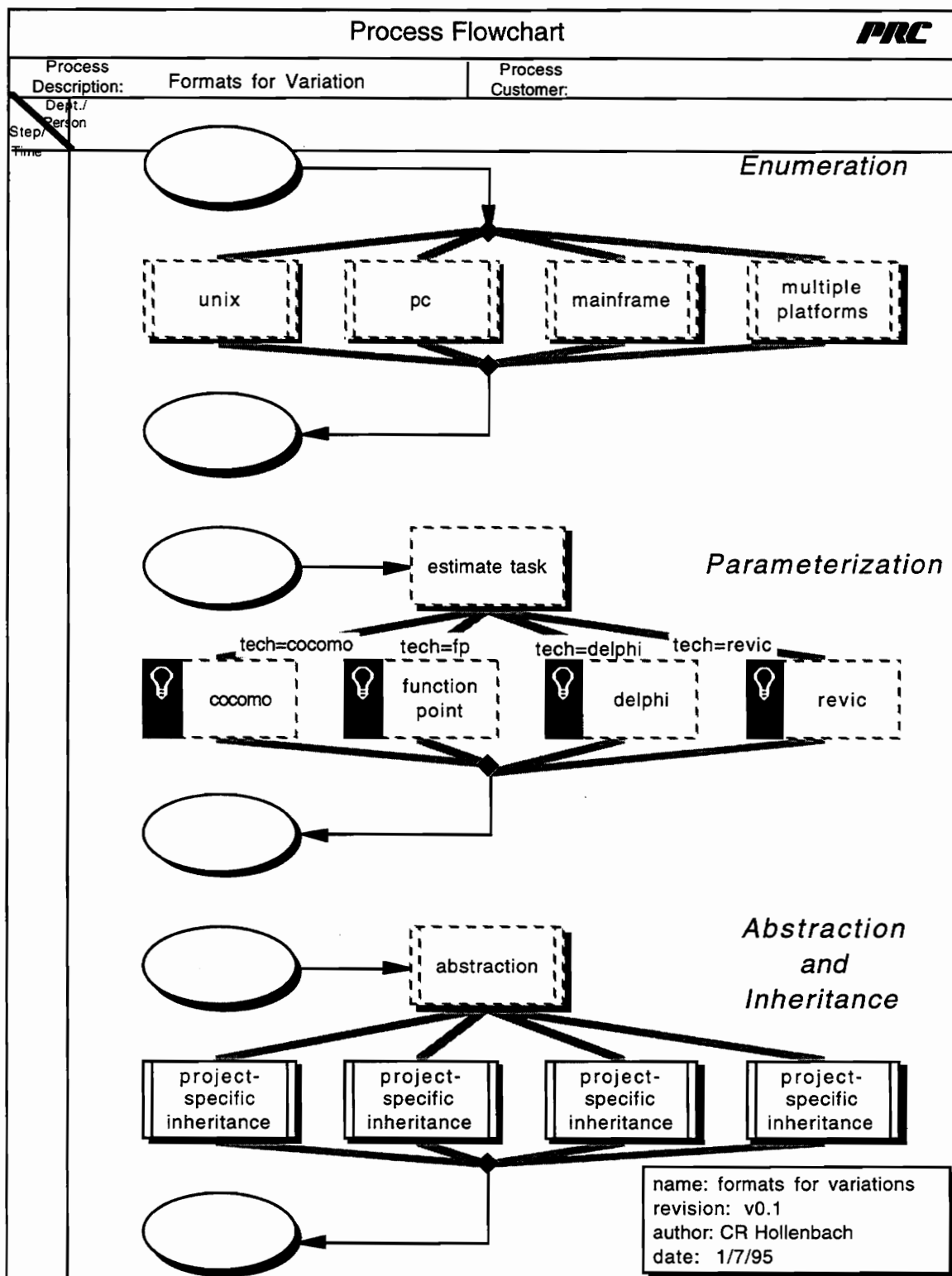
<b>Tasks</b>	Describe the tasks that must be accomplished within the process. For ease of reference, the tasks should follow the process diagram referenced as the main exhibit. If the process is procedural, describe the tasks in the order that they must be accomplished, numbering each task step. Parenthesis the responsible group to the left of the task, as shown below:  (<Participating group>) <Task description>  Use action verbs to describe the tasks. Reference by process ID all tasks that are further described elsewhere. Note any particular procedures, practices, or methods that are employed in any step.
<b>Tools</b>	Describe suggested or mandatory tools used during any step of the process.
<b>Resources</b>	Describe resources that are necessary to enact the process.
<b>CONTEXT DESCRIPTION</b>	
<u><b>Domain</b></u>	
<b>Applicable Domain</b>	List the application domains to which this process is applicable.
<b>Required Domain Knowledge</b>	Describe the knowledge of the application domain that participants in this process must exhibit.
<b>Usage Information</b>	Describe how past projects have used this process, including the results of the process (i.e., associated metrics) and the lessons learned.
<u><b>Organization</b></u>	
<b>Organization Size</b>	Briefly describe the organization size that limits this procedural method.
<b>Organization Structure</b>	Define specific groups or functions that must be in place to execute this procedural method.
<u><b>Project:</b></u>	
<b>Project Duration</b>	Describe the applicability of this process in regards to project length, e.g., any project that requires over 1 person-month.
<b>Problem Complexity</b>	Describe complexity constraints. Generally phrased in terms of software size.

**Figure 4-1. Process Notation at PRC - Part 2 of 3**

<b>Software Engineering Skill Level</b>	Describe the software engineering skill level that is needed. If the team is less skilled, compensate with more peer reviews or with higher skilled people on the review team.
<b>Physical Team Locality</b>	Define whether the team must be collocated or can be physically distributed.
<b>Communication Infrastructure</b>	Describe what information resources and inter group communication paths or mechanisms must exist.
<b><u>Management</u></b>	
<b>Cost/Benefit</b>	Describe the costs and benefits of this process in both the short and long term.
<b>Risks</b>	List management and technical risks associated with the execution of this process.
<b>MEASUREMENT DESCRIPTION</b>	
<b>Quality Indicators</b>	Describe those performance (or outcome) measurements of this process. These indicators should be linked closely to valid customer requirements. These measures should be measurable, verifiable, and cost effective.
<b>Process Indicators</b>	Describe those measurements that are to be taken at critical points during the process and used to track and assess the effectiveness of the process itself. These in-process measures should also be measurable, verifiable, and cost effective.



**Figure 4-1. Process Notation at PRC - Part 3 of 3**



**Figure 4-2. Formats for Variations**

### 4.3 Context for Process Definition and Reuse

Figure 4-3 depicts the context of the process definition methodology and shows the life cycle of a process description. The figure describes graphically the steps necessary to create a standard reusable process description, instantiate it for use on a PRC project, refine its use, and feedback information to improve the process description for subsequent use.

This process life cycle has the following several steps:

1. Initiate the Process Definition Effort. The process definition effort begins when the PRC Software Engineering Process Group (SEPG) gives its approval to begin the work. The work is usually given to the full time SPI staff or to a working group that contains full-time support.
2. Define the Reusable Process. This step creates process descriptions that are reusable on projects by using domain analysis techniques. The output is one of more process descriptions, along with tailoring guidance and output work products; these are stored in the Process Asset Library (PAL) and usually packaged into a process manual or handbook. The process descriptions also are integrated into the PRC Standard Life Cycle (SLC) and statically tested to ensure they are fit for (re)use. This step draws many of its requirements from the Organization Process Definition Key Process Area (KPA) of the Software Engineering Institute's (SEI) Capability Maturity Model (CMM) for Software [Paulk 93b]. See Section 4.4 for a fuller discussion.
3. Develop Training for the Reusable Process. Once the reusable process is completed, training for that process is developed. The training is stored in the PAL and provided to the projects for tailoring and subsequent delivery to the project staff. Changes to the organizational process are captured in change request forms and sent to the "Organization Process Definition" process for incorporation in future processes.
4. Instantiate the Reusable Process on a Project. The reusable process is tailored to meet the specific requirements and environment of a PRC

project. Project-specific process descriptions are the result, again stored in the PAL and sometimes packaged into a handbook or set of desk instructions for staff reference. Change requests are passed to appropriate earlier process steps. See Section 4.5 for a complete discussion of this step.

5. Tailor the Reusable Process Training to the Instantiated Project Process.

The training for the reusable process is tailored to match the project-specific process description. The training is given to the appropriate staff just before they enact (or execute) the process. The tailored training is stored in the PAL and changes resulting from lessons learned are sent back to the appropriate process for inclusion.

6. Enact the Process on the Project. The project-specific process is enacted, i.e., put into practice on the project. Project management tracks the enactment of the process in order to monitor and control the project. The quality assurance function audits the project staff to ensure that the process is faithfully enacted. Measurements are taken, stored in the PAL, and used in the following step.

7. Refine the Process. On the basis of the measurements collected during the previous step, the process is evaluated to see if it is stable and capable. If it is not, an analysis is performed to understand the common or special causes. The process definition is then refined, re-trained as appropriate, and re-enacted. Refinements are sent to the "Project Process Definition" process, and change requests are sent to the appropriate organizational process if applicable. The goal of this step is to produce a process that is predictable and produces a consistently high quality product.

8. Administer the Process Asset Library (PAL). All reusable processes and related training are inserted into the Process Asset Library (PAL), as are other related process assets. The "Administer the Process Asset Library" process ensures that the PAL assets are current, measures the PAL usage, and removes items that are no longer useful. PAL assets are controlled using appropriate configuration management practices.

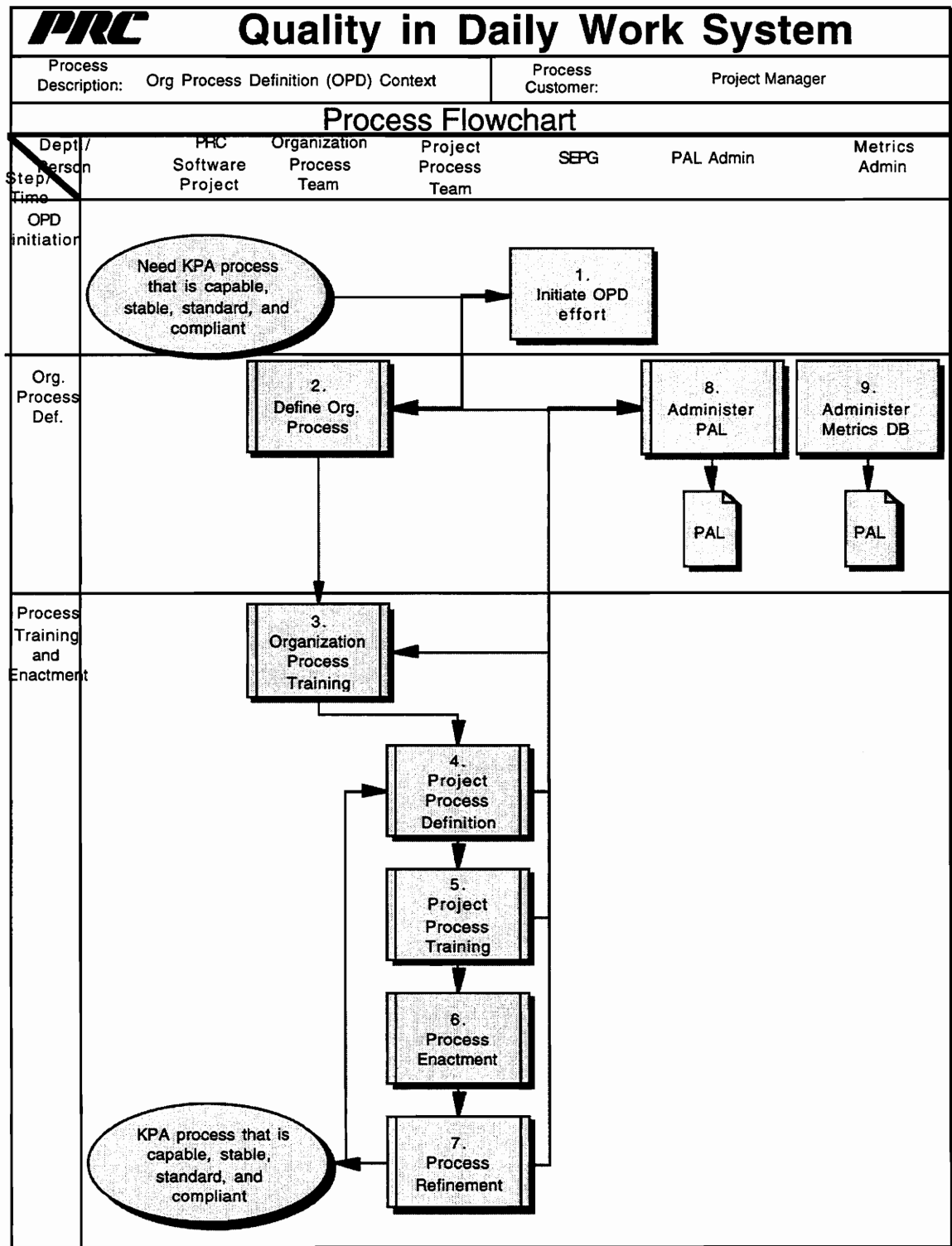


Figure 4-3. Process Definition Life Cycle

One PAL administrative task of particular interest is the change control to the organizational process. During the execution of the first six steps, problems in the process definitions and training are sometimes encountered. These problems are documented and changes to the process are suggested. The "Organization Process Definition (OPD) change control" process describes the review, assignment, and authorization of these changes to current or upcoming process definition and training efforts in a controlled fashion.

9. Administer the Metrics Database. Measurement data from the process (collected in step 5) is placed into the metrics data for analysis and subsequent improvement efforts.

#### **4.4 Process Definition Process**

This section describes how to create a reusable process description. The method draws its roots from a generalization of standard software development and domain analysis techniques, particularly Arango's study of domain analysis methods. An implementation of this method is described in Section 5.1.

The objective of the process definition methodology is to develop and maintain a reusable set of software process assets that improve process performance across the projects and provide a basis for cumulative, long-term benefits to the organization.

##### **4.4.1 Customers**

The customers of the reusable process definition process are:

- PRC projects that want to instantiate the reusable process on their project, either at project startup or for a process improvement initiative.
- PRC Proposal efforts - for use of both the reusable processes and work products as well as the measurement data in the metrics database.



- Reusable Process Training efforts - The reusable process description is a necessary input to the process of creating the associated training.
- Software Engineering Process Groups (SEPGs) - for the control of changes to process assets.
- All PRC employees - for access to and use of the PAL.
- Senior and Project Management - for use of the metrics database

The customer requirements for the reusable process definition process are:

- The reusable process descriptions must be enactable practices that can effectively be put into practice within PRC.
- The reusable process descriptions must be reusable by the PRC software project community.
- The reusable process descriptions must reflect CMM-compliant practices on PRC projects wherever possible. The processes must make use of project processes currently in use within the PRC project community whenever possible.
- The reusable process descriptions must be available to the entire PRC organization.
- The reusable process descriptions must define measurement data, including measurement collection mechanisms.

#### **4.4.2 Process Interfaces**

This section describes entrance criteria, inputs, outputs, and exit criteria for the reusable process definition process.

As for entrance criteria, three exist. First, a system to maintain and support reusable process definition efforts must be completed before this method is enacted. The system includes the establishment of a PAL and a metrics database, the definition, development, testing, and training of this process,

and the selection and use of associated tools. Second, policy must be written and approved which mandates reusable process definition. Third, adequate funding and resources must be provided for the development, support, and maintenance of the reusable process definition process, the reusable process descriptions themselves, the PAL, and the metrics database.

The following are inputs to the reusable process definition process:

- Approval of a reusable process definition process effort to design a set of standard, reusable processes.
- Project processes, methods, and tools within the given process domain.
- PRC and client process and product standards.
- Process asset change requests.
- Process assets
- Process and product metrics

The exit criteria for the reusable process definition are as follows: The reusable process definition process is completed when the process is defined in detail, integrated with the current process architecture, and is proven to meet process and product goals.

The following are outputs from the Organization Process Definition (OPD) process:

- Process Definition Action Plan
- Process Analysis Document
- One or more process descriptions
- Process Handbook
- Process Static Analysis Report

The reusable process descriptions use a standard template, shown in Figure 4-1. All reusable and project-specific processes use this form, providing a standard, transferable format between projects and organizations. The following section further describes the process definition activities.

#### **4.4.3 Process Activities**

This section describes the six activities in the reusable process definition method.

*Process Requirements Analysis:* This step defines the operating conditions under which the process will be enacted and the goals and requirements that the process will satisfy. A process definition and evaluation action plan are constructed that contains this information plus basic project planning data.

*Process Analysis:* The process analysis step provides a basis for generic reusable process definitions by examining current project-specific processes and relevant industry process data. Domain boundaries and content are specifically described and current processes from within PRC are collected, as well as all other relevant process data. A thorough literature search should be performed to include available technical books and articles, world-wide-web searches, externally and internally available processes, standards, and interviews with domain experts. Note that the volume of process data will vary with the process domain; for instance, literature on change control processes is much more available than that on domain analysis processes.

The goal of process analysis is to find the essential attributes of the processes and to use them as the basis for grouping the processes. The nature of the analysis is very intuitive; yet, some basic guidance can be given. The process domain data is evaluated to locate entities, events, operations, or relationships that are then modularized. These items are then analyzed to determine similarities, variations, combinations, and trade-offs that help structure the process for reuse by PRC projects.

Similarities and combinations identify possibilities for creating common processes or process features. Similarities between all or most of the project-

specific processes provide the basis for a reusable process or process feature. An abstraction of the project-specific processes that contains all similarities can be constructed. If similarities exist among a few but not all processes, then these combinations can be considered for a common framework for a subset of projects.

Variations and trade-offs describe possibilities for reusable process variants. If there is enough knowledge about the possible ways a process may vary for a given feature, then that feature may be parameterized. If this is not possible, then the known variations are enumerated for later selection by the projects. Trade-offs point to possibilities to decompose a process into process elements and capture the trade-offs in a similar fashion as with the variations, as described above. Another option is to abstract the common features into a reusable process and allow the projects to add project-specific features, relating to the trade-offs, to an instantiation of it. Refer to Section 4.2 for a description of how process flowcharts depict variations.

*Process Preliminary Design:* In this step, precise and accurate descriptions of the process and its interfaces are created. For each process, the customers and their requirements are defined as are the process inputs and outputs, dependencies, entrance and exit criteria, and quality indicators. The following sections of the process notation, described in Section 4.2, are used for this purpose: general information, customer description, interface description, and the measurement description (quality indicator part).

Entrance criteria tell the user when a given process can be initiated while exit criteria summarize the essential process activities that must be accomplished before the process can be terminated, often accompanied by some standard of performance. Inputs are the work products that are used during the process; outputs are those that are produced. Quality indicators describe measurements that give an indication of whether and how well the process client's needs were fulfilled. Section 4.1 more fully discusses entrance and exit criteria, inputs, outputs, and quality indicators.

When designing the process interfaces, it is important to integrate the process into other existing processes. PRC has constructed a process architecture of standard PRC processes. This process architecture integrates processes from all software engineering disciplines. The architecture is the starting point for integration of the reusable process. It also provides a framework for reference during reusable process definition.

There is a logical five-step sequence that can be used to determine the process's quality indicators, based on [Basili 84]; the first two activities occur during the analysis step while the last three occur during this step. They are:

1. Define the customer. Customers were determined during the "process analysis" step. Remember that for a given process there can be multiple customers: external clients, end users, the internal users of the process's work products, and senior management. If possible, identify a single primary customer.
2. Define the customer requirements. Again, customer requirements were determined during the "process analysis" step.
3. Define customer goals. From the requirements, select a set of high priority requirements and set targets for them that are reasonable, understandable, measurable, believable, and achievable.
4. Define a goal-related question. For each of the customer goals, determine a question that shows progress towards meeting that goal. If the process will probably attain somewhere around 85% or more of the goal, phrase the question in the negative to highlight a more usable scale.
5. Define a question-related metric. For each of the questions, determine a metric that will measure compliance with the goal and answer the associated question. PRC calls this metric an indicator. There are two types of indicators: quality indicators and process indicators. Quality indicators measure how well the process fulfilled the desired process goals; process indicators are an "upstream" measure that show how well the process will fulfill the given quality indicator.

*Process Detailed Design:* The process detailed design step defines the procedural, or "how"-related, process characteristics as well as the contextual information for the process. These characteristics include the methods, procedures, roles, process indicators, and feedback and control mechanisms. The characteristics are defined using the process notation described in Section 4.2, specifically the procedural, context, and measurement (process indicators) sections. The reusable processes can be published in a process handbook, for subsequent tailoring to project-specific characteristics.

It is important to use roles rather than agents in the reusable process. Roles refer to essential task-oriented functions. Agents refer to the specific people or groups that perform the roles. Agents differ greatly between projects, based on project size, contractual commitments, subcontract arrangements, past project organizations, and other project environment concerns. Because of this variability, the reusable process descriptions describe functions and allow the projects to map the functions to the organizational agents as appropriate. In a few situations where CMM requirements are specific to an agent, an agent implementation of a function is described within the process description.

Process indicators are designed during this step. As noted above, process indicators are metrics that are collected during the implementation of the process that provide an early indication of how well the quality indicator will be fulfilled. Refer to Chapter 3 for more information about quality and process indicators.

*Process Code and Unit Test:* The process code and unit test step automates a reusable process or a portion of the process. Trade studies are performed that weigh the benefits of commercially available tools versus the construction of an internally developed tool set. One example of a process that can be automated is the configuration management check-in and check-out function; the automation is usually supplied through a configuration management version control system or a robust commercial configuration management toolset. If commercial tools are selected, there is inevitably the task of tailoring the tool to meet the characteristics of the organization or

project. In this case, the process definition and its contextual process architecture provide the necessary information to effectively tailor the tool; in other words, process definition is a necessary precursor to process automation. If it is decided to internally develop the automation, the process description then becomes the functional specification for the automated process, also called the process program. In this case, the process code and unit test step transforms the enactable process description into an executable process program.

*Process Integration and Testing:* The process integration and testing step ensures that the defined reusable process can meet product and process goals. This is usually accomplished through static analysis of the process and the process architecture that contains the integrated process.

*Process Measurements:* The reusable process methodology is measured using quality and process indicators. As discussed in Section 4.2, quality indicators measure the success of reaching the customer's valid requirements, whereas process indicators are upstream measures of the process's capability of reaching those requirements.

Quality Indicators: The following are the quality indicators:

- Number of projects consulted per organization process. Of the available projects, prioritize those with the highest measured maturity score in the associated process area (using the PRC maturity questionnaire).
- Percent of project-specific process reuse, as defined by:

$$(RP/AP) * 100$$

Where

RP = Number of reused project-specific processes per organization process.

AP = Number of available project-specific processes per organization process.

Note that during the initial implementation of the organization process definition process, this metric is better collected at the KPA level, rather than for individual processes.

- Standard set of project tracking metrics, including total effort, milestones missed/made, and cost. During the initial implementation of organization process definition process, there is an emphasis on schedule metrics.

Process Indicators: Process indicators for the reusable process methodology are derived from the quality indicators. For instance, number and percentage of project-specific processes used in the reusable design are captured during the process analysis phase. The project tracking metrics are collected and monitored at the end of each of the major steps in the process.

#### **4.5 Process Tailoring Methodology**

The process tailoring methodology identifies a reusable process and tailors it to meet the specific requirements and characteristics of a given project. A team of project or business unit staff or an expert in the process area perform the methodological steps, illustrated in Figure 4-4. The output of the tailoring method is a set of process descriptions that textually and graphically describes the project-specific process. These process descriptions are in the form of the process notation described in Section 4.2. An example implementation of the process tailoring methodology is described in Section 5.2.

The tailoring is performed at the direction of the local SEPG; the tailoring plan is documented in the SEPG's software process improvement plan. After planning, the next step is the selection of an organizational (i.e., reusable) process, downloading it from the PAL. The process may include a number of sub processes that are also selected for tailoring. The tailoring team may also select from the PAL a variation of the reusable process that is useful for a subset of projects with certain common characteristics (e.g., prototypic efforts, contract type, hardware platform, specific data models).



The tailoring team first modifies the general description section. The process name, identification number, and version number are modified to reflect the project's ownership. The purpose, standards, and related processes are modified if necessary. Related processes are those that interact and support the process; they do not include the "calling" process.

The tailoring team then adds detail to the customer description section of the reusable process so that it describes specifically the customer of the process. The team carefully identifies the primary customer, whether it is external, management, or the staff that executes the 'next process.' Often there is more than one customer.

Through discussions with the customer, the team elicits requirements, adding project-specific details to those requirements already existing in the reusable process description. Standards are often a fertile source of requirements that must be allocated to specific processes. The set of requirements, along with the organizational process definition, then become the constraints on the process tailoring; requests for deviations are presented to the Software Engineering Process Group (SEPG) on a case-by-case basis. In this way, consistency between projects is maintained and arbitrary process changes are minimized.

Using knowledge of the specific project input and output work products and the existing process architecture, the team adds project-specific detail to the interface specification section of the reusable process description. The team also tailors the reusable process metrics, defined in the measurements description section of the reusable process description, to measure how well the customer's requirements are satisfied. The team tunes the metrics to the peculiarities of the project contract, available collection methods, perceived risks, and maturity level, among other things.

Next, the process methods and associated indicators are tailored to accommodate the specific functional assignments on the team, expertise in currently used methods, and existing project processes. The functional roles defined abstractly in the organizational processes are applied to specific groups

or individuals. The organizational process is compared with existing processes or process fragments for consolidation. The goal is to achieve the process purpose with as minimal an effect on the existing process as possible. Finally, the process context is completed so that the contextual constraints of the process are recorded.

Like the process definition process, the next steps are to automate the process, if applicable, integrate it into the process architecture, and test it. Process automation identifies opportunities to add tool support to portions or all of the process, and to design and acquire those tools. The testing is accomplished through peer reviews, augmented by standard testing techniques for the automated portions of the process.

Finally, the team gives the tailored process to the local SEPG for final approval, along with all applicable waivers and deviation requests. The approved process is added to the PAL, where it is accessed by those responsible for the reusable process in order to modify or amplify the existing reusable process where appropriate.

A potential for simple automation exists during the tailoring process. Some of those modifications that the tailoring team makes are minor, like changing the name, identification number, and version number in the general description section to reflect project ownership. A macro language and associated tool could be used to generate these changes when the process is checked out of the PAL. As more knowledge of the process develops, the tool could be expanded to assist the tailoring team in selecting the type of modification.

#### **4.6 The Storage and Retrieval of Reusable Processes**

This section outlines how reusable processes are stored and retrieved during the process of both generating reusable processes and reusing these processes.

The "Administer the PAL" process description that follows describes maintenance and support activities for the Process Asset Library (PAL), a repository of software process-related documentation.

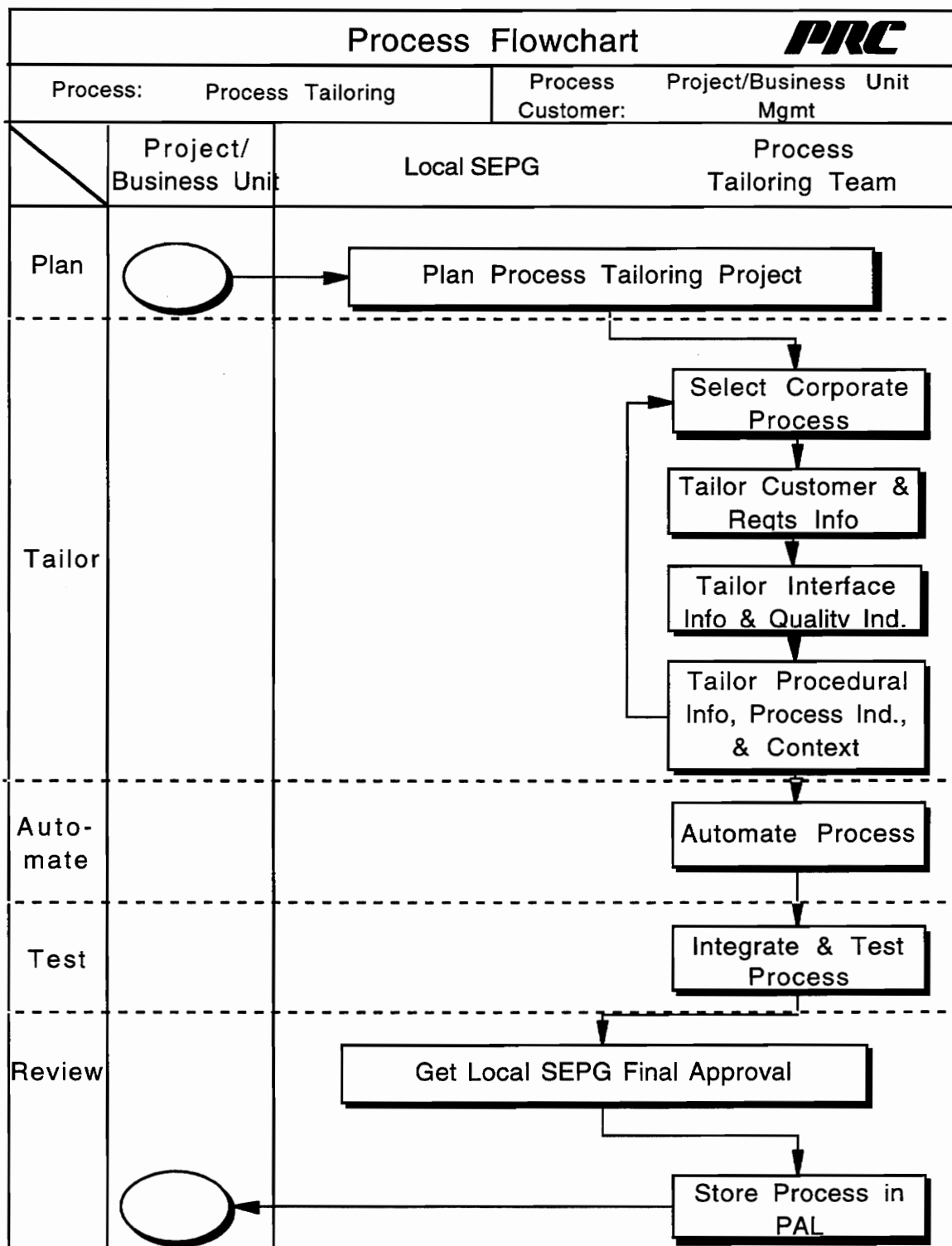


Figure 4-4. Process Tailoring Methodology

PRC has employed World-Wide Web (WWW) technologies as a basis to implement and maintain the PAL. PRC employees access the PAL through an internal network, whose security allows only PRC employees to access it. The PAL uses hypertext markup language (html) files to describe assets. Search, Suggest, and Contribute functions are provided. The PAL stores assets in compressed formats and expands them when transferring them to the employee's desktop. WWW technologies provide the means to collect and analyze PAL usage and to connect to other asset libraries around the world. PRC remote sites that do not have access to the PAL are sent stand-alone versions of the PAL at regular intervals.

The customers of the "Administer the PAL" process are:

- Software Process Improvement (SPI) community within PRC - Anyone involved in SPI is a potential customer of the PAL. Significant subgroups of the community are the PRC SEPG and lower level SEPGs, the Phoenix teams, SPI working groups, projects involved in project startup, and project members involved in project SPI activities.
- Proposal staff - Proposal personnel who are concerned with software management, quality assurance, configuration management, quality improvement, SPI, or technical processes can find applicable documents in the PAL.

The customer requirements for the "Administer the PAL" process are:

- The PAL must contain current copies of all highly demanded process assets in a usable and secure environment.
- All process assets must be placed under configuration management control, i.e., they are managed and controlled.
- Candidate documents must be reviewed and approved before placed in the PAL.
- PAL documents must be catalogued for easy access.

- The PAL contents must be made available to software projects, proposal efforts, and other software-related groups within PRC.
- The use of the PAL contents must be monitored, reviewed, and used to maintain the library.

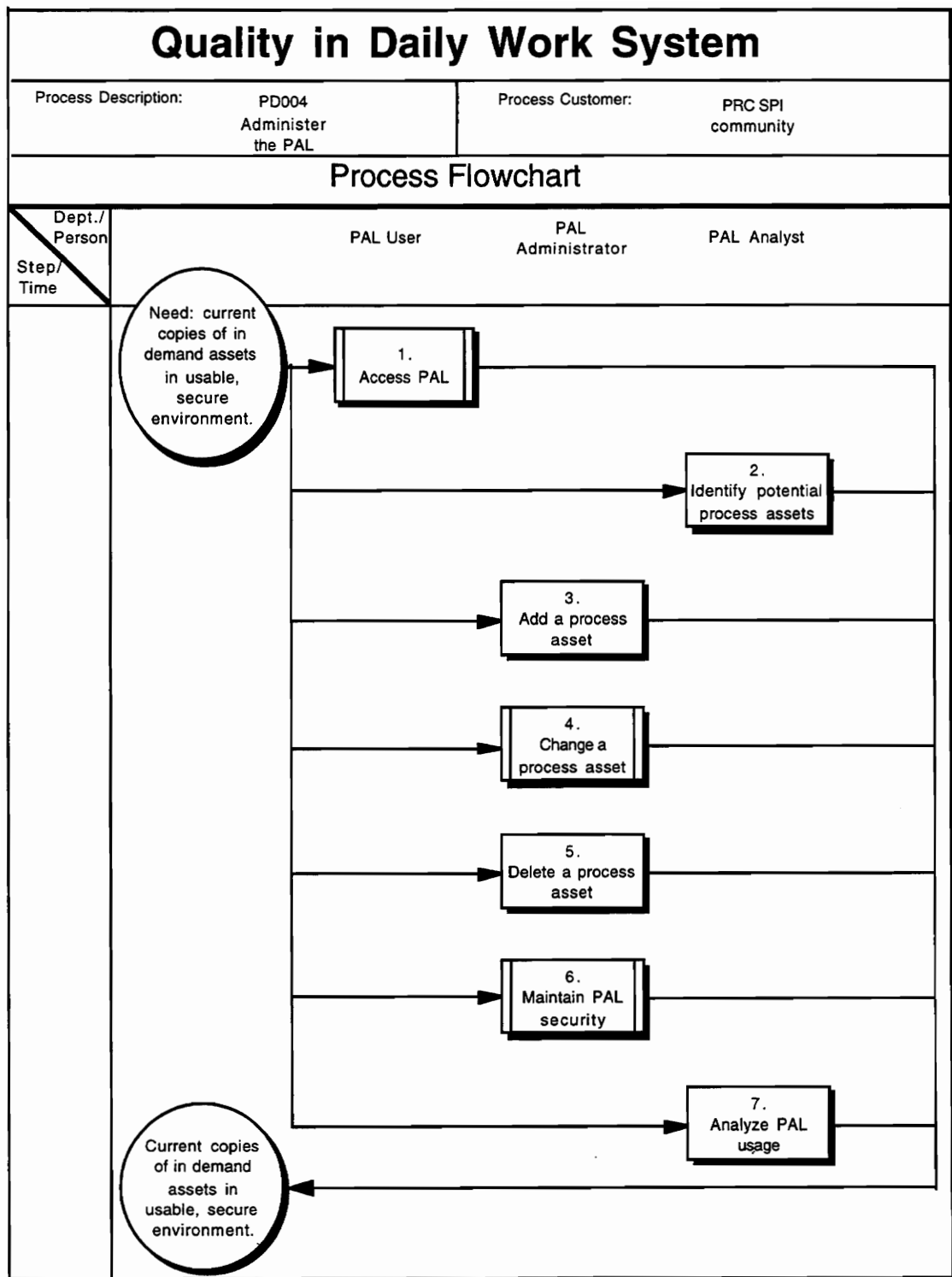
Figure 4-5 depicts the activities in the "Administer the PAL" process and the following paragraphs describe them. Note that the activity numbers in the figure correspond to the numbers below:

1. Access the PAL. Access to the PAL is gained through standard WWW measures. Using a web browser, users access the PAL through a published address, the universal resource locator, or URL. Once the user has entered the PAL, '.html' text describes the various ways to access the process assets.
2. Identify potential process assets. The PAL administrators determine potential process assets through a variety of means: polling the SPI community, looking for assets in each of the KPAs, requesting processes and related work products from the active SPI projects, etc. The administrators add the identified process assets to the PAL, and circulate a list of newly added items to the FSG SEPG and the rest of the SPI community.
3. Add a process asset. PRC employees add the process assets through the "contribute" function. The user completes a submittal form that is sent to the PAL and logged. The employee also sends a copy of the asset to the PAL, placing it in a contribution directory. Each day the PAL locates the newly sent submittal forms and the matching asset, appends the correct application suffix, formats and compresses the asset as necessary, and places the asset in the appropriate location within the PAL. When the asset is placed into the correct location, it is also placed under configuration management control using a revision control system.
4. Change a process asset. Assets are changed by contributing a new version of the asset to the PAL, using the "add a process asset" activity described above.

5. Delete a process asset. The PAL administrator deletes the process asset after there is not sufficient interest for its inclusion in the PAL. The deletion is accomplished by removing all references to it from all PAL '.html' files and by deleting its entry from an internal asset table. Deletion occurs by removing all references to it from the PAL '.html' files; the deleted file still physically resides on the PAL and still has its versions stored in a revision control system.
6. Maintain PAL security. At the point of writing, security for the PAL is a function of a secure network gateway.
7. Analyze PAL usage. World-Wide-Web (WWW) servers have the facility to collect usage metrics based on access and download. The PAL makes use of these collection functions. On a regular basis, PAL administrators analyze the usage to anticipate future demands and problems.

Scripts running on the PAL server collect the following indicators:

- Number of PAL accesses per day
- Number of added, modified, and deleted processes per month
- Top 25 most highly-used assets



**Figure 4-5. PAL Administration**

## **5. Implementations of the Software Process Reuse Methodologies**

This chapter discusses two representative implementations of the software process reuse methodologies. Section 5.1 describes the definition of reusable configuration management processes, based on the process definition methodology from Section 4.4. Section 5.2 describes the tailoring of a reusable peer review process, based on the process tailoring methodology from Section 4.5.

### **5.1 Definition of Reusable Configuration Management Processes**

As discussed earlier, the project to define configuration identification processes was part of a larger effort to revise current PRC Level 2 KPA process definitions. In the case of Configuration Management (CM), some project configuration identification process descriptions existed but many projects had not documented the processes that were in place. Corporate configuration identification processes existed in a rudimentary form. A common framework was needed to increase the potential of reusing both project-specific and corporate CM processes.

The new framework addresses both system and software configuration management and included processes for the creation of all system development baselines; identification and labeling of system components; revision control of both documentation and software; creation of software and system builds for testing, delivery, and release purposes; and the creation of a CM library system. Special attention was paid to software-related configuration identification, as most PRC projects involved in the case study were either largely or wholly software projects and these projects wanted to improve their software configuration management maturity vis-a-vis the Capability Maturity Model (CMM) for Software.

While many of the projects involved in this portion of the case study have system engineering components to them, software engineering personnel often perform these activities. Therefore, the scope of this portion of the case



study was expanded to include system level configuration identification as well as software configuration identification.

The following paragraphs describe how the six-step process definition method was applied to configuration management.

*Process Requirements Analysis:* An initial survey was conducted of major configuration management projects within PRC in order to identify the state of process definition within the configuration management discipline. There were several pockets of excellence used as sources for project-specific process definitions, tool information, and general CM expertise. Also identified at this time were the applicable PRC policies and standards. Additionally, an analysis of CM requirements within the CMM was undertaken.

The measurable goals of configuration identification were established as the integrity of the CM baselines, measured by the number of incorrect baseline items over the total number of baseline items.

*Process Analysis:* During process analysis, process data was collected, the domain boundaries for configuration identification were defined, and relationships between the project-specific processes were identified, modularized, and analyzed.

Process data was gathered from several sources. First, those PRC projects that had defined CM processes sent them for analysis. Other data sources were existing corporate process definitions, military standards, configuration management textbooks, and configuration management plans.

There were several questions regarding domain boundaries. First, should configuration identification or configuration status accounting include the construction of CM libraries? We chose configuration identification because libraries were needed for more than the accounting function. Second, should baselines be included? Baselines were included as part of configuration identification because it seemed like a natural extension of labeling and identification, two standard configuration identification tasks, and because several CM authorities placed baselines within configuration identification.

Lastly, how shall the versioning of system parts be allocated between configuration identification and configuration control? Again, we chose to place the versioning function within configuration identification, but designed a strong interface between the two functions' processes.

The next step was to look for commonalty. All CM standards and textbooks agreed that the major configuration identification functions are the selection and labeling of configuration elements. As noted above, we included CM baselines and libraries within the scope of configuration identification. An initial survey of PRC's configuration management experts confirmed that these were common elements of their CM efforts. We chose to define each of these as a major configuration identification process.

Next we looked for common features within each major process. Within configuration item selection, there did not seem to be further commonalty; the process data did not suggest any heuristic for selection, other than the effects of either too many or too few configuration items. Within labeling there was some consensus on types of labeling schemes, but no further common subactivities were uncovered. CM libraries were found to be more data intensive rather than process intensive, besides the obvious process steps to developing a database schema. There was, however, within the process of baselining configuration elements several possibilities for subprocesses.

A baseline is a collection of configuration elements, constructed at key development points. Processes from the projects suggested that there were several actions taken on baselines: preparatory audit, creation, and promotion. Likewise, there were several actions taken on baseline elements: creation of a baseline element, check in, and check out a version of a baseline element.

While there was consensus in the CM community on these common configuration identification processes and subprocesses, there was no consensus on how they interrelated. A standard sequence of processes had to be prepared during process design.

*Process Design:* When the sequence of configuration identification processes was examined, there seemed to be three sequential steps: preparation for development work, support for development work, and creation of baselines. Preparation for development work included creating permissions for subsequent work and providing templates for new work. Support for development work included revision support (i.e., check-ins and check-outs of appropriate work products), configuration element selection, and configuration element labeling. Creation of baselines included auditing the existence of the appropriate development work products revisions, creating preliminary baselines, verifying their correct construction, and promoting baselines when approval was secured.

Upon further examination, we noted that the same sequence of activities was used regardless of the baseline that was created (e.g., functional, allocated, developmental, or product), with one exception. Integration and test phases also required the construction of a testable configuration in addition to the creation of a baseline. Therefore we created one set of configuration identification processes for non-integration phases, and another for those involving integration.

Parenthetically, we chose not to describe and teach these two sets of configuration identification processes in our CM courses. Instead, we taught a process sequence for each baseline that was instantiated from one of the two sets, in order to reduce the amount of work to construct a usable process sequence for a given process.

The process variations that existed within PRC pertained to implementation details, like development platforms and CM tool kits. In order to maintain and control complexity, the configuration identification functions were abstracted to remove dependencies on these implementation details. Figure 5-1 lists the reusable configuration identification process definitions that resulted from the process design step, arranged hierarchically.

Figure 5-2 is the process definition for the configuration identification macro process. Note that in the process flowchart (Part 4), ILS stands for Integrated

Logistics Support, which addresses the managerial and technical aspects of a system's operational support.

No process automation was attempted for the corporate configuration identification processes.

*Process Knowledge Acquisition Methods:* Knowledge of the configuration identification processes was acquired through a variety of avenues. Interviews with CM representatives from projects undergoing software process improvement efforts provided most of the project specific information. These representatives provided project-specific processes, textual process descriptions, checklists, and plans. After draft reusable process descriptions were created, the representatives were again interviewed to provide feedback on the process descriptions, and to determine the appropriateness of the level of detail, the correctness of the processes, and their utility to their projects. This response was used to update and strengthen the descriptions. The configuration identification reusable processes were then distributed for review.

*Process Testing:* The configuration identification processes that were a result of the domain analysis were tested through reviews. The initial set of processes were first presented to each of the configuration managers on our largest projects for their review. They were also given to team members for review. Finally, they were taught to a pilot class, again for review, before presenting the material in a formal PRC training course.

*Process Tailoring Knowledge Acquisition Methods:* In conjunction with the creation of the reusable process descriptions was the creation of a training course for projects who were to reuse these processes. The training was given to configuration management experts who were to tailor the reusable processes to their projects, and then to train their project staff on their use. Through course exercises and homework, these experts both tailored the processes and provided feedback on the tailoring methodology.

The training course for tailoring reusable processes was given six times over a three month span of time. The feedback from these course offerings proved to be invaluable to validating and improving the process tailoring methodology. The next subsection and also Chapter 6 describe the measurements of the tailoring process.

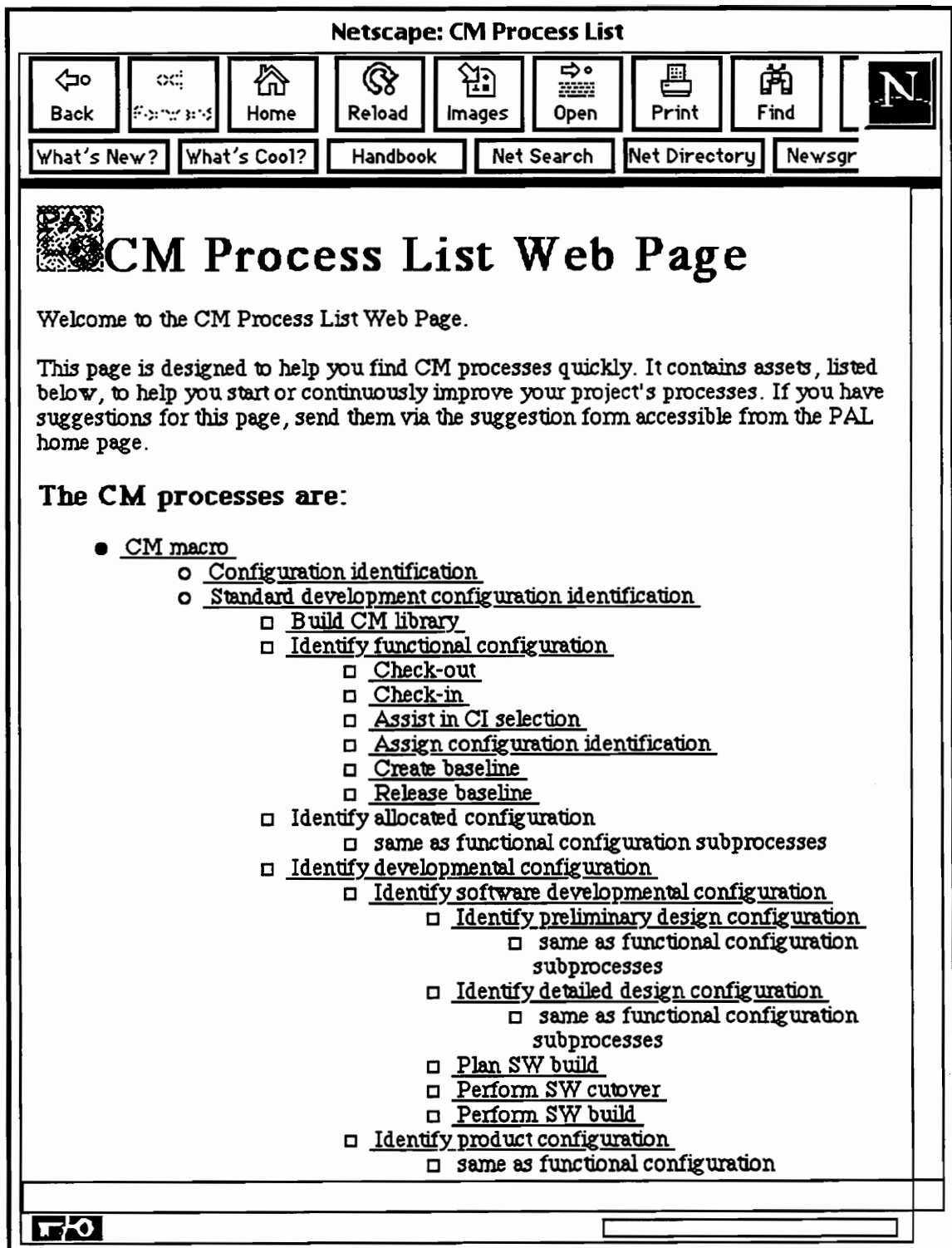


Figure 5-1. Configuration Identification Process Definitions

<b>Name</b>	CM200 - Configuration Identification
<b>GENERAL INFORMATION</b>	
<b>Process ID</b>	CM200
<b>Process Purpose</b>	To incrementally establish and maintain a definitive basis for control and status accounting for a CI throughout its life cycle.
<b>Standards</b>	CMM v1.1 MIL-STD-973 IEEE 828-1990, 1042-1987
<b>Related Processes</b>	CM100 Configuration Management Planning CM300 Configuration Control CM400 Configuration Status Accounting CM500 Configuration Audits
<b>Version Number</b>	v2.1
<b>CUSTOMER DESCRIPTION</b>	
<b>Customer</b>	External client Project management
<b>Requirements</b>	CMM v1.1 CM.AC.04.* - The software work products to be placed under configuration management are identified.  CMM v1.1 CM.AC.07.* - Products from the software baseline library are created and their release is controlled according to a documented procedure.  CMM v1.1 CM.AC.08.* (or CSA) - The status of configuration items/units is recorded according to a documented procedure.  MIL-STD-973 24-Nov-93 5.3.* - Configuration identification.  ANSI/IEEE Std 828-1990 2.3.1.* - Configuration identification.  ANSI/IEEE Std 1042-1987 3.3.* - Configuration identification.
<b>INTERFACE DESCRIPTION</b>	
<b>Entrance Criteria</b>	The contract is awarded to PRC or management approves the commencement of a development effort.

**Figure 5-2. CM200 Configuration Identification Process - Part 1**

<b>Inputs</b>	Contract SOW Proposal
<b>Outputs</b>	Functional baseline Allocated baseline Product baseline(s) Internal baselines, one per PRC SLC stage CM Library System
<b>Exit Criteria</b>	The project is completed.
<b>PROCEDURAL DESCRIPTION</b>	
<b>Responsibilities</b>	Roles that participate in the process are described at lower levels of process decomposition. Overall responsibility lies with the group tasked with the CM function.
<b>Tasks</b>	<p>The tasks that must be accomplished during this process are illustrated in Figure 5.2 - Configuration Identification Process Flowchart and described below.</p> <p>CM210 Build CM Library  CM220 Identify Functional Configuration  CM230 Identify Allocated Configuration  CM240 Identify Software Developmental Configuration  CM250 Identify Product Configuration</p>
<b>Tools</b>	<p>Tools that are used in configuration identification usually include a Database Management System to house the configuration identification record system and the configuration control (i.e., change request) record system, a version control system for checkouts and checkins, and build utilities (e.g., the UNIX 'make' command). Often, larger more 'full-featured' CM systems are bought to automate more of the CM processes.</p> <p>It should be noted though that the CM functions to be performed on the project must be defined before beginning a tool selection process. Knowing what you want to do before acquiring the method of how to do it will provide the right level of requirements for the trade analysis that will ensue.</p>
<b>Resources</b>	TBD

**Figure 5-2. CM200 Configuration Identification Process - Part 2**



<b>MEASUREMENT DESCRIPTION</b>	
<b>Quality Indicators</b>	<p>The quality indicators are:</p> <p>Q1 - Hours expended in CM activities (planned and actual)</p> <p>Q2 - Status of scheduled CM activities (planned and actual)</p> <p>Q3 - Cost expended in CM activities (planned and actual)</p>
<b>Process Indicators</b>	<p>Process indicators tracked per subprocess are</p> <p>P1 - Hours expended in CM activities (planned and actual)</p> <p>P2 - Status of scheduled CM activities (planned and actual)</p> <p>P3 - Cost expended in CM activities (planned and actual)</p>

**Figure 5-2. CM200 Configuration Identification Process - Part 3**

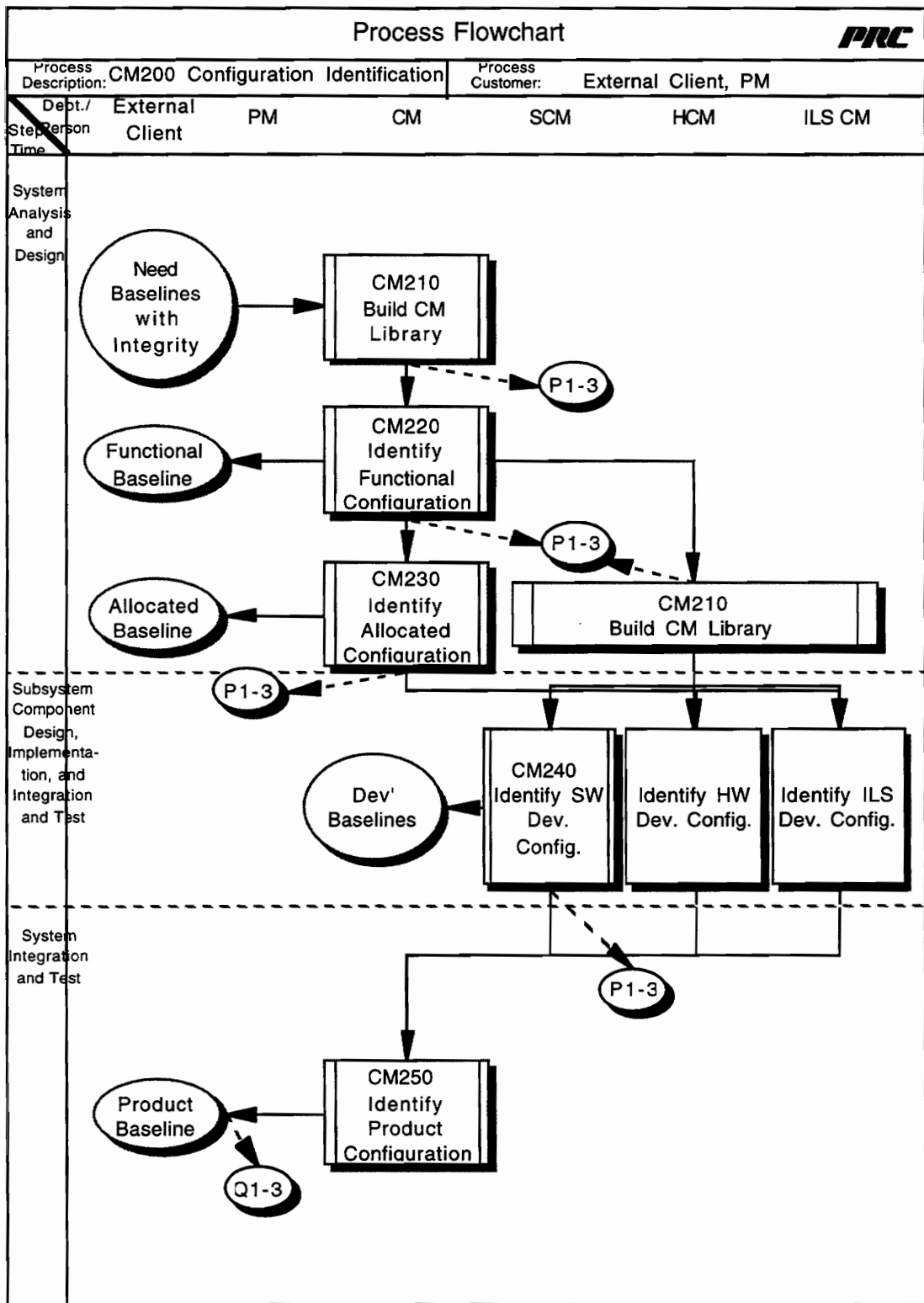


Figure 5-2. CM200 Configuration Identification Process - Part 4

## 5.2 Tailoring of a Reusable Peer Review Process

This section describes the tailoring of a peer review process by a project in one of PRC's business units. This instantiation took just over two hours by a team of two business unit personnel, one hour to tailor the textual process description and another to tailor the process flowchart.

The project wanted to conduct peer reviews on resolutions to problem reports and change requests, but they wanted to perform the peer reviews with the least number of people in the shortest amount of time while still maintaining the integrity of the process. The rest of this section describes how the process was tailored using the steps of process tailoring methodology. Figure 5-3 shows the corporate process and Figure 5-4 shows the result, the project's tailored process. Note that the process notation in Figures 5-3 and 5-4 does not contain the context section, as defined in Section 4.2, which was added after the tailoring session occurred.

*Plan Process Tailoring Project/Select Corporate Process:* The tailoring was done as part of a process tailoring workshop so a full project plan was not created. The corporate process was retrieved from the PRC Process Asset Library described in Section 4.6.

*Tailor Customer and Requirements Information:* The project added their name to the process ID and modified the purpose. The customer was the project's management. The requirements were made specific to the project: understand the technical quality of the resolution and locate defects in code before testing.

*Tailor Interface Information and the Quality Indicators:* The project modified the interface information to reflect the way the project wanted to run reviews. The manager started the process when the resolution was ready to be reviewed and a moderator was assigned. The project plan was the input and a completed checklist took the place of the corporate peer review report. The exit criteria included completing the checklists and assignment

worksheets, and placing the checklist in the appropriate software development folder. One quality indicator was added for schedule slippage.

*Tailor Procedural Information and Process Indicators:* The major modifications were to reduce the number of participants to three: the moderator/reviewer, the author, and a reviewer. A few of the tasks were eliminated; the original numbers were maintained to show the deletions. The process flowchart was also modified. Instead of referring to lower level processes, the major characteristics of each step were added to the flowchart. Two process indicators were eliminated.

*Automate Process/Integrate & Test Process/Get SEPG Approval/Store in PAL:* The project decided against automation. Static testing was accomplished through a peer review. The local SEPG approved the tailored process and submitted it to the PAL for later retrieval.

<b>Name</b>	PR200 Conduct a Peer Review
<b>GENERAL INFORMATION</b>	
<b>Process ID</b>	PR200
<b>Process Purpose</b>	"The purpose of Peer Reviews is to remove defects from the software work products early and efficiently. An important corollary effect is to develop a better understanding of the software work products and of defects that might be prevented. "- Software Engineering Institute's (SEI) Capability Maturity Model (CMM)
<b>Standards</b>	CMM, v1.1, Level 3 Peer Review Key Process Area
<b>Related Processes</b>	CM300 - Configuration Control RM000 - Requirements Management PR100 - Setting Up a Peer Review Program
<b>Version Number</b>	v1.0
<b>CUSTOMER DESCRIPTION</b>	
<b>Customer</b>	Project Management
<b>Requirements</b>	Understand the technical quality of a given work product. Locate major defects as early as possible.
<b>INTERFACE DESCRIPTION</b>	
<b>Entrance Criteria</b>	PR100 - Setting Up a Peer Review Program
<b>Inputs</b>	Peer Review Strategic Plan
<b>Outputs</b>	Peer Review Report Peer Review Metrics Peer Review Action Items Revised Work Product (if peer review disposition is "conditionally approve")
<b>Exit Criteria</b>	Peer review is conducted; peer review report is completed and distributed; action items are added to the action item database; and if the peer review disposition is "conditionally approve," the work product is revised and approved.

**Figure 5-3. Corporate Peer Review Process - Part 1**

PROCEDURAL DESCRIPTION		
<b>Responsibilities</b>	<b>Moderator:</b>	Responsible person for setting up the schedule, getting the packages out on time, assigning specific review tasks to Reviewers, leading the discussion, keeping the process moving, cutting off digression, and reporting the results. This role requires special training. The Moderator does not have to be a member of the project technical staff, and should not be the manager of any other participant.
	<b>Author:</b>	The primary creator of the product being reviewed.
	<b>Reviewers:</b>	2-4 technical people whose job includes creating this type of product. These people should be familiar with the project, but don't have to be full time project staff members.
	<b>Recorder:</b>	Takes notes during the review meeting. This person must possess enough technical understanding of the issues to record the discussions. One of the reviewers may be asked to be Recorder or they could all take turns.
<b>Tasks</b>	The following tasks are accomplished during this process:	
(Moderator)	1.	Moderator selects Recorder and Reviewers, and schedules the peer review. (PE020)
(Author)	2.	The work product to be reviewed is completed and given to the Moderator. (PE038)
(Moderator)	3.	The Moderator ensures that the work product meets readiness criteria. (PE021)
(Moderator & Reviewers)	4.	The Moderator distributes peer review materials and checklists to the Reviewers and assigns them roles. (PE021)
(Reviewers)	5.	The Reviewers review the work product and return a list of potential defects to the Moderator. (PE022)
(Moderator & Author)	6.	The Moderator ensures that sufficient review has occurred and distributes defect lists to Author. (PE023)
(All)	7.	The Moderator conducts the peer review and issues minutes from the peer review. (PE023)
(Moderator)	8.	The Moderator enters the noted defects into the problem tracking system. (PE024)
(Author)	9.	The Author reworks the product to remove the noted defects, if necessary. (PE025)

**Figure 5-3. Corporate Peer Review Process - Part 2**

(Moderator)	10. The Moderator verifies that the noted defects are resolved. If so, the Moderator issues a resolution memo. (PE026)
(Moderator)	11. If the product requires major rework, the Moderator schedules another peer review (exit this peer review process and initiate another). (PE026)
(Moderator)	12. The Moderator prepares review metrics. (PE026)
(Moderator)	13. The Moderator ensures that all peer review exit criteria are satisfied. (PE026)
<b>Tools</b>	Action item database
<b>Resources</b>	Action item database
<b>MEASUREMENT DESCRIPTION</b>	
<b>Quality Indicators</b>	Q1 - Total number of defects per KSLOC/docpage Q2 - Total elapsed time to complete peer review process
<b>Process Indicators</b>	P3 - Number of potential defects per KSLOC or docpage P4 - Number of preparation hours per reviewer P5 - Number of actual defects per KSLOC or docpage  P1 - Elapsed time to plan review P2 - Elapsed time to prepare for review P6 - Elapsed time to conduct review P7 - Elapsed time to record review results P8 - Elapsed time to conduct follow-up

**Figure 5-3. Corporate Peer Review Process - Part 3**

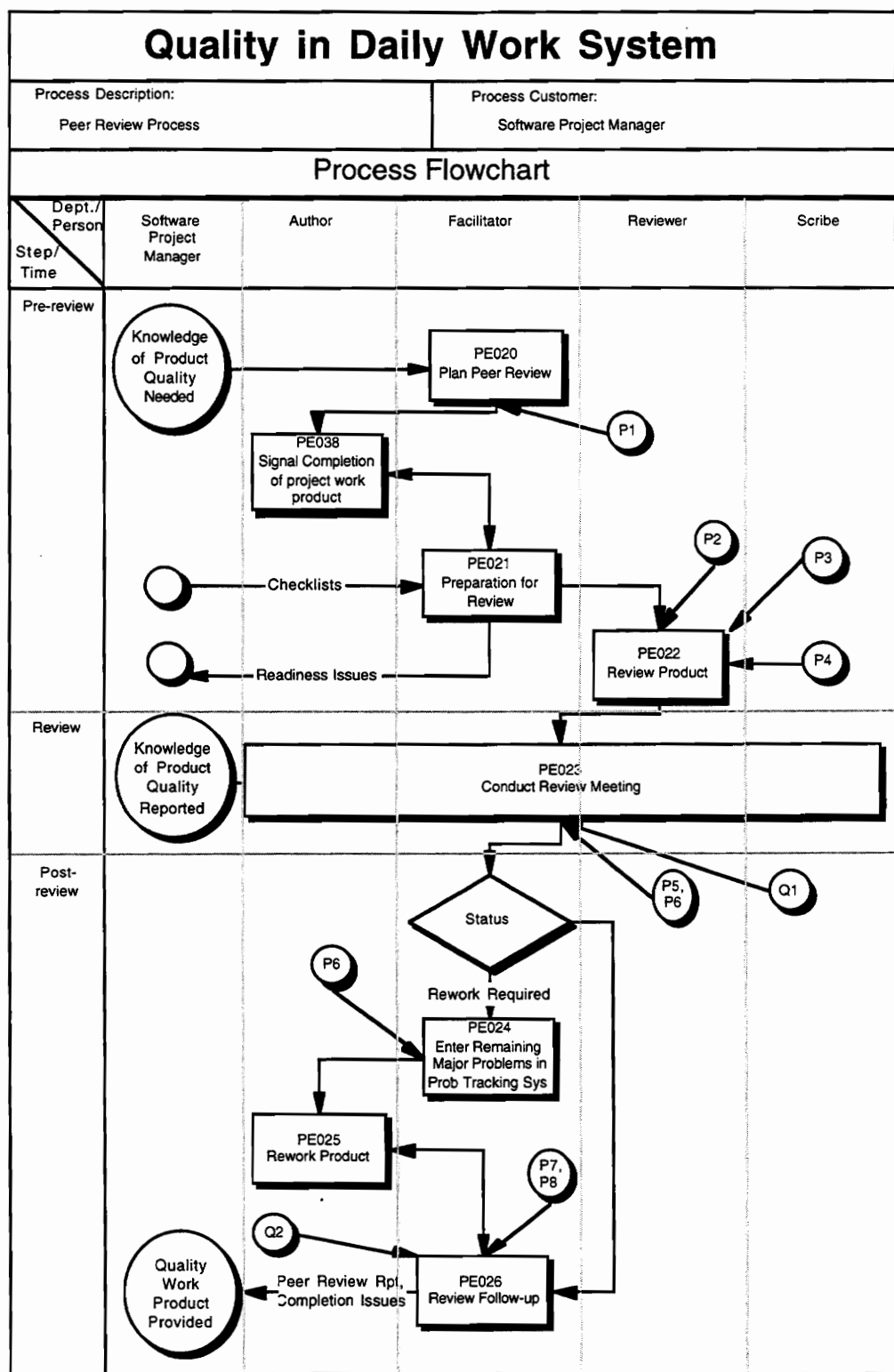


Figure 5-3. Corporate Peer Review Process - Part 4



<b>Name</b>	<Project> - PR200 Conduct a Peer Review
<b>GENERAL INFORMATION</b>	
<b>Process ID</b>	<Project> PR200
<b>Process Purpose</b>	The purpose of the modified peer review process is to conduct peer reviews of PR/CR resolutions with the least number of people in the shortest amount of time while still maintaining the formality of the process.
<b>Standards</b>	CMM, v1.1, Level 3 Peer Review Key Process Area
<b>Related Processes</b>	CM300 - Configuration Control RM000 - Requirements Management PR100 - Setting Up a Peer Review Program
<b>Version Number</b>	v1.0
<b>CUSTOMER DESCRIPTION</b>	
<b>Customer</b>	<Project> Management
<b>Requirements</b>	Understand the technical quality of a PR/CR (Problem Report/Change Request) resolution.  Locate defects in code before testing.
<b>INTERFACE DESCRIPTION</b>	
<b>Entrance Criteria</b>	Software development manager assigns someone as a technical reviewer of a PR/CR resolution.  PR/CR resolution is at a stage where it can be reviewed.
<b>Inputs</b>	Detailed Project Plan (describes schedule for PR/CR completions via the software development manager's work assignment worksheet)  PR/CR resolution

**Figure 5-4. Tailored Peer Review Process - Part 1**

<b>Outputs</b>	<p>Completed checklists (takes place of peer review report)</p> <p>Completed software development manager's work assignment worksheet (signed off to show completion)</p> <p>Peer Review Metrics</p> <p>Peer Review Action Items</p> <p>Revised PR/CR resolution (if peer review disposition is "conditionally approved")</p>
<b>Exit Criteria</b>	<p>Peer review is conducted; checklists are completed and filed in the appropriate SDF; action items are added to the problem tracking database; if the peer review disposition is "conditionally approved," the work product is revised and approved; and the software development manager's work assignment worksheet is signed off by QA.</p>
<b>PROCEDURAL DESCRIPTION</b>	
<b>Responsibilities</b>	<p><b>Moderator:</b> Responsible person for getting the PR/CR resolution from the author, distributing it to the other reviewer, reviewing the product, consolidating noted defects, and completing his checklist. This role requires training. The Moderator does not have to be a member of the project technical staff, and should not be the manager of any other participant.</p>
	<p><b>Author:</b> The primary creator of the product being reviewed.</p>
	<p><b>Reviewer:</b> A technical person who is familiar with the project, but doesn't have to be a full time project staff member. The reviewer is responsible for completing his checklist and returning it to the moderator.</p>
<b>Tasks</b>	<p>The following tasks are accomplished during this process:</p>
(Moderator)	<p>2. The moderator gets the PR/CR resolution from the author at the time when it is scheduled for review. (PE038)</p>
(Moderator)	<p>3. The Moderator ensures that the PR/CR resolution meets readiness criteria. (PE021)</p>

**Figure 5-4. Tailored Peer Review Process - Part 2**

(Moderator)	4.	The Moderator distributes PR/CR resolution and associated checklist to the other Reviewer and specifies a return date. (PE021)
(Moderator & Reviewer)	5.	The Moderator and the Reviewer review the work product and returns to the Moderator the checklist, which includes potential defects. (PE022)
(Moderator)	6.	The Moderator consolidates the two checklists and gives the consolidated checklist to the Author, and resolves any disagreements or discrepancies. (PE023)
(Moderator)	8.	The Moderator enters the noted defects into the problem tracking system and action items in the action item database. (PE024)
(Author)	9.	The Author reworks the product to remove the noted defects, if necessary. (PE025)
(Moderator)	10.	The Moderator verifies that the noted defects are resolved. (PE026)
(Moderator)	11.	If the product requires major rework, the Moderator schedules another peer review (exit this peer review process and initiate another). (PE026)
(Moderator)	12.	The Moderator prepares review metrics. (PE026)
(Moderator)	13.	The Moderator ensures that all peer review exit criteria are satisfied. If so, the Moderator signs the checklist as completed. (PE026)
<b>Tools</b>		Problem tracking database Action item database
<b>Resources</b>		Problem tracking database Action item database
<b>MEASUREMENT DESCRIPTION</b>		
<b>Quality Indicators</b>	Q1 - Total number of defects per KSLOC/docpage	
	Q2 - Total effort to complete peer review process	
	Q3 - Total amount of schedule slippage	
<b>Process Indicators</b>	P4 - Number of preparation hours per reviewer	
	P5 - Number of actual defects per KSLOC/docpage	
	P1 - Effort to complete software development manager's work assignment worksheet	
	P6- Effort to consolidate and resolve checklists	
	P7 - Effort to record review results	
	P8 - Effort to conduct follow-up	

**Figure 5-4. Tailored Peer Review Process - Part 3**

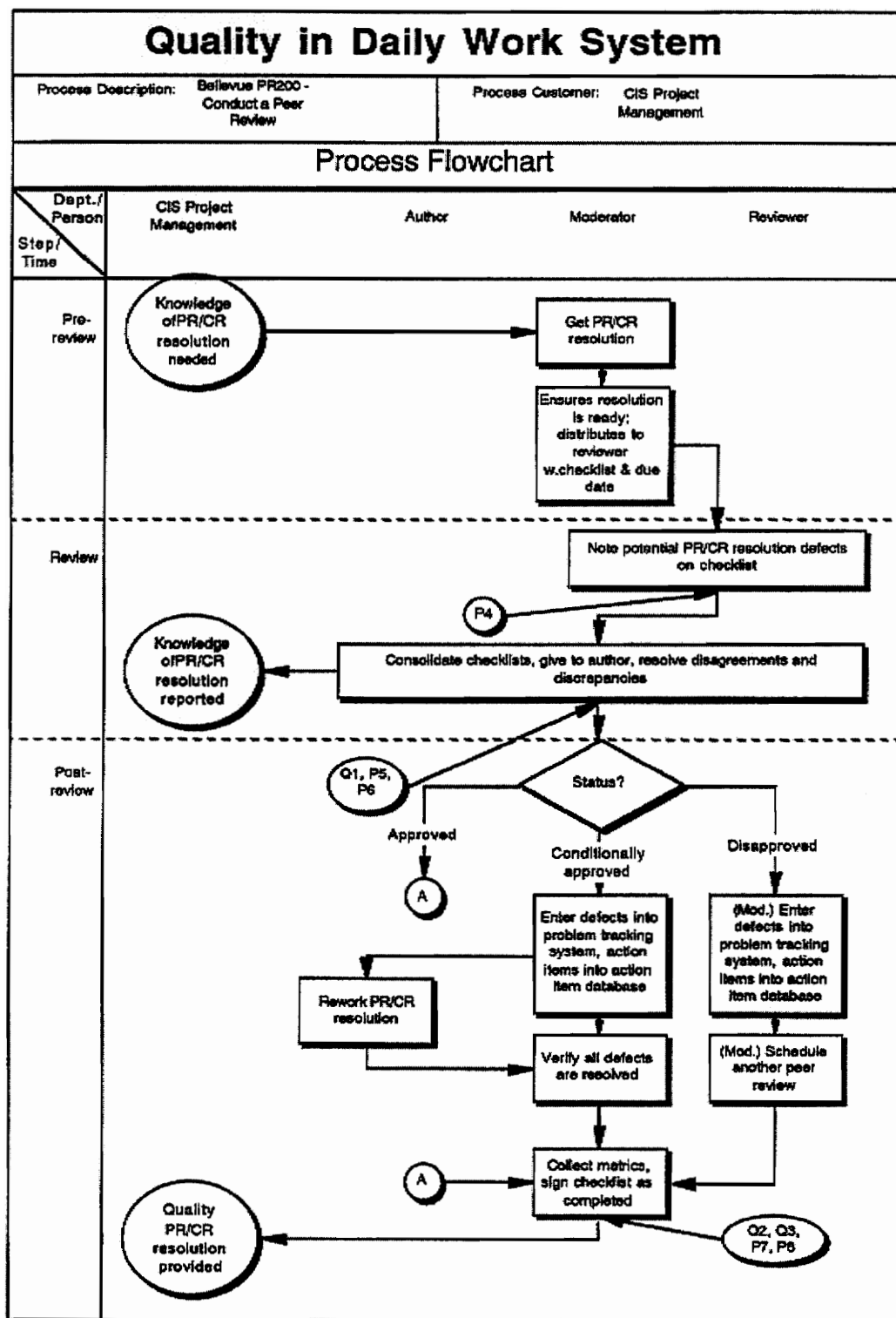


Figure 5-4. Tailored Peer Review Process - Part 4

## **6. Case Study**

This section describes a case study conducted at PRC from November 1994 to July 1995 to analyze, measure, and increase software process reuse. The study had several parts: first, organizational processes were redefined using domain analysis techniques; second, the organizational processes were taught to the company; third, a simple methodology for process tailoring was taught at each organizational process training class; fourth, the organizational processes were saved in a web-browsable process asset library; and finally, projects applied the process tailoring method while reusing organizational processes, sometimes aided by corporate consultation.

A working group within PRC composed of representatives from all line organizations as well as full-time SPI staff performed most of these countermeasures. The PRC SEPG (Software Engineering Process Group) formed this team to create corporate training for level 2 key process areas and to modify or redefine processes so that they were compliant with version 1.1 of the Capability Maturity Model for Software. They used the process definition methodology defined in Section 4.4 to collect project-specific processes and data and to create reusable processes. Release 2.0 of the corporate process descriptions, numbering about 120 processes, was the result of their efforts. These processes were stored in a web-browsable Process Asset Library (PAL) which is accessible by PRC employees. Specific KPA web pages were built to house the reusable processes, project-specific instantiations, related process assets, and training material. The team gave the training associated with these processes first in pilot and then in corporate training situations. In each of these, the students were taught how to tailor the corporate/reusable processes to meet their project-specific needs, using the process tailoring methodology defined in Section 4.5. Finally, instances of process definition and/or reuse over the 1991 to 1995 time period were collected.

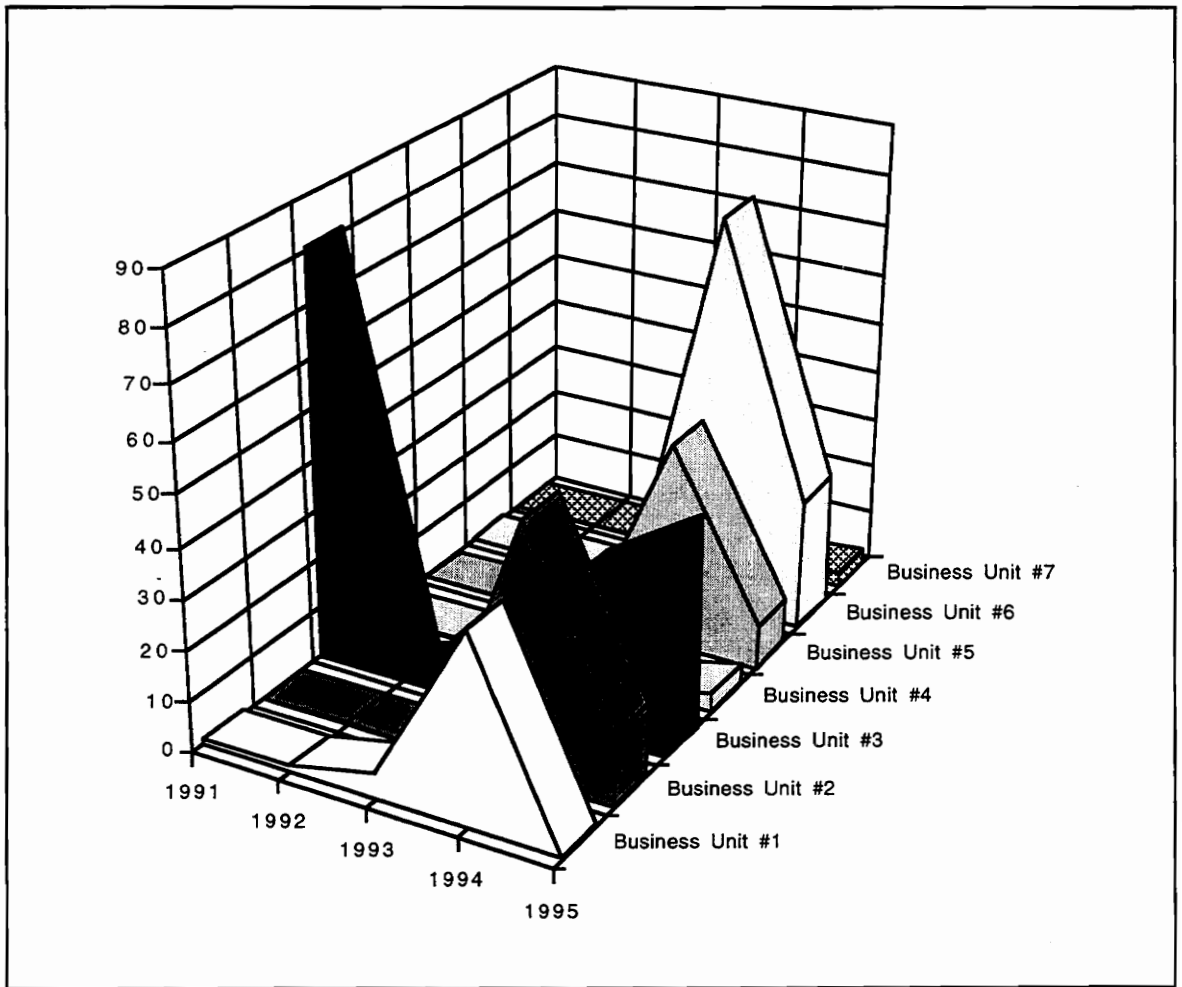
The following are results of the case study. First, the process definition activities of the business units under study are examined between 1991 and

1995. Next, reuse levels are compared between 1994 and 1995 to examine the result of the process definition and tailoring methods. Third, the benefits of process tailoring are examined. Fourth, reuse levels are compared between key process areas. Finally, other findings are summarized.

*Process Definition Activities Between 1991 and 1995:* The study of process reuse at PRC showed that seven business units had expended significant investments in process definition and reuse. 438 specific instances of process definitions were uncovered, spanning a five year time period. Figure 6-1 shows the processes defined by the seven business units by year. Note that several business units are increasing their process definition activity, shown by the positive slope of their lines, while other, sometimes more mature, organizations are in a period of less process definition activity, noted by their negatively sloping lines. These periods of activity and inactivity are normal and in turn affect the number of software process reuse opportunities per year. During 1994, projects within these business units were typically working to complete processes associated with level 2 key process areas; in 1995 their focus generally turned to level 2 issues other than process definition. Since this case study was conducted in only the first eight months of 1995, the number of instances with potential for process reuse was noticeably less than in 1994.

In Figure 6-1, note that Business Unit #3 has an inordinately large number of processes defined during 1991. This business unit participated fully with the PRC Technology Center in the development of PRC's corporate processes, and provided projects on which the corporate processes were defined. Its 1991 figure reflects the adoption of those processes.

*Reuse Levels Compared:* Has use of the method increased process reuse? To answer this question, we compared the process reuse percentage in 1994 before the method was used and in 1995 after the method was introduced. As Table 6-1 shows, process reuse increased from 41% in 1994 to 55% in 1995. We believe that the domain analysis and process tailoring methods and training contributed to this increase. All business units participated in the training, and business units #4 and #7 have set a standard to begin all process



**Figure 6-1. Defined Processes by Business Unit by Year**

definition efforts by identifying reusable processes. More time is needed to measure the effects of the domain analysis and process tailoring methods. The decrease in total processes from 1994 to 1995 is due to incomplete data from 1995 (through July only), and normal fluctuations in the process creation and use cycle.

**Table 6-1. Process Reuse in 1994-1995 by Business Unit**

Business Unit	1 9 9 4			1995 (to 8/1)		
	Reused	Total	% age	Reused	Total	% age
Business Unit #1	0	38	0.00	0	0	0.00
Business Unit #2	0	50	0.00	1	14	0.07
Business Unit #3	26	31	0.84	37	42	0.88
Business Unit #4	0	0	0.00	3	4	0.75
Business Unit #5	35	41	0.85	0	9	0.00
Business Unit #6	37	79	0.47	10	26	0.38
Business Unit #7	0	1	0.00	3	3	1.00
Total	98	240	0.41	54	98	0.55

*Benefits of Process Tailoring Method:* Significantly, the time to develop a project specific process was dramatically lower through reuse than through project process definition efforts, showing at least a 10 to 1 increase in efficiency. Metrics from both corporate and project process definition efforts place the effort spent to design processes for a KPA at around 800 to 1000 or more person-hours, depending on the breadth and depth of the KPA. One project process team serves as an example: this team built four processes in 800 hours (1 process in 200 hours), their efforts spanning over one full year. Results from process tailoring show that a project process can be instantiated in two hours, one to tailor the process textual description, and one to tailor the graphical description of process roles and tasks. When a peer review of the process is added, the total time to instantiate and produce a project-specific process is around 20 hours. Thus, conservatively speaking, our PRC experience shows a 10 to 1 improvement in time to define a project-specific process.

These results should be tempered with a few caveats. First, these figures represent initial data; a larger body of metrics is needed to draw conclusive findings. Secondly, the effect of training must also be noted. Those groups that tailored processes received training in the process area while earlier



groups did not; the training prepared them to be ready to tailor. At the same time, both groups had organizational processes at their disposal; those groups that chose to reuse them were able to significantly reduce the amount of information they had to learn and master.

These preliminary results are significant in that two strong process improvement requirements voiced by PRC projects are 1) time to improve, the quicker the better, and 2) effort to improve, since most improvement efforts are performed on tight, limited budgets.

*Process Reuse by Key Process Area:* Process reuse levels varied among the process domains used at PRC, the key process areas from the Capability Maturity Model (CMM) for Software. For each key process area in levels two and three, figures 6-2 and 6-3 show the number of processes defined and number of processes reused in each of four categories, respectively. The categories are: 1) "instantiated without modification" which represents the highest possible reuse, 2) "instantiated and modified," but the project process still shows a clear derivation from the organizational process, 3) "reference only; no clear derivation" which means that the projects used the organizational process as one of many sources but the resulting project process did not show a strong relationship to it, and 4) "no reuse," that is, the organizational process was not reused at all.

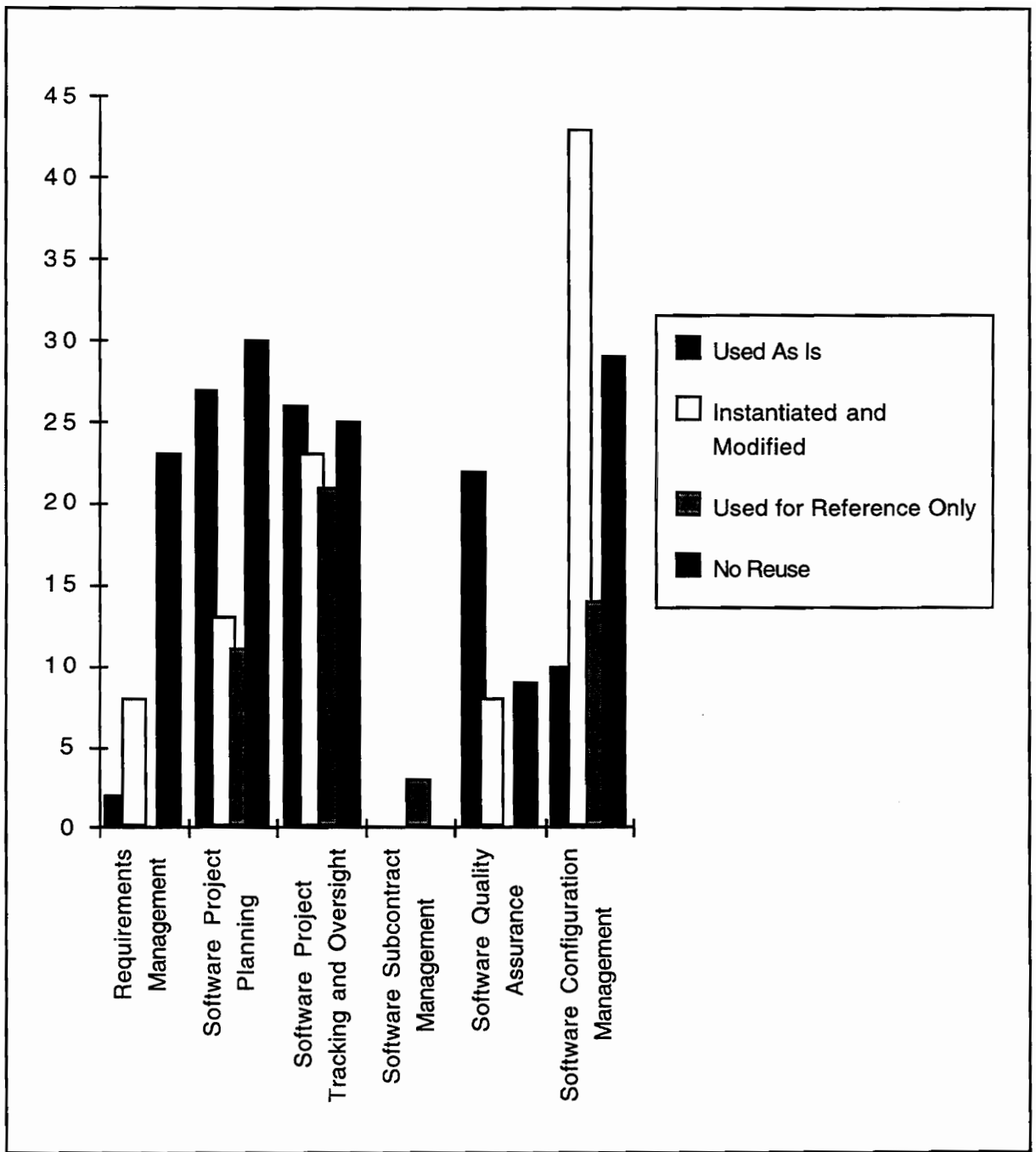
First, the number of processes varies widely between key process areas (KPAs), from three for software subcontract management (SM) to over 90 for both software project tracking and oversight (PT) and software configuration management (CM). There are two reasons for the low number of SM processes: first, PRC normally does not work in an environment where this key process area is applicable, and second, for those few organizations that do require subcontractors that fit the CMM criteria, it is easier to tackle improvements in other areas where interaction with non-software personnel is not required. The number of requirements management processes is low primarily because the relative simplicity of the KPA and the domain interpretation used by PRC in its corporate RM processes. PRC divides requirements analysis into two parts: a software product engineering (PE)

KPA part devoted to the elicitation, analysis, and approval of requirements, and a requirements management part devoted to managing changes to the established requirement set. The number of software quality assurance processes is also relatively low again due to its relative lack of complexity. In PRC, most quality assurance functions are viewed as the same regardless of the work product or process being examined.

The amount of process definition by business units at level 3 varies greatly, as shown in Figure 6-3. Processes in the organization process focus and organization process definition KPAs are low in that most business units rely solely on the PRC corporate processes and therefore do not see any need to redefine or instantiate these processes for their business unit. Software product engineering (PE) shows high levels of process definition. Since during most of the time period covered by this case study there were no corporate PE processes defined, business units defined their own, knowing that by defining processes they are more able to manage the software development process itself. For the most part, the level 3 KPA, intergroup coordination, has not received much attention, in that most business units are working at attaining compliance at level 2. Finally, the peer review KPA shows fairly high levels of process definition. The peer review KPA is an easy KPA to implement; it is small, self-contained, and does not require massive organizational changes. For these reasons, the peer review KPA is often implemented first, knowing that an early process improvement success aids in the implementation of other improvement efforts. More will be said about this KPA when amount of reuse is discussed.

The amount of reuse in the level 2 and 3 KPAs also varies markedly. Figures 6-2 and 6-3 again are used to illustrate the type of reuse, either instantiated, instantiated and modified, reference only, or none.

A few comments are in order concerning the level 2 KPAs. In the RM KPA, the level of reuse reflects a significant disagreement regarding domain boundaries as mentioned earlier. Business units seem to define their own processes rather than reuse corporate processes. This situation provides an opportunity to redefine the RM corporate processes, using domain analysis



**Figure 6-2. Process Definitions in Level 2 Key Process Areas by Reuse Type**

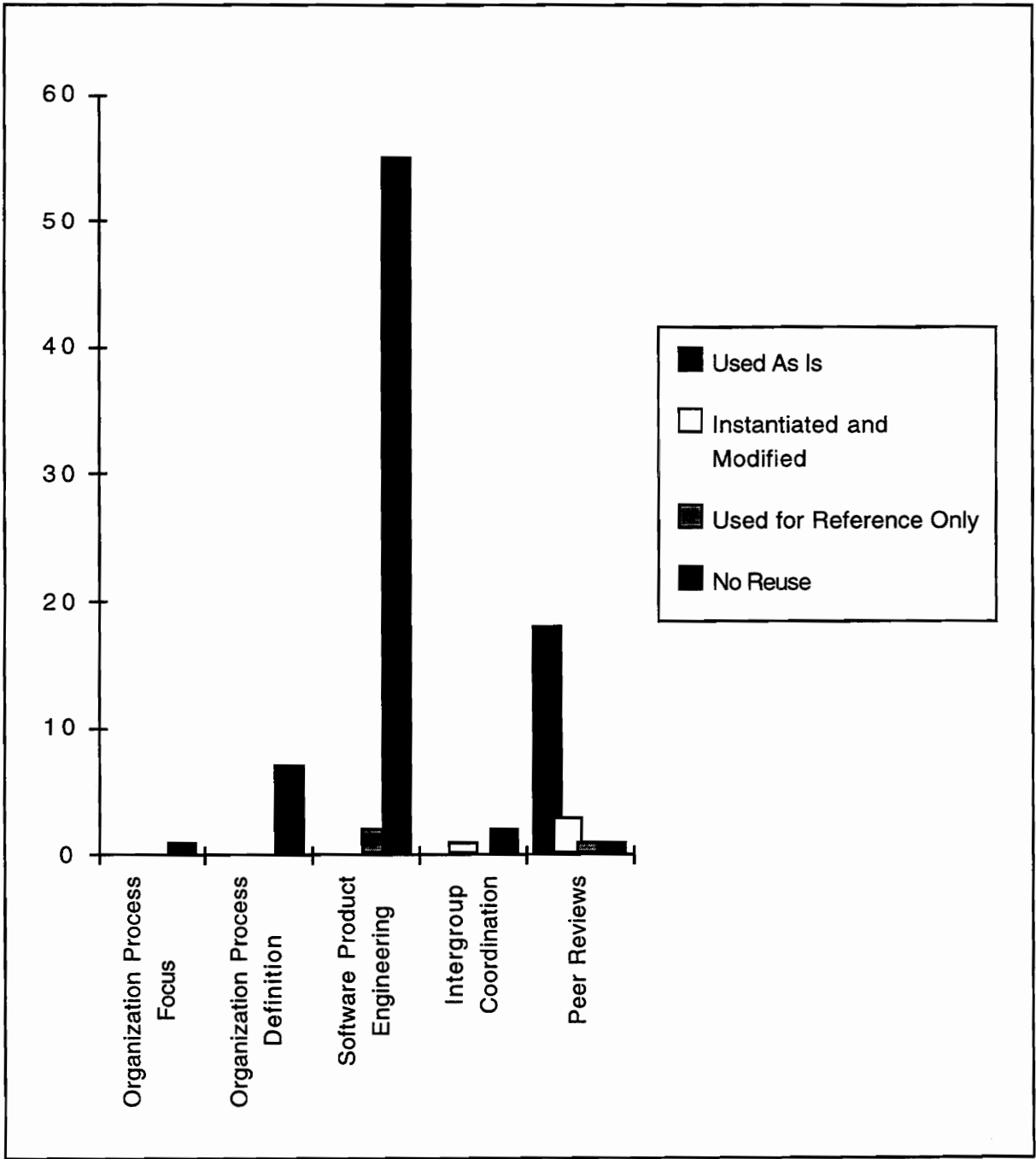


Figure 6-3. Process Definitions in Level 3 Key Process Areas by Reuse Type

techniques and employing greater involvement and review with business unit personnel. In the QA KPA, the level of instantiated and instantiated & modified reuse is remarkably high, pointing to the rather ubiquitous and highly transferable nature of the KPA processes. At first sight, the levels of reuse within the CM KPA are surprisingly low. The number of instantiated processes is about 10 whereas other level 2 KPAs number between 20 and 30. Yet the number of instantiated and modified processes is larger than any other KPA, measuring about 40 processes. Upon further examination, the reason for this is clear: the implementation of CM processes is severely affected by project platforms and toolsets. The corporate CM processes abstract beyond this platform level, requiring modification to make the corporate process usable at the business level.

The peer review KPA distinguishes itself in regard to level of reuse. Roughly 80 percent of the business unit process definition efforts reuse the corporate process entirely by simply instantiating it for their organizational unit. Thus the peer review process domain is ideal for process reuse and remains the goal for other process domains.

*Other Case Study Findings:* The confidence that the business unit has in the process definition significantly influenced the adoption of processes. Figure 6-4 shows that business units #1 and #2 have virtually no instances of process reuse, yet as Figure 6-1 illustrates, these business units have defined significant numbers of processes. It is interesting that these two business units have no working process definition relationships, past or present, with the PRC corporate process improvement staff. Business units #3 through #7 on the other hand either have developed processes with the corporate staff (business units #3 and #6), are currently working together with the corporate staff (business unit #4), or have made a conscious decision to reuse corporate process assets whenever possible (business unit #7). Business unit #6 exhibits confidence in the processes of business units #1 and #2 from which a significant number of staff came, and has therefore reused many of their processes, further strengthening the link between process reuse and confidence in the process itself.

Figure 6-4 further highlights the relationship between process producer and process reuser by showing number of processes reused. Those business units whose personnel were most active in producing processes in previous situation were also the most active in reusing those same processes. Note the reuse levels of business unit #3, #5, and #6, all with personnel actively involved in previous process production.

Another interesting portion of the study dealt with our process tailoring experience with several business units and KPAs. Processes have been instantiated and modified in the following KPAs: peer reviews, requirements management, software project tracking and oversight, software quality assurance, and intergroup coordination. In each of these cases, an "instantiated" or "instantiated and modified" process was created in two to three hours. Participants in this process are usually excited about its power, surprised at the results, and eager to apply the same method to other processes.

One final item regarding our findings was that reuse of process within PRC was not limited to reuse of corporate processes. One business unit in particular has taken advantage of processes from another business unit and from a tool vendor. In the first case, no particular tailoring was required; the processes that involved cost planning and tracking were reused without modification. In the case of the process from the tool vendor, the business unit used the process and tool documentation as a reference for process definition, but were not able to instantiate or instantiate and modify the tool's process information.

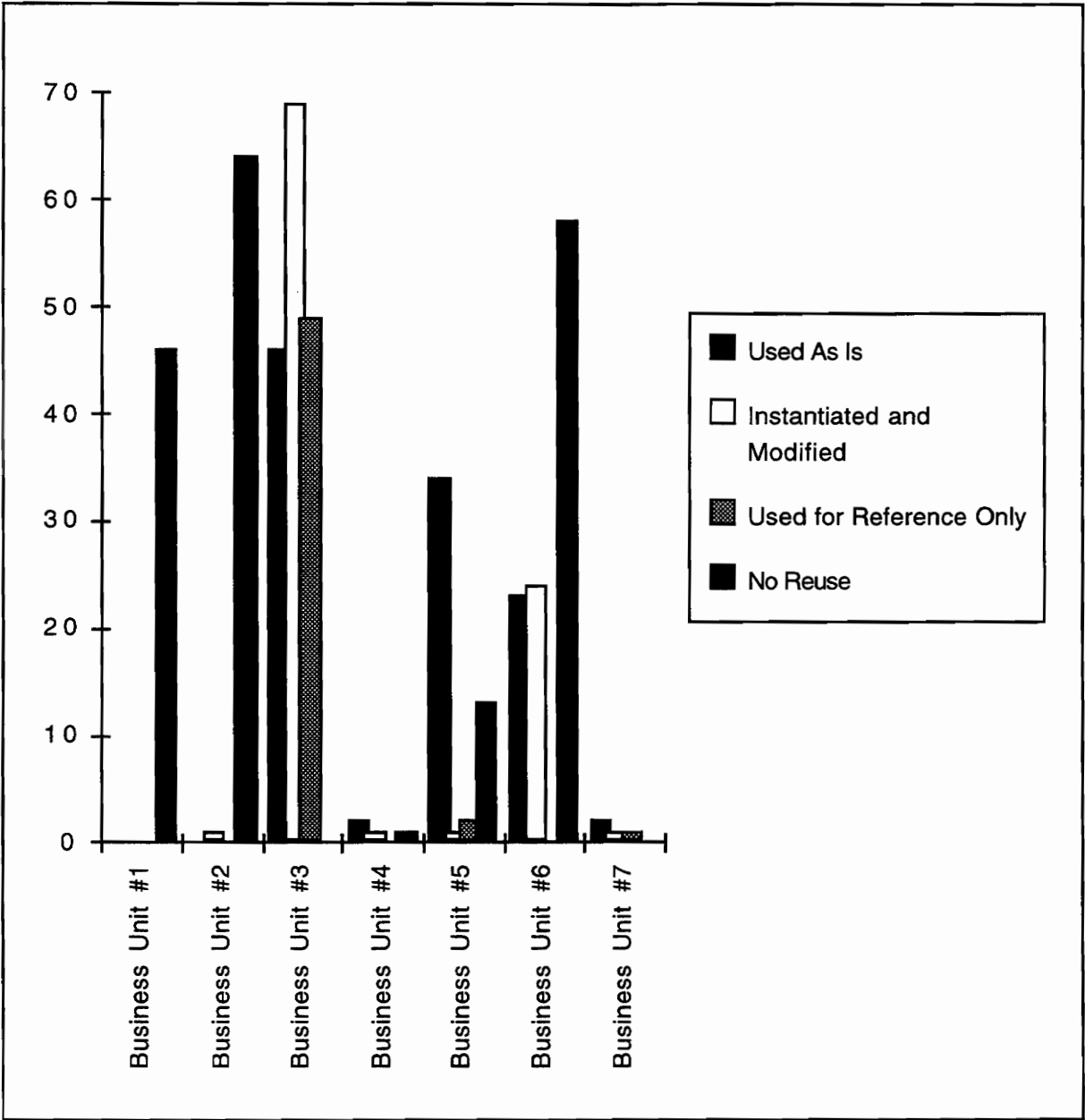


Figure 6-4. Percentage of Process Reuse by Business Unit

## **7. Conclusions**

### **7.1 Conclusions**

This thesis shows that principles garnered from the domain analysis and software reuse areas are applicable to software process improvement. Domains of software processes exist and benefit from domain analysis. Reusable abstractions of project-specific processes can be created and reused. When made available through web-browsable repositories, a mechanism exists that allows continuous improvement of the reusable asset.

The case study showed trends in increasing process reuse. Process reuse increased from 41% in 1994 to 55% in 1995. Early results show a 10 to 1 decrease in time and effort to create a project or business unit process description when instantiating a reusable process. Unfortunately, the case study did not contain enough data points to quantitatively assess the value of the two proposed solutions, a process definition methodology and a process tailoring methodology; more time is needed.

### **7.2 Future Research**

There are a number of avenues available for future research:

- First and foremost, the case study needs to be extended so that enough time is allowed to quantitatively assess process reuse. Sufficient data should be collected and statistically analyzed to test the thesis. The cost benefit of process domain analysis and the instantiation of reusable processes should be studied further.
- Factors affecting process reuse need to be studied. How do goals, process structure, organizational and human aspects, and automation affect process reuse? What methodologies and meta-process architectures can be created to quantify these relationships?
- Process definition and reuse need to be studied in other problem domains in order to construct a more global and complete



representation of processes. Different organizational structures and non-software domains should be investigated, including those within industrial engineering and operational research areas.

- The essential and unique attributes of processes as a class need to be studied and defined. Can a formal language be developed to group processes? What would be required and how would one transition to its use?
- How can the underlying process models of CASE tools be integrated into existing organizational process domain models? How would one efficiently and effectively implement such an integration?
- What is the relationship between reusable processes and reusable products? How can the research and practice in software reuse be leveraged and integrated with the research and practice in software process improvement? How can the gradual, continuous improvement of processes be used to support the breakthrough improvements of domain analysis?

## References

- [Arango 89] Arango, G. "Domain Analysis -- From Art Form to Engineering Discipline," Proceedings of the 5th International Workshop Software Specification and Design, CS Press, Los Alamitos, Calif, 1989, pp. 152-159
- [Arango 94] Arango, G. "Domain Analysis Methods," from Software Reusability, edited by Schafer, W., Prieto-Diaz, R., and Matsumoto, M., Ellis Horwood, 1994
- [Armitage 93] Armitage, J., Kellner, M., Phillips, R. Software Process Definition Guide, Software Engineering Institute SEI-93-SR-18, Pittsburgh, PA, August 1993
- [Arthur 93] Arthur, L., Improving Software Quality: An Insider's Guide to TQM, New York: John Wiley, 1993
- [Bailin 91] Bailin, S. KAPTUR: A tool for the preservation and use of engineering legacy, CTA, Inc., Rockville, MD 20852, 1991
- [Basili 84] Basili, V., Weiss, D. "A Methodology for Collecting Valid Software Engineering Data," IEEE Trans. Software Engineering, Vol. SE-10, No. 3, November 1984
- [Bechtold 94] Bechtold, R., Brackett, J., Redwine, S. Process Definition and Modeling Guidebook, Volume 2: Advance Applications of MPDM, Software Productivity Consortium, SPC-92041-CMC, Version 02.00.02, March 1994
- [Castano 93] Castano, S., De Antonellis, V. "Reusing Process Specifications", from Information System Development Process, Proceedings of the IFIP WG8.1 Working Conference on Information System Development Process, Como, Italy, 1993

- [Conte 86] Conte, S., Dunsmore, H., Shen, V. Software Engineering Metrics and Models, Benjamin Cummings Publishing Company, Menlo Park, CA, 1986
- [Curtis 92] Curtis, W., Kellner, M., Over, J. "Process Modeling," Communications of the ACM, Vol. 35, No. 9, September 1992, p. 75-90
- [Ett 95] Ett, W., Kellner, M., Over, J., Phillips, D. Defining Manually Enactable Processes Using the Process Definition Information Organizer Templates, Electronic Systems Center, Contract No. F19628-C-029, Task IV02.1, CDRL Sequence A014-010, Defense Technical Information Center, Cameron Station, Alexandria, VA 22304-6145, March 6, 1995
- [Fagan 86] Fagan, M. "Advances in Software Inspections," IEEE Transactions on Software Engineering, Volume SE-12, Number 7, July 1987, pages 744-751
- [Feiler 92] Feiler, P., and Humphrey, W. Software Process Development and Enactment: Concepts and Definitions, Software Engineering Institute CMU/SEI-92-TR-04, Pittsburgh, PA, September 1992
- [Frakes 93] Frakes, W. lecture notes, "Advanced Topics in Software Engineering: Software Reuse, Domain Analysis, Re-engineering", Virginia Tech, 1993
- [Frakes 95] Frakes, W., Fox, C. "Sixteen Questions About Software Reuse", Communications of the ACM, Vol. 38, No. 6, June 1995
- [Frakes 96] Frakes, W., Terry, C. Software Reuse Models and Metrics: A Survey (to appear), ACM Computing Surveys, 1996

- [Gale 90] Gale, J., Tirso, J., and Burchfield, C. "Implementing the Defect Prevention Process in the MVS Interactive Programming Organization," IBM Systems Journal, Vol. 29, No. 1, 1990, pages 33-43
- [Ginsberg 95] Ginsberg, M., presentation, "Tailoring and the CMM," The 95 Software Engineering Symposium, Software Engineering Institute, September 1995
- [Humphrey 89] Humphrey, W. Managing the Software Process, Addison-Wesley, 1989
- [Jaworski 90] Jaworski, A., Hills, F., Durek, T., Faulk, S., Gaffney, J. A Domain Analysis Process, Technical Report Domain-Analysis-90001-N, V 01.00.03, Software Productivity Consortium, Herndon, VA 22070, January 1990
- [Kang 90] Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, S. Feature-Oriented Domain Analysis (FODA). Feasibility Study, Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Pittsburgh, PA 15213, November 1990
- [Krasner 92] Krasner, H., Terrel, J., Ett, W. Using SPMS to Support Process Modeling and Project Planning, STARS Program, Task IT15.2, CDRL 4024-001, Gaithersburg, Maryland, July 24, 1992
- [Kumagai 91] Kumagai, A., Riddle, W. Session Summary: ISPW6 Opening Session, Proceedings of the 6th International Software Process Workshop, Hakodate, Japan, October 1990, IEEE Computer Society Press, 1991
- [Latour 91] Latour, L., Wheeler, T., Frakes, W. Descriptive and Prescriptive Aspects of the 3 C's Model: SETA1 Working Group Summary. Ada Letters, 1991. XI(3): p. 9-17.

- [Lubars 91] Lubars, M. "Domain analysis and domain engineering in IDeA", Domain Analysis and Software Systems Modeling, IEEE Computer Society Press, 1991
- [McDaniel 95] McDaniel, M., presentation, "Best Practices for Tailoring the Software Process," The 95 Software Engineering Symposium, Software Engineering Institute, September 1995
- [Over 94] Over, J., Kellner, M. tutorial, "Fundamentals of Software Process Definition," 1994 SEPG National Meeting, Dallas, TX, Software Engineering Institute, Pittsburgh, PA, 1994
- [Paulk 93a] Paulk, M., Curtis, B., Chrissis, M., and Weber, C. Capability Maturity Model for Software, Version 1.1, CMU/SEI-93-TR-24, February 1993
- [Paulk 93b] Paulk, M., Weber, C., Garcia, S., Chrissis, M., and Bush, M. Key Practices of the Capability Maturity Model, Version 1.1, CMU/SEI-93-TR-25, February 1993
- [Prieto-Diaz 87] Prieto-Diaz, R., "Domain Analysis for Reusability," Proceedings of COMPSAC '87, IEEE, 1987
- [Prieto-Diaz 91a] Prieto-Diaz, R., "Implementing Faceted Classification for Software Reuse", Communications of the ACM, Vol. 34, No. 5, May 1991
- [Prieto-Diaz 91b] Prieto-Diaz, R. & Arango, G., Domain Analysis and Software Systems Modeling, IEEE Computer Society Press, Los Alaitos, Calif., 1991, page 13
- [Radice 85] Radice, R., Roth, N., O'Hara, Jr., A., Ciarfella, W. "A Programming Process Architecture," IBM Systems Journal, Vol. 24, No. 2, 1985

- [Redwine 91] Redwine, S., "Organizational Properties and Software Process Models," Proceedings of the 6th International Software Process Workshop: Support for the Software Process, IEEE Computer Society Press, 1991
- [Redwine 93] Redwine, S., "Software Process Architecture Issues," Proceedings of the 7th International Software Process Workshop: Communication and Coordination in the Software Process, IEEE Computer Society Press, 1993
- [Sutton 90] Sutton, S., Heimbigner, D., Osterweil, L., "Language Constructs for Managing Change in Process Centered Environments," Proceedings of the Fourth SIGSOFT Symposium on Software Development Environments, Software Engineering Notes 15, 1990
- [Weide 91] Weide, B., Ogden, W., and Zweben, S. "Reusable Software Components" from Advances in Computers, Vol. 33, Academic Press, Inc. 1991

## **Vita**

Craig R. Hollenbach  
PRC Inc., Technology Center  
1500 PRC Drive, Mailstop 5s2a  
McLean, VA 22102  
703/556-2006  
hollenbach\_craig@prc.com

### **EDUCATION:**

MS, Computer Science, Virginia Polytechnic Institute and State Univ., 1995  
BS, Music Education, Lebanon Valley College, 1974

### **CAMEO:**

Mr. Hollenbach has more than 15 years experience in the production and process improvement of software engineering systems. Currently, he chairs the PRC Technology Center Software Engineering Process Group (SEPG) and an internal working group tasked with developing CMM Level 3-5 corporate processes. He is a member of the PRC SEPG, which coordinates the software process improvement program within PRC, and the Systems Integration SEPG. Mr. Hollenbach is responsible for PRC's organizational process definition capability, including the development and tailoring of reusable processes. He has developed the first PRC Process Asset Library and prototyped its present Web format. His process improvement experience includes modeling and standardizing software engineering processes, developing corporate software engineering guidelines, assessing project software capabilities, developing configuration management tools, providing consultation services on methodological and CASE tool issues, and developing and conducting software engineering training. His product engineering experience includes the analysis, design, and development of information management, data base, and telecommunications systems. His systems engineering experience includes the analysis and design of local area network and office automation systems.

## EMPLOYMENT HISTORY:

11/85 - Present, Principal Computer Analyst, PRC Inc.

03/83 - 11/85, Member of Technical Staff, Contel

10/80 - 03/83, Member of Technical Staff, Moshman Associates, Inc.

## PUBLICATIONS/PRESENTATIONS:

Hollenbach, C. and W. Frakes. *Software Process Reuse*. in *Seventh Annual Workshop on Software Reuse*. 1995. St. Charles IL

Hollenbach, C. and W. Frakes. *Software Process Reuse in an Industrial Setting*. submitted to *Fourth International Conference on Software Reuse*. 1996. Orlando, FL: IEEE CS Press.

Hollenbach, C. *Organization Process Definition Experiences at PRC Inc.* presentation at *1995 SEPG Conference: Practical Experiences: Process at Work*. 1995. Boston, MA

Hollenbach, C. *Bringing CMM-Compliant Peer Reviews to Projects*. in *PRC's Technology Transfer*. 1994. McLean, VA

Hollenbach, C. *Phoenix Approach to Process Definition and Management*. presentation at *PRC Software Process Improvement Technical Symposium*. 1993. McLean, VA

## PROFESSIONAL AFFILIATIONS:

IEEE Member, Software Engineering  
SEI Affiliate

## AWARDS/HONORS:

Who's Who in American Colleges and Universities, 1974