This paper appeared in the IBM Systems Journal, 33(3):501–529, 1994.

Emerging Technology Supporting the Process Cycle
The Process Reuse Study at IBM CAS

George T. Heineman
John E. Botsford
Gianluigi Caldiera
Gail E. Kaiser
Marc I. Kellner
Nazim H. Madhavji

Abstract

The Process Reuse Study (PRS) is dedicated to research related to software development and maintenance processes. These processes are key elements in understanding and improving the practice of software engineering. PRS attempts to provide a holistic vision by defining a life cycle for software processes and methods for designing, improving, and performing software process models. In this report, we describe the motivation and rationale for the various areas of research of PRS and present our current status.

1 Introduction

In recent years, the software engineering community has come to recognize the importance of the processes that are employed to develop, support, and maintain software. Curtis *et al.* [20] cite the following objectives as motivation for developing and applying models of software processes:

Objective	*Section*
Facilitate human understanding and communication	3
Support process improvement	4
Automate process guidance	5
Automate process execution support	5
Support process management	6

These objectives are presented throughout the paper as tables (reproduced from [20]), each containing clusters of related goals. We match current research within PRS against these goal clusters. This list of goals is by no means exhaustive, but it does provide discrete markers in software development processes by which one can measure

the progress and impact of PRS. In the next section, we describe the problems that PRS is intended to solve. We then present the PRS framework and in separate sections discuss how PRS addresses each of the objectives. We conclude with a discussion of the contributions of PRS to date and plans for future work. This is the first publication which fully describes the breadth and depth of PRS research.

The problem

The primary concern of the software engineering community is the development and support of quality software. The two basic approaches toward this goal (see [82], for example) are improving *product* quality and improving *process* quality. Product quality focuses on end deliverables, and is associated with such concepts as rate of fault occurrence, mean time to failure, and other measurable quantities. There are known methods to improving product quality, such as code reviews. Process quality focuses on the processes used to produce end deliverables, and is concerned with such topics as accuracy (the degree to which the product produced by the processes matches the intended result) and fitness (the degree to which the people involved in the processes can faithfully follow specified actions). If we consider code reviews within a particular process, we see that the process is greatly affected by such things as: when the code reviews should be performed, how they will be done, who should participate, and how the results should be applied.

While the traditional focus has been on product quality, there is an increased awareness of the benefits of improving the quality of processes. The International Organization for Standardization (ISO) has created an ISO 9000-3 standard [43] specifically for the software industry. This standard defines a life cycle quality system governing the development and maintenance of software. The Malcolm Baldridge Award [31] has "Management of Process Quality" as one if its seven assessment categories. Finally, the Capability Maturity Model (CMM) [75, 76, 77], produced by the Software Engineering Institute (SEI) at Carnegie Mellon University, defines increasing maturity levels of an organization based on the quality of the processes employed. This trend of heightened process awareness can be viewed as the maturing of software engineering as a discipline.

The PRS approach

PRS is a consortium of faculty and students from four universities and the SEI, together with a principal investigator at the IBM Centre for Advanced Studies (CAS). PRS produces a holistic vision of software processes by defining a life cycle and a method for designing, improving, and performing software process models. The basic premise of PRS is that the quality of a software product is largely determined by the

quality of the processes used to develop and maintain it [41]. Product improvements can be achieved by constructing models from the actual processes, improving the models, and then implementing the improved processes in practice.

Expected benefits

The benefits from this work are expected to accrue in three areas:

1. PRS results would play a significant role in controlling the quality of the software that is developed and the productivity of the development group. PRS is currently working with several development groups at IBM who are interested in applying process technology.
2. IBM Lab staff will be exposed to process modeling techniques. We expect to uncover new process methods and techniques, and design and implement new tools and novel process models. These tangible research results will be packaged for developers, and we expect that a set of state-of-the-art techniques will be transferred to a subset of the Lab population (say the SEPG[1]). In addition, IBM Lab staff will be exposed to a constant and increasingly sophisticated process-awareness program. PRS will provide process training and exposure to tangible research results to developers, simplifying the future use of software process technology at IBM.
3. IBM Lab staff will increase its contact with the academic world. All of the PRS students have spent time at CAS, with most students staying for several months, and one of the PRS faculty spent three months visiting CAS. This interaction between IBM and the academic world introduces the researchers of PRS to real-world problems, and IBM benefits by receiving real-world solutions.

Terms and definitions

Before discussing PRS, we define some terms and definitions that might not be clear to readers unfamiliar with software processes; most of these are taken from [28], where Feiler and Humphrey define a set of essential terms and concepts about software processes.

- A **process** is a set of partially ordered steps intended to reach a goal. The goals for software development processes include the production, or enhancement, of quality software products. Other software processes include maintenance.

[1] Software Engineering Process Group.

- A **process model** abstracts and captures those aspects of a process relevant to the modeling formalism used. Any abstraction of a process can, in fact, be a process model, but process models are most useful when they can be analyzed, simulated, and validated. They can also be used to aid process understanding.
- **Agents** are the entities that perform a process model by carrying out individual process steps.
- **Automation** is the use of machine agents to perform individual steps.
- An environment **enacts** a process by providing automation and guidance to the agents carrying out the process while enforcing any process constraints. An enacted process is an active process.
- An **enactable process** is *instantiated* from a process model and contains all information necessary for an environment to enact the process.
- **Organizational structure** is the configuration of people and other resources that perform activities within an environment and the relationships between them.
- **Process management** involves all activities that plan, control, and manage processes.

2 Process Reuse Study

Effective software processes are one of the most significant assets of a large software development organization; unfortunately, they are often undervalued. The basic premise of PRS is that the quality of a software product is largely determined by the quality of the processes used to develop and maintain it [41]. Describing software processes, studying, and improving them can improve the quality of the software and the way it is developed. Software processes must be as meticulously maintained as the software that they have produced since they might need to change over time. What is needed is nothing less than a life cycle view for software processes.

Madhavji [60] provides such a life cycle view, containing four parts: description and definition, customization and instantiation, enactment, and improvement. From these steps, a *process cycle* is created that defines the key roles played by humans, categories of tools used, goals and policies governing the process, and interrelationships and feedback among the different roles. The process cycle in Figure 1 defines the scope of all process steps necessary for the development and evolution of software processes. To realize this process cycle, PRS has developed a framework, illustrated in Figure 2, that is an implementation of the process cycle. We now describe in more detail the process cycle and the framework.

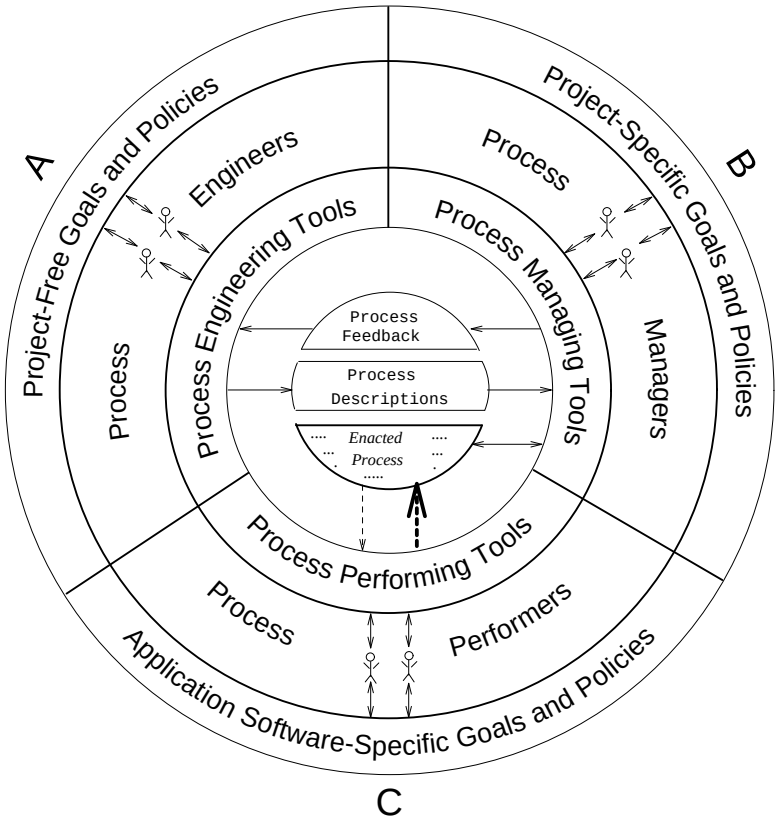

Figure 1: The Process Cycle [60]

2.1 Process cycle

The process cycle is divided into three sectors which represent the engineering, managing, and performing of software processes. In sector A, *process engineers* design, construct, and improve generic process models[2]. These models are generic in the sense that they have not yet been tailored a for particular software project. The models are customized by *process managers* in sector B and are instantiated for use; *process performers* in sector C carry out the processes.

Two essential concepts to the process cycle are the tools used by each sector (see Figure 3) and the feedback among sectors. The success of the process cycle depends heavily upon the success of these tools. It is expected that no single tool can be used in all sectors. Some tools are very adept at modeling and simulating processes, but have no means of enactment, while other tools created to enact processes might be weak in modeling capabilities. The ability to integrate a heterogeneous set of approaches and tools is essential to an effective implementation of the process cycle.

The second important aspect of the process cycle is the feedback among sectors that can be used in each sector to improve processes. Feedback from process performers

[2]When the context is clear, we refer to these simply as models.

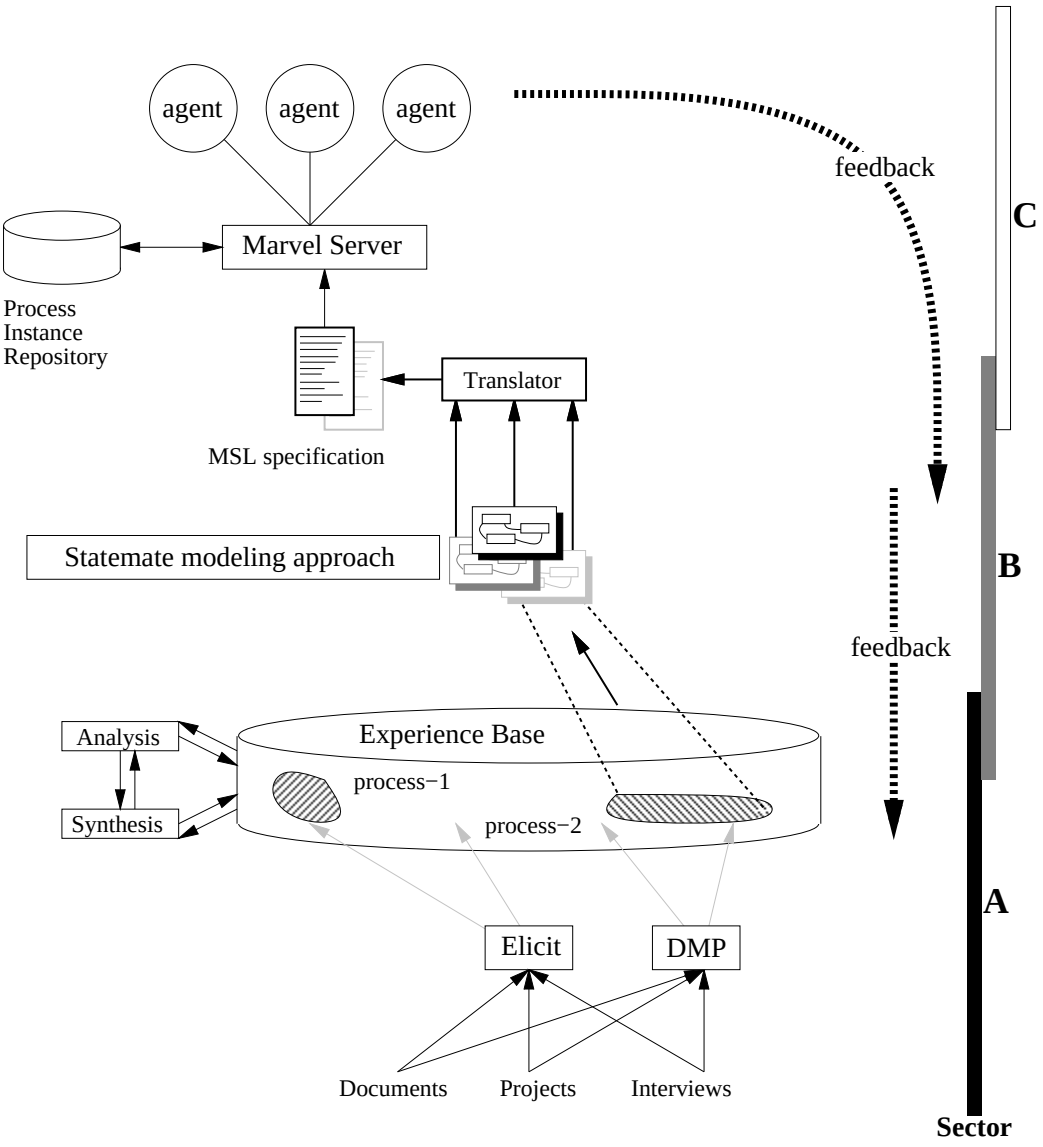

Figure 2: The PRS Framework

Sector A	Tools to create, and operate on, fragments of processes within simulated environments. Example: `elicit`
Sector B	Tools to operate on specific descriptions within simulated and actual environments. Example: `statemate`, `marvel`
Sector C	Tools that operate on the software parts being constructed in actual environments. Example: `cc`, `vi`

Figure 3: Process Cycle Tools

in sector C might reflect the smoothness with which assigned tasks are carried out, bottlenecks in the development process, negative effects caused by certain timing constraints, and the need for additional process steps or removal of superfluous ones. Both qualitative and quantitative data about processes are useful. Process managers can use feedback in several important ways. First, project-specific processes can be improved through modifications in response to the feedback. Second, generalizations and improvements in models can be suggested to the process engineers in sector A. Third, an iterative approach (see Section 4.2, for example) can be used to identify the appropriate changes to be made to organizational structures and development processes to satisfy project goals. This cycle of building models, tailoring them, and improving them through feedback is continuous [59].

2.2 PRS framework

The PRS framework, illustrated in Figure 2, is our proposed implementation of the process cycle. The approaches on the left side of the figure are matched with an appropriate sector of the process cycle on the right. For example, the Elicit approach occurs within Sector A. We now outline the PRS approach to software processes by describing each of the entities of the framework.

Process engineers first model a process by observing existing processes within an organization. The Elicit and DMP projects (discussed in Section 3) address the aspects of understanding an organizational structure and its processes by extracting the appropriate information from documents and interviews. As models are created and tailored, they are stored in an *experience base*. This repository contains the models and their histories—all modifications to models, lessons learned, and decisions and alternatives.

The experience base is derived from the component factory concept proposed by Basili *et al.* [8]. This repository has three levels of representation that contain descriptions of the agents in the organization, the activities they perform, and the processes used. The three-tiered structure of the repository (see Section 3.9 for further details) suggests that the modeling formalism used should have some mechanism for supporting multiple levels of abstraction.

The SEI-developed Statemate[3]-based modeling approach [52, 53, 42] models software processes through formal visual languages based on *hi-graphs* [25], a form of diagrams using boxes, directed lines, and hierarchical nesting. When a process is modeled, three forms of charts are produced, each providing a separate yet interrelated view of the process. *Activity charts* capture the functional perspective of the process and focus on the activities being performed while hiding the actual details of how they are carried out. *State charts* provide the behavioral perspective and focus on the

[3]Statemate is a trademark of i-Logix, Inc.

timing and ordering of the individual process steps. Finally, *module charts* describe the organizational units involved and the physical communication channels used to transfer information between the activities. These three charts provide an integrated vision for process models and correspond directly to the three levels of the experience base.

Statemate [25] provides the analytic and simulation capabilities required by process engineers. With this tool, engineers can step through a modeled process, run batch simulations, generate events, and observe the reactions of models. Recent work by Kellner [50] shows how Statemate can be used to provide quantitative estimates of process time and resource allocation. Process managers can correlate these estimates with statistics derived from the actual performance of processes. Because Statemate offers little help in performing processes, PRS incorporates the MARVEL process centered environment [13].

In MARVEL, a process model is defined by a process modeling language, called MSL, with each process step encapsulated by a *rule*. Each rule is composed of a condition, an optional activity, and a set of effects. The condition is a prerequisite that determines when the process step can be carried out. The effects are the immediate implications of the process step. Each process model contains an object-oriented data schema that defines the process state and the organization of the particular product data. MARVEL provides process automation through backward chains, that attempt to satisfy the prerequisites of a process step, and forward chains, that carry out all implications of a process step. Recent work has shown how Statemate charts can be automatically translated into MARVEL rules [35], thus producing a multi-user environment that can enact a given process model. Sector C tools (see Section 2.1) can be incorporated into the MARVEL environment, and measurements can be taken as processes are performed, providing useful feedback to process managers.

In the following four sections, we discuss PRS research within the context of our original objectives. At the close of each section we assess how well PRS research addresses the goal clusters for the objectives. This list of objectives is not exhaustive, but it is a convenient yardstick against which the effectiveness of PRS can be measured.

U1	Represent processes in a form understandable by humans
U2	Enable communication about and agreement on software processes
U3	Formalize processes so that people can work together more effectively
U4	Provide sufficient information to allow an individual or team to perform the intended processes
U5	Form a basis for training the intended processes

Table 1: Goals for human understanding and communication

3 Human Understanding and Communication

The first step towards our goal of improving software processes is understanding them. Table 1 presents five motivating goals for employing models for understanding software processes. Industrial software processes often require more than a year to carry out a development or support cycle [16], making it inconvenient to acquire understanding through direct observation. Because software processes are extremely complex, managers and participants frequently lack a broad sense of intellectual control over them. Even simple processes quickly become complicated because there can be many processes performed simultaneously by different people, so that no one person often understands the entire set of processes. Consequently, gaining understanding is a valuable goal in its own right.

The goal cluster in Table 1 focuses on effectively communicating the description of a process to others, such as workers, managers, and customers [74]. Process developers and researchers must always remember that their goal is to help people understand processes. Models and definitions are simply vehicles for this purpose and are useless if they themselves cannot be understood. A complicating factor in facilitating understanding is that so many diverse groups access models, including software process engineers, project managers, software engineers, system engineers, software executives, and customers. These different groups may place widely differing demands on a modeling approach because of their different information needs and expertise. Visual approaches, good use of abstraction, and multiple perspectives offer promising techniques for coping with these challenges. We now describe the two PRS techniques used to construct models from existing processes: Elicit and DMP.

3.1 Elicit

It is our belief that valid models are most easily derived from existing processes. Madhavji *et al.* [65] describe a methodical approach, called Elicit, that elicits models from active processes. Elicit is actually a *meta process* for eliciting software process models with the following steps:

1. Understand an organizational environment
2. Define elicitation objectives
3. Plan the elicitation strategy
4. Draw out process information
5. Synthesize and translate models
6. Perform the analysis

Elicit[4] progresses from an implicit state (undescribed, enacted processes) to an explicit state (described, analyzed models). The derived models are collected and made part of the experience base (see Section 3.9) for the organization, ready to be used for further understanding and improvement. Eliciting software process models from enacted processes involves all the activities that are necessary in drawing out process information from various sources and synthesizing it into appropriate models. This elicitation is similar to bottom-up process re-engineering; they both identify the work that an organization should perform and define efficient and effective processes for that organization. Elicit is an evolutionary method which produces feedback that might lead to the iterative repetition of previous steps.

Elicit has been applied and documented in several laboratory and industrial environments. The basic concept of Elicit is discussed in detail in [65]. At IBM Software Solutions Toronto Laboratory, Elicit has been applied to elicit the lab-wide recommended requirements process [64]. Based on this experiment, Elicit was then analyzed and improved [63]. Formalizing, applying, measuring, analyzing, and improving Elicit was the subject of a PRS student's Master's thesis research project [37]. Elicit has further been studied in a laboratory environment to elicit a literature assimilation process [47].

In an early application of the process cycle, Madhavji *et al.* [66] designed a semi-formal process notation meant to be used as a vehicle for understanding and communicating process information. Process notations were developed in each sector of the process cycle using a specification-implementation paradigm. Further work on the infrastructure needed to support the communication of processes is outlined in [68]. These human understanding and communication goals were in fact at the center of an earlier IBM-McGill project on software processes [69] which led to the recommendation of a process modeling method and of a supporting tool. We now describe in detail the six steps to Elicit.

3.2 Understand an organizational environment

The first step of the meta process is to understand the environment from which models need to be elicited. This step is of fundamental importance since it defines realistic

[4]When the context is clear, we abbreviate *Elicit meta approach* with *Elicit*.

objectives upon which the rest of the elicitation process is based. The following partial list illustrates some of the knowledge captured by this step:

- *Projects of concern*: Is the process to be examined project-specific, multi-project, or organization-wide?
- *Team structure*: Are the teams organized hierarchically or democratically? Are they autonomous or authoritarian?
- *Degree of process awareness*: How strong is process awareness among the various groups and individuals in their respective roles? What purposes would models serve? Are the expectations from described processes unreasonably high? How soon are the process descriptions required? What quality of process descriptions are required?

The input to this step is organization knowledge; its outputs are documented contexts considered useful for the elicitation tasks that we now describe.

3.3 Define objectives

In this step, the specific objectives of the elicitation task are defined. These objectives are used to measure the quality of the elicited models. Without such objectives, it would be difficult to control the elicitation project. Some objectives to be considered are software process domain, granularity, consistency, completeness, cost, urgency, and resource constraints.

The software process domain is especially important because it provides a setting within which particular elicitation objectives achieve the greatest impact. The appropriate domain should be chosen, based on many factors, including:

- *Known process problems*: Which parts of the process are ill-defined or not well understood? Which parts of the software process contribute to software quality problems? These weak process areas are revealed by process assessments.
- *Expected process changes*: For example, if a new set of tools are to be added to the design process, this process may need to change to accommodate the new tools.

Granularity and consistency determine the amount of effort to be undertaken by setting the level of details of the elicited models. The more fine-grained a process model needs to be, the more effort will be required to produce it. Completeness also impacts the cost, since it determines the scope of the elicitation. Urgency and resource constraints determine how soon the elicited model needs to be produced, and whether a given set of human resources will be available to complete the desired elicitation.

3.4 Plan elicitation strategy

This step builds an elicitation plan and allocates appropriate resources so that when the plan is executed, the desired quality of the elicited process model is developed within budget and on time. This activity involves identifying the scope of the elicitation task in detail, scheduling interviews and document-understanding sessions, identifying the deliverables, selecting elicitation tools, and allocating computing and human resources. This planning step can be as complex as the management of software projects.

The elicitation strategy is affected by both the objectives and the contexts from the previous two steps. For example, the objectives may determine a budget and time schedule that limits the number (and type) of people to be interviewed and documents to be examined. In addition, the documented contexts contain knowledge about particular documents and specific people that can help plan and schedule elicitation activities. The role of the persons planning the elicitation strategy is important and should be carried out by highly skilled and influential process experts. An ineffective plan is likely to result in low-quality descriptive models and wasted time and effort. What is more, if such low-quality models are used in a software project for training practitioners, or as a basis for process improvement, then a great deal of damage can occur.

Since Elicit is iterative, the elicitation plan might require either the objectives (Section 3.3) to be redefined or the contexts (Section 3.2) to be described more clearly. This step produces a plan to be used for elicitation of models from the identified processes.

3.5 Draw out software process information

The plan created by the previous step is carried out, including reading process documentation, holding interviews, analyzing question responses, analyzing the elicited model, and demonstrating the elicited process descriptions to management. The Elicit tool[5] prompts the user at each step for process information that it represents as process models. In essence, Elicit takes care of the mechanical aspects of eliciting, leaving the user with the creative tasks of providing appropriate information. This can save considerable elicitation time and effort and reduces the chance of missing information or having erroneous information.

Once the information has been elicited, a model is created from the structured process descriptions. At this point, static semantic checks can validate the elicited model against the objectives defined in Section 3.3. In iterative fashion, the previous steps might need to be re-executed to further refine the contexts, objectives, and elicitation

[5] A tool supporting the Elicit meta process is also called Elicit.

plans. This step produces a descriptive process model.

3.6 Synthesize and translate

Once the elicited process model has been statically reviewed, its dynamic behavior needs to be examined for correctness. This could be done as part of Section 3.5 if the eliciting tool has behavioral analysis capabilities. Instead of building dynamic analysis and simulation capabilities into our elicitation tool, however, we simply translate the elicited model into a representation suitable for dynamic analysis by a commercial tool. Given this approach, the two key issues for translation are: (1) the compatibility of the modeling formalisms supported by the elicitation tool and the simulator; and (2) the method of translation (that is, hand or automatic translation) of the elicited process model.

The first issue requires the representation scheme supported by the simulator to be at least equivalent to, or a superset of, that supported by the elicitation tool, otherwise information will be lost during translation. The second issue implies that either an automatic translator is constructed, or careful hand-translation of process components is carried out, followed by reviews. To avoid the complications of translations, Elicit allows the process engineers to hand-translate the elicited information into Statemate charts (see Section 2.2) that can then be analyzed using Statemate.

3.7 Analyze

The final step of Elicit analyzes the behavioral aspects of the translated process model and, if necessary, corrects it so that it reflects the dynamics of the enacted software process. For this purpose, there is a need for appropriate process analysis and simulation tools. The process models should be checked for deadlocks or race conditions, starvation among subprocesses, reachability problems, idle time and delays in the process, and behavioral ambiguities [48, 53].

The process engineer should also observe the degree of parallelism in process subcomponents, obtain animated traces of simulated paths, experiment with "what-if" scenarios, and calculate quantitative predictions. Researchers have shown how software process models can be analyzed to make prescriptive improvements [50, 70, 71, 72]. Here, the objective is to make descriptive improvements to the elicited model that reflect the behavior of the enacted process. Prescriptive and descriptive improvements can be made at the same time.

In iterative fashion, all previous Elicit steps might need to be re-executed if flaws are found in the process model. Once this step has terminated, Elicit is complete.

3.8 Package

The fulfillment of Elicit occurs during its packaging phase. Once the information has been elicited, the constructed model must be stored in a meaningful manner. Packaged information related to the elicitation task can be reused in other tasks (such as new elicitation tasks or process improvement efforts) and can be helpful for educating process users. Packaging is particularly worthwhile if there are on-going or numerous elicitation (or process improvement) efforts in the organization. There are several key perspectives to consider when packaging the elicited information: objects of interest to package, users of the packaged information, and reusability of the packaged information.

The objects of interest can be categorized into either *products* of Elicit or *experience* gained in eliciting software processes. Clearly, we want to store the models created by Elicit; we also store the experience gained, including the contexts, objectives, and elicitation plans constructed by Elicit. This ability to store all pertinent information increases the integrity and cohesion of all elicited models.

The users of the packaged information are varied; they include process educators, process improvers, managers, and elicitation-task performers. For example, process educators can use elicited models for software process-understanding tutorials, while process improvers can use the elicited process model and organizational environment as a basis for process improvements. Project managers can obtain reports to create cost/benefit analyses while elicitation task performers can use the packaged strategies in other elicitation tasks. The number of users and the ease with which they access the packaged information increases its reusability. For example, organizational environment details would be extremely useful in eliciting other software process models within the same software project.

The process modeling formalism needs to be flexible enough to satisfy the different needs of the users who will be accessing the packaged information. Two features we have found useful in modeling formalisms are multiple levels of abstraction and multiple perspectives. The users should be able to start understanding the process models from the top working down, from the bottom working up, or even from the middle working either way [48, 53]. Furthermore, representing multiple perspectives on a process can be quite beneficial; these may include functional, behavioral, organizational, and informational perspectives [20]. Capabilities, such as levels of abstraction, precision, and multiple perspectives, offer distinct advantages over representations based upon narrative text or a single type of diagram. PRS envisions that all elicited models will be stored in an experience base whose structure we now describe.

3.9 Experience base

The experience base is derived from the component factory concept by Basili *et al.* [8]. The component factory attempts to increase the quality and productivity of software by targeting three goals: improving the effectiveness of the software processes, reducing the amount of rework, and reusing life cycle products. The production of software using reusable components is a significant step forward for all three of these goals, but there are still problems in achieving higher levels of reuse. The current inability to package experience in a readily available way prevents the transfer of experience from one project to another. Another problem is the difficulty in recognizing experience that is appropriate for reuse. Finally, reuse needs to be an integral part of software development processes before it can be truly effective. The experience base addresses these concerns by applying the concept of the component factory to the domain of software processes.

Like the component factory, the experience base has three levels of abstraction representing the different aspects of a process. The highest and most abstract level, Reference, describes the agents in the organization. The Conceptual level represents the interface of the agents and the flows of data and control among them. The lowest level, Implementation, defines the actual implementation, both technical and organizational, of the agents and of their connections specified at Conceptual level. These three levels correspond exactly to the three perspectives modeled by the Statemate-modeling approach described in Section 2.2.

The experience base is similar to the Process Asset Library (PAL) [55] prototype developed in 1992 by the SEI in collaboration with the STARS[6] program of the U.S. Department of Defense. The PAL prototype includes nine processes, each represented by a model, and seven by a process guide as well. These nine assets were primarily based on pre-existing process documentation, ranging from an IEEE standard, to process guidebooks, to a journal article. The PAL prototype contains over twelve hundred pages of process documentation (models and guides) and was the first publicly available collection of industrial-strength models.

3.10 Descriptive Modeling Process

Since 1987, Kellner and his colleagues at the SEI have developed more than a dozen models of industrial-scale software processes. Several of these are descriptive models of processes as they are actually performed in a specific real organization [49, 52, 53]. This experience has been coalesced into a Descriptive Modeling Process (DMP) which has been taught in detail to over 200 software professionals for use in their organizations. This three-day workshop offered by the SEI will be documented in a forthcom-

[6] Software Technology for Adaptable, Reliable Systems.

ing paper. DMP produces (semi-) formal models of a current processes as practiced in a specific organization. Considerable effort and care are necessary to accurately reflect actual organizational practice, as opposed to the process documentation or as people wish the process were performed. Developing such models involves extensive interviews with process performers to elicit the information and verify the models. The primary inputs to DMP are existing process documentation and knowledge of the as-practiced processes from those who perform and manage them. DMP creates a descriptive model of the as-practiced processes in a variety of representation languages, including Statemate, ETVX (Entry-Task-Validation-eXit), and IDEF0. Most descriptive process models are developed in an iterative, top-down fashion. As an example, one comprehensive model developed at the SEI involved interviews of approximately 25 individuals and 5 iterative rounds of construction activity [49].

Both DMP and Elicit create process models that clearly describe the processes and can be used to aid the performers of the processes (U4) to train new members in the processes (U5).

Achievements in understanding

Formalizing existing processes increases the understanding of the processes. We have found that modelers always increased their understanding of processes through the experience of developing models. More importantly, when process participants, managers, and other personnel viewed a model, they almost invariably reported increased understanding (U1). The impact that models have on process understanding shows their communicative power. Personnel ranging from software engineers to senior managers have found these models to be understandable and useful. The models have allowed them to clearly visualize how the various process components are interrelated. Their comments, questions, and insights arising during presentations of these models demonstrate that models are valuable communication vehicles. Process participants have gained a deeper understanding of portions of the processes in which they do not directly participate, and management personnel have gained substantial understanding as well [49, 53, 64].

The following analysis of the understanding and communication benefits taken from a specific SEI modeling effort [53] shows how U2 and U3 are addressed. In this study, Kellner and Hansen focused on the process of identifying and making changes to Technical Orders[7](TOs) in response to changes in the software; in particular, changes to the Operational Flight Program (OFP) for the F-16A/B multi-role fighter. A typical new release of the F-16A/B OFP required changes to 3,000 pages of TOs spread over as many as 100 separate documents.

Our efforts at modeling the TO Modification Process led to a number of

[7] A USAF term referring to technical documentation for end users.

> very important results. The first of these is a notable increase in understanding of the process by those involved in executing and managing it.... Several different organizational subunits are involved in various stages of the TO Modification Process. Not surprisingly, the view of most of these subunits was somewhat parochial, focusing on their specific subtask with little awareness of overall implications....
>
> In the course of our interviews, we gained an increased appreciation of the overall goals of the process, as well as a recognition of the effect of regulations and standards on the process. Communication of this information to relevant personnel can result in significant benefits. A common understanding of the overall process and the role each group's work plays in its successful completion, may lead to increased goal congruence among those involved....
>
> We also found that graphical representations of the process were far more effective communication vehicles than narrative presentations. Rapidly building a common base of understanding seems crucial in arriving at a point where fruitful discussions can occur regarding the impact of new technology, process streamlining, effects of regulations, and so forth.

Once a model has been constructed, its usefulness can extend to other domains. For example, a model created to describe the factors that impact the success of the requirements engineering process [22] was subsequently applied to process design, customization, and reuse [78].

I1	Identify all the necessary components of high-yield software development or maintenance processes
I2	Reuse well-defined and effective software processes on future projects
I3	Compare alternative software processes
I4	Estimate the impacts of potential changes to a software process without first putting them into actual practice
I5	Assist in the selection and incorporation of technology (for example, tools) into a process
I6	Facilitate organizational learning regarding effective software processes
I7	Support managed evolution of a process

Table 2: Goals for process improvement

4 Process Improvement

The second goal cluster (Table 2) is concerned with evolutionary improvements to a process. There are a variety of reasons why an existing process might need to be changed:

- Potential gains in productivity
- Potential gains in product quality
- External factors, such as schedule pressures or political factors, require the process to be modified or steps to be removed
- Process refinement by the development team
- Invalid assumptions under which the process was designed

Software process models support the identification and analysis of potential improvements. This is a primary objective of software process modeling because of its connection with the improvement of software quality, cost, and scheduling. The models help highlight areas of opportunity for process improvements and can be used to evaluate potential improvements before they are put into practice. Models serve as storehouses for modifications, lessons learned, and tailoring decisions, thus recording the evolution of a process along with the outcomes of all changes made. This retrospection provides a basis for evaluating the relative success of such changes. In addition, tailoring decisions may be formalized and stored as part of the models, so that this knowledge can be consistently applied again in the future [49].

There are many different approaches to software process improvement. A brief list includes the Quality Improvement Paradigm (QIP) [10], Total Quality Management (TQM) [33], and SEPG's application [30] of the SEI's CMM [75]. Tool insertion in the software process is another approach that shows great promise [87]. In the next few sections, we describe the various research of PRS dedicated to process improvement.

4.1 Tool Insertion Method

Bruckhaus [17] describes TIM[8], a comprehensive method for inserting a tool into a software process that plans, executes, and controls the tool insertion. Currently, the method of inserting a software tool into a software development process is mostly *ad hoc*. Tools are sometimes purchased based on informal recommendations and are put immediately into practice; often a tool does not live up to its expectations. Little or no planning is done for the use of a new tool in a particular process. It is often not clear what benefits can be expected when a specific tool is inserted into a specific process, nor how the impact of inserting the tool can be measured. Huff [39] describes some additional complexities of tool insertion:

- The actual cost [of tool insertion] may run five to eight times greater than the cost of a CASE tool alone.
- It may take one to two years of preparation by a dedicated team of six to nine people to reach the pilot stage.
- Unanticipated costs may lead management to terminate a promising CASE tool project or may increase resistance to future CASE tool acquisitions.

Bruckhaus is currently developing TIM as part of a study to measure and analyze the impact of inserting a specific tool into an on-going, real-world software development process at the IBM Software Solutions Toronto Laboratory. As steps of TIM are executed, feedback is used to deliver improved versions of TIM. Software process modeling is TIM's primary vehicle for reasoning about, planning, quantifying expected benefits of, and measuring the impact of, tool insertion [62]. TIM has eight steps:

1. Model the development activity of interest (if the model does not already exist)
2. Identify a set of Key Improvement Areas
3. Measure current process context, data essence, and process performance
4. Quantify the goal of inserting a tool in terms of process performance
5. Select a tool and perform a pilot study
6. Customize the tool (if possible) to best suit the insertion goals
7. Insert the tool and monitor process performance
8. Take corrective actions (iterate)

TIM uses its own measurement method, TIM/M, to quantify the expected impact of tool insertion, and then monitors the actual tool usage to verify the forecasts. TIM/M is a refinement of Basili's GQM paradigm [9], with a specific focus on tool

[8]Tool Insertion Method.

insertion. Like GQM, TIM/M defines a hierarchical structure of process aspects of interest. Measurement programs defined with the help of TIM/M cover the following process aspects: process context, data essence, and process performance. Each of these aspects is then repeatedly broken down into sub-aspects. Finally, each aspect is tied to one or more metrics that help describe the aspect. Since this structure is not limited to a particular process, TIM/M can help one design measurements in any given process context (for example, requirements planning, design, or testing). TIM will have its greatest impact on goal 15.

4.2 OPT approach

Seaman [84] presents OPT[9], an approach to improve an organizational structure and a software development process when considered together as a single system. This method is patterned after QIP [10] and models the relationships between an organizational structure and a development process so that these relationships can be quantitatively measured and specific improvements can be planned in the system.

Relationships between the organizational structure and a development process are modeled using the OSL and ASL[10] [84] languages. An OSL specification describes the non-process relationships, called *links*, between different elements of the organization. Examples of such links are *reports-to* and *manages*. An ASL specification describes the agents that execute the process in terms of the specific activities that they perform and the interactions between them. Since a process model often contains extreme detail, it is often difficult to model an organizational structure since the important information must be abstracted out to be seen clearly. An ASL specification provides this abstraction and is more useful than a process model for analyzing the structure of an organization. It also provides the bridge between a development process and an organizational structure, allowing the two to be considered together as a system.

The goal of OPT is to measure how well suited a development process and an organizational structure are for each other. In other words: *How good is this process for this organization? How good is this organization for performing this process?* OPT targets two complementary factors that characterize the relationships between an organizational structure and a process. The insight OPT provides is that these factors can be measured. The first factor is the distribution of responsibility for process activities among the members of the organization. This captures the effect that the process has on the organization. The second factor is the process communication (which can either facilitate or hinder the efficient flow of information) within the organization. This captures the effect that the organization has on the process.

OPT is part of the planning phase (third step) of a QIP improvement cycle. The first

[9] Organization and Process Together.
[10] Organizational/Architecture Specification Language.

QIP step characterizes the environment by creating an organizational structure and a corresponding process model. After the initial organizational model is complete, the second QIP step creates quantifiable goals using the GQM [9] approach. These goals define the means by which improvements will be validated. Reasonable expectations are calculated from the baseline provided by the first step. The third QIP step decides the actions to take to satisfy the goals; the OPT approach is used for this purpose. As the actions are executed, the metrics chosen earlier are evaluated and analyzed to provide real-time feedback. Once all actions are completed, a *post mortem* analysis is conducted to determine if the original project goals have been satisfied. Any changes are integrated into the organizational model, and the cycle is prepared for the next QIP iteration.

OPT addresses I1 and I6 by validating a particular software process for a particular organizational structure. The key insight of OPT is that a process is not conducted in a vacuum; it affects, and is affected by, the organization that carries it out.

4.3 Quantitative approaches

Raffo [80] describes a quantitative approach to process improvement. In a process change study, Raffo used software process models to predict the impact of potential process changes before a substantial commitment of time and resources was made. Such analysis allows process improvements to be prioritized based on their potential performance impact. This approach forecasts the impact, in quantitative terms, of a proposed process change before it is put into place in the actual organization. Using this method, management can ask: *What will be the value (impact) in our organization of making a (proposed) change to our process?* This impact is measured in quantitative terms, such as effort (aggregate and/or time-profile) and schedule (total duration and/or intermediate milestones). This notion of impact focuses on the results of the change after it has been stabilized in the organization.

This investigation targets a real issue with which moderately mature organizations already wrestle. As an example, one product team at the IBM Software Solutions Toronto Laboratory collected in a database about 200 proposals for improving processes employed in supporting a particular large product. The project manager then asked the process analyst on the project to prioritize these proposals on a cost/benefit basis. The process analyst felt reasonably able to estimate the implementation costs (such as, documentation, training, and so forth), but had a serious need for a way of estimating the impact of a change in quantitative terms. The process change study directly addresses that need and helps develop a business case to obtain managerial support for specific process improvement proposals.

Raffo's approach uses simulation models of the affected portions of process with and without the proposed change. Stochastic modeling with Monte Carlo simulations is

used, although deterministic modeling can also be performed (simpler, but less realistic). This approach explicitly models the complex interdependencies among process components and employs a new technique, called Task Element Decomposition, for handling interdependent operation times in large-scale systems; this is a contribution to the operations management field. A multi-attribute decision making framework is used for comparing process alternatives, allowing management to ask such questions as: *If the starting code had more errors, would source code inspections still be beneficial? If slack time in the process wer reduced, how would process performance be affected?*

The actual modeling techniques are built upon the SEI-developed Statemate-based modeling approach; this work most directly builds upon the modeling extensions to support quantitative management planning and control [50]. These software process modeling techniques also support sensitivity analyses of the models, helping the modeler:

- Assess confidence in the results
- Fine-tune the proposed process
- Explore learning curve effects
- Suggest viable alternative processes

The results to date are fully documented in a university working paper [79] totaling over one hundred pages. The first major phase of the work developed an approach to the quantitative comparison of alternative processes, together with supporting techniques—thus directly supporting goal I3. A comprehensive representative case has been explored in detail, namely adding a code inspection process to an existing process that employed design reviews and unit tests, but not code inspections. The primary performance measures of interest were aggregate effort, duration, and defects remaining at the conclusion of the process. In this investigation, the process change resulted in somewhat more total effort, more total duration, but substantially fewer remaining errors [79].

The long term goal of this work is to develop a method, and supporting techniques, for forecasting the impact of potential process changes—thus directly supporting goal I4. This work provides a vital foundation by developing an approach and techniques for comparing alternative processes based on estimated quantitative performance from predictive software process models. A very preliminary method for the larger problem of process change has also been documented, but it requires further exploration. Additional example process changes will be considered, from which a method will be refined and documented and then tested on a full-scale real-world situation. This work is expected to be applicable to process changes at the scale of a CMM's Key Process Area (KPA) [76, 77] or smaller, but may not be well suited to larger scale questions, such as comparing entire CMM maturity levels.

4.4 Process generalization

Within a large organization, process improvement issues span multiple projects. There is a need, therefore, to improve not only individual processes in various projects (see sector B in Figure 1) but also generic process capabilities across a set of projects (sector A). The advantages of generic models include:

- Corporate-wide improvement and standardization of process components
- Development of an organization-specific culture (16)
- A base model that may be tailored to meet the needs of specific projects

Experience shows that while many organizations do indeed perform changes to prescriptive generic process models, they often have little or no information about the aspects of project-specific process models that need to be considered when changing the generic model. What is needed is a mechanism that builds *descriptive* generic process models to provide a coherent vision of commonality and variability among project-specific processes. These descriptive generic models are useful for driving the desired changes to prescriptive generic models. At McGill, researchers have developed a method, called Generalizer, for building descriptive generic process models from a set of project-specific ones [38]. The key steps to this method are:

- Elicit project-specific process models
- Decompose and categorize project-specific models
- Match process components from different projects
- Obtain goal-oriented views from categorized components
- Make cross-project comparisons based on process views (13)
- Identify commonality among these views
- Synthesize common components into descriptive generic models

In its simplest form, a generalization method leads to a pure generic descriptive model. In practice, however, while making organization-wide change decisions, it is often desirable to examine various *degrees* of generic descriptive models as possible starting points for the changes. A suitable starting point is selected based on change-related, cultural, cost, technological, quality, standardization, or other factors. Using threshold values, process engineers can specify degrees of generality when building generic descriptive models. For example, a process engineer can set the threshold to 70%, implying that the descriptive generic model will contain all process components that are common to at least 7 out of 10 projects.

The output of this generalization method are descriptive generic models that show how project-specific models can be created from the organization's generic process

models. A tool (also called the Generalizer) is being developed to aid the building of generic prescriptive process models. It allows process engineers to experiment with threshold values to create the most suitable generic descriptive models. The Generalizer works in conjunction with the Elicit method and tool (described in Section 3.1) in that Elicit helps build descriptive process models and Generalizer helps generalize project-specific models.

4.5 Evolving processes

The approaches outlined so far can target specific improvements to a process model, but we need an additional mechanism that aids the evolution of an enacted process (goal I7); the MARVEL project has such a tool, called the Evolver [45]. MARVEL [13] is a process-centered environment that defines a process model by a set of production system-style rules. Each process model is augmented by an object-oriented data schema that defines the process state and the composition of an objectbase that maintains all the data used by the process. As MARVEL enacts the process it updates the process state stored in the data schema of the objects to accurately reflect its real-world state. Changes to a process model may directly affect existing objects; the process state, in particular, is maintained by attributes associated with each object that may need to be updated to be consistent with the new process model. Even some of the most trivial process changes might require the entire objectbase to be updated. For example, since each object inherits from a generic ENTITY class, adding an attribute to this class will update all objects.

The Evolver operates on an existing MARVEL environment (with a particular process model) and the evolution consists of two steps. First, the new data schema is compared against the original one, and a detailed analysis of their differences is reported. This allows the process engineer to view the consequences of a particular schema change. The second step is concerned with the process model. MARVEL allows a process engineer to construct a sequence of rules whose forward chain must be atomically carried out; that is, if a concurrency conflict occurs at any point during the rule chain, the entire rule chain is rolled back. During this second step, the Evolver constructs a graph from the original process model with rules as nodes and chains between rules as edges in the graph. A similar graph is constructed from the new process model and the two are compared to detect when these atomic rule sequences are shortened or extended. In the latter case, the Evolver automatically generates a batch script of MARVEL commands that executes the necessary rules in the new process model to make the objectbase consistent with respect to the new graph; in both cases, the process engineer is notified of all changes. The reports generated by the Evolver can be used to view the effects a particular process change will have (I4); if the process engineer determines the change to be too costly, the evolution process can be terminated, restoring the MARVEL environment.

This process evolution operates in an *off-line* fashion. Thus, the process itself can be in progress when evolution occurs but must be quiescent; that is, all atomic rule chains have terminated and the environment is waiting for the next request to continue the process. Since the main goal of the Evolver is its ability to resume long-lived processes after changes, this off-line approach is acceptable.

Achievements in improvement

A prerequisite to improving software processes is the need to understand them; such an understanding can be expressed in the form of models, as we have seen in Section 3. Another important prerequisite is measurement. To facilitate measurement, valid and reliable instruments to measure various attributes of the process, and the software products, must be constructed. Recent work [24] has led to a method for the development of such instruments. This method uses software process models to define relevant metrics [62].

Armed with a suitable set of metrics, and an understanding of processes, the improvement task has a higher likelihood of succeeding. Such an empirical perspective has been advocated as a basis for improvement [23]. A major benefit resulting from formalizing existing processes has been a substantial number of recommendations for process improvement. Many of the SEI-developed Statemate models have been subjected to analysis (manual and some automated) and recommendations were made for process improvement as well as for technology insertion. Some of the procedural issues observed from analyzing these models include bottlenecks, insufficient parallelism, and management control versus productivity. The opportunities for beneficial technology insertion include the use of advanced document production technology, configuration management tools, and project management and planning tools [49].

Two examples, both from the SEI, show the success of using process modeling techniques to guide process improvement. The first is the case of the process used by the U.S. Navy to support the operational software for the F-14A aircraft. The corresponding model depicted the full software support process, from receipt of a software trouble report, change request, or enhancement request, through to release of the corresponding software change to the field. An extensive analysis of the model identified a total of 13 major issues for possible process improvements, resulting in over 30 recommendations for modifications to methods, procedures, and technology usage [49]. The second example is provided by the case of the military handbook (MIL-HDBK-347) on software support for mission critical systems [88]. The SEI modeling and analysis identified 35 issues that could improve the handbook process or its exposition. These improvements were revealed *after* the completion of an extensive public review process that had solicited written comments from over two hundred people [55].

The issue of change management, which is at the heart of goal 17, has been investi-

A1	Define an effective software development environment
A2	Provide guidance, suggestions, and reference material to facilitate human performance of intended processes
A3	Retain reusable process representations in a repository
A4	Automate portions of processes
A5	Support cooperative work among individuals and teams by automating process details
A6	Automatically collect measurement data reflecting actual experience with a process
A7	Enforce rules to ensure process integrity

Table 3: Goals for automation

gated in [61], where a model of changes, together with its infrastructure support, are presented. Besides the provision of a framework for changes, a primary contribution of this work is the identification of the items of change and of their properties. Another look at the process evolution issue is provided in [67], where the process cycle is used as a basis for process maintenance.

The experience base discussed in Section 3.9 is the centerpiece for any strategy to reuse software processes (I2). Some results towards this direction can be found in the P/MARVEL environment [57]. This MARVEL environment is really a meta-process which allows a process engineer to design a process and verify its behavior in an isolated test environment. Multiple processes can be developed simultaneously, allowing process fragments to be transferred and reused. P/MARVEL is further discussed in Section 5.1.

5 Automation

The objectives of automated development support and automated guidance[11], shown in Table 3, have captured the interest of a large number of researchers. Early research on automation focused on providing a collection of independent file-based tools such as `make`, `vi`, `rcs`, and `sccs`. The invocation of these tools was, however, left up to the user. Software development environments (SDEs) then appeared that were considered "intelligent" since they automated some of the tool invocations and provided a means for storing information, such as the current state of a project, in databases. This "intelligence" was limited, however, because it was mostly fixed for each environment. SDEs were then created that could tailor their behavior based upon the specification of a desired process. These process-centered environments (PCEs) have process engines that "enact" process models, a term used to encompass enforcement, automation, and guidance of the users in carrying out the process.

[11] In this paper, we combine the two goal clusters relating to automation.

Many existing process-centered environments use some form of rules to define software processes, because declarative rules are believed by many researchers to be the most natural way to express at least the local constraints on process steps. Major exceptions include Arcadia [44], which uses an imperative notation called APPL/A [86] based on Ada, HFSP [46], which uses an extension of attribute grammars, and Melmac [21], Slang [5], and Process Weaver[12] [29], which use a form of Petri nets. As described in Section 2, MARVEL plays a central role in the PRS framework through its ability to enact processes. Before presenting MARVEL in more detail, we briefly describe some related PCEs.

Important examples of rule-based PCEs include GRAPPLE [40], which applies a planning system to rules similar in form to those of MARVEL, and Darwin [73], which employs backward chaining in the style of Prolog to enforce a set of "laws" that govern development activities and software changes. Most of the rule-based PCE projects are currently working toward support for multiple users, notably the Common Lisp Framework (CLF) [4], Oikos [3] and Merlin [83]. CLF supports a checkout/merging model but has no central objectbase or process engine. Oikos uses a blackboard to communicate among separate workspaces, so there is somewhat more coordination required than with CLF; Merlin's approach is similar to MARVEL's. To determine whether a PCE is effective (A1), we measure it against the set of requirements defined by Ben-Shaul *et al.* [13], which must be fulfilled by any general PCE. As described in that paper, MARVEL addresses each of the requirements.

5.1 Marvel

The goal of the MARVEL [13, 36] project is to develop a PCE kernel that guides and assists a team of users working on a medium-scale software development effort. The behavior of the generic kernel is tailored by an *administrator* who provides the schema, process model, tool envelopes, and coordination model for a specific project. The *user*, in contrast, generally sees only the resulting environment instance.

A process administrator writes a specification of the project data schema and process model using MSL, MARVEL's process modeling language. The administrator then loads these specifications into the kernel, creating a MARVEL environment instance that supports both the data and process management requirements of the project. The data schema is specified in terms of classes, each of which consists of a set of typed attributes. Existing source code can be *immigrated* from the file system into a MARVEL objectbase using the Marvelizer utility [85].

The administrator defines the process (or workflow) by creating process steps corresponding to individual software development tasks. Each step is encapsulated by a *rule* with a name and typed parameters. The body of a rule consists of a query

[12] Process Weaver is a trademark of Cap Gemini Innovation.

```
compile [?c:CFILE]:
    (and (exists PROJECT ?p  suchthat (ancestor [?p ?c]))
         (forall INC     ?i  suchthat (member   [?p.include  ?i]))
         (forall HFILE   ?h  suchthat (member   [?i.hfiles   ?h]))):

    # 0. If an INC object is archived, don't automatically chain here
    # 1. If this CFILE has been analyzed or
    # ------------------------------------------------------------
    (and no_forward (?i.archive_status = Archived)        # 0.
                    (?c.status = Analyzed))               # 1.

    { COMPILE compile ?c.contents ?c.object_code ?h.contents ?c.history }

    # 0.a Mark CFILE as successfully compiled
    # 0.b Update the object code time stamp
    # 1.  Update status of object.  Chain to outdate rules
    # ------------------------------------------------------------
    (and (?c.status = Compiled)                             # 0.a
         (?c.object_time_stamp = CurrentTime));             # 0.b
    (?c.status = ErrorCompile);                             # 1.
```

Figure 4: CFILE compile rule from C/MARVEL

to bind local variables (for example, the set of `#include` ".h" files for a compilation task); a complex logical condition on the actual parameters and bound variables that must be satisfied prior to initiating the activity of the step; an optional activity in which a software development tool may be invoked; and a set of effects, each of which asserts one of the activity's possible results (if there is no activity, there can be only one effect). Forward and backward chaining over the rules enforce consistency in the objectbase and automate tool invocations; enforcement and automation are the two forms of enaction in MARVEL. This consistency enforcement is exactly the mechanism needed to satisfy A7. The chains between rules form a rule network, with rules as nodes, and chains between rules as edges. A sample rule for compiling a C file (using the `cc` tool) is shown in Figure 4.

Process enaction is mainly user-driven, as opposed to system-driven. The user decides when to request a particular process step and enters a command with the name and actual parameters of the step. MARVEL then selects the "closest" matching rules (there may be more than one) and evaluates each of these rules in turn until it finds one whose condition is already satisfied or can be satisfied through backward chaining. The activity, if any, of this rule is then executed. Afterwards, one of the effects is selected according to a status code returned by the activity, and MARVEL forward chains to any other rules that are implications of this effect. If none of the conditions of the matching rules can be satisfied, however, then the user is informed that it is not possible to undertake that process step at this time. Note that since rules have multiple effects, it may be possible that an attempted backward chain results in an undesired effect, but the chain is not then "undone" because that would be counterproductive (consider a backward chain to generate correct object code by compiling source code that produces syntax error messages instead).

The condition of the rule shown in Figure 4 permits backward chaining from it to ARCHIVE all the include directories of the project (predicate 0), but forward chaining into it is prevented by the `no_forward` directive. Either backward chaining to ANALYZE (using the `lint` tool) the C file itself, or forward chaining into this rule from the ANALYZE rule is permitted—in the latter case the analyzed C file becomes the parameter (predicate 1). Additional details about the rule formalism and its chaining engine are given in [36]. If the compilation is successful, the envelope executing the activity returns a status code of `0`, and the first set of assertions marks the CFILE as `Compiled` and updates the time stamp of the object code. If, however, the compilation fails, a `1` is returned, and the status of the CFILE is marked as `ErrorCompile`. In either case, forward chaining is carried out to rules whose condition is made satisfied by the assertions.

Multiple users of the same environment instance are supported by a client/server architecture [13]. A client provides the user interface, checks the arguments of commands, and executes tool envelopes; the process engine, synchronization management, and objectbase reside in the central MARVEL server. Scheduling is first-come, first-served with rule chains interleaved at the natural breaks provided when clients execute activities. Clients may run on the same or different hosts as the server, but the enveloping facility assumes a shared network file system where the software components under development reside. The external view is illustrated in Figure 5. Additional details about multi-user issues, primarily concurrency control policies specified by the administrator in the coordination model, are found in [6, 34]. MARVEL's support for schema and process evolution, discussed in Section 4.5, is described in [45].

Synchronization among multiple users has three layers. Conventional locking is augmented by a lock compatibility matrix, part of the *coordination model* provided by the process administrator. This matrix provides support for composite objects by an ancestor lock table—a generalization of intention locks. Lock modes for kernel operations (for example, `add`, `delete`), as well as defaults for rule subparts and tool invocations, are also specified. The default concurrency control policy distinguishes between chaining for consistency versus automation purposes [7]. Chains for the purpose of maintaining consistency is mandatory and are treated as atomic, serializable transactions; if a consistency chain encounters an unresolvable lock conflict, the entire chain is aborted (rolled back). In contrast, chains for automation purposes are optional and are treated as sequences of distinct transactions, one for each rule; they can be terminated (and not rolled back) at rule boundaries. A preliminary coordination modeling language specifies scenarios where this default policy can be relaxed to increase concurrency and enhance collaboration [6]. The administrator defines the conflict resolution using primitive operations to `notify` a user, `abort` a rule chain, `suspend` a rule chain until another has completed, or `ignore` the conflict.

Conventional file-oriented tools are integrated into a MARVEL process without source modifications, or even recompilation, through an *enveloping* language [32]. The rule activity indicates the tool and envelope name, with input literals and attributes to be

supplied as arguments as well as output variables for binding to any returned results; an implicit status code selects the actual effect from among those given in the rule. The body of an envelope is a shell script, written in one of the conventional UNIX shell languages: `sh`, `ksh`, or `csh`.

Achievements in automation

CASE tools were heralded as the solution to the software crisis when they were first developed; in retrospect, we have not seen this to be true. They have, however, proved to be extremely useful. The MARVEL PCE described here shows that processes can be, to some degree, automated to provide assistance to the users. MARVEL has been used to enact a variety of processes. Our two code production environments, OZ/MARVEL and C/MARVEL, support the development of software using C. These environments manage code bases of 200,000 and 150,000 lines of code, respectively, and support teams of programmers. The OZ/MARVEL process was designed within our process development environment, P/MARVEL, by taking the C/MARVEL process and reusing process fragments, tailoring, and adding new features. P/MARVEL allows multiple processes to be developed simultaneously and is a limited attempt at constructing a process repository (A3). DOC/MARVEL is a document preparation environment using LaTeX which the MARVEL group used to produce a four-volume set of manuals totaling over four hundred pages. In this environment, as many as five technical writers were working concurrently.

To provide guidance and reference material to the user (A2), the MARVEL client allows the user to graphically browse the rule network to see the process flow. For example, Figure 6 contains a fragment of a rule network which shows what could happen if the user fires the EDIT rule on an object from the CFILE class. There is an atomic consistency chain of three rules (edges labeled **C**) which will occur if the user makes a change to the CFILE object. Once this chain has completed, an automation chain of two rules occurs, to ANALYZE and COMPILE the CFILE object.

MARVEL's primary goal is to enact a process model (A4); it does so in a user-driven fashion. That is, when the user completes a step of the process (by firing a rule) MARVEL uses forward chaining to carry out other process steps whose prerequisite has become satisfied. MARVEL also executes backward chaining when a user initiates a process step whose prerequisite is not satisfied. In this case, the goal-directed backward chain attempts to satisfy the prerequisite by finding other process steps that, if executed, would make the original prerequisite satisfied.

MARVEL has no built-in mechanism to collect statistics on the actual experience of a process, but the administrator can design tool envelopes that record information about the process (A6). For example, the OZ/MARVEL process has been written so that information is recorded every time the EDIT rule is fired on an object or a new

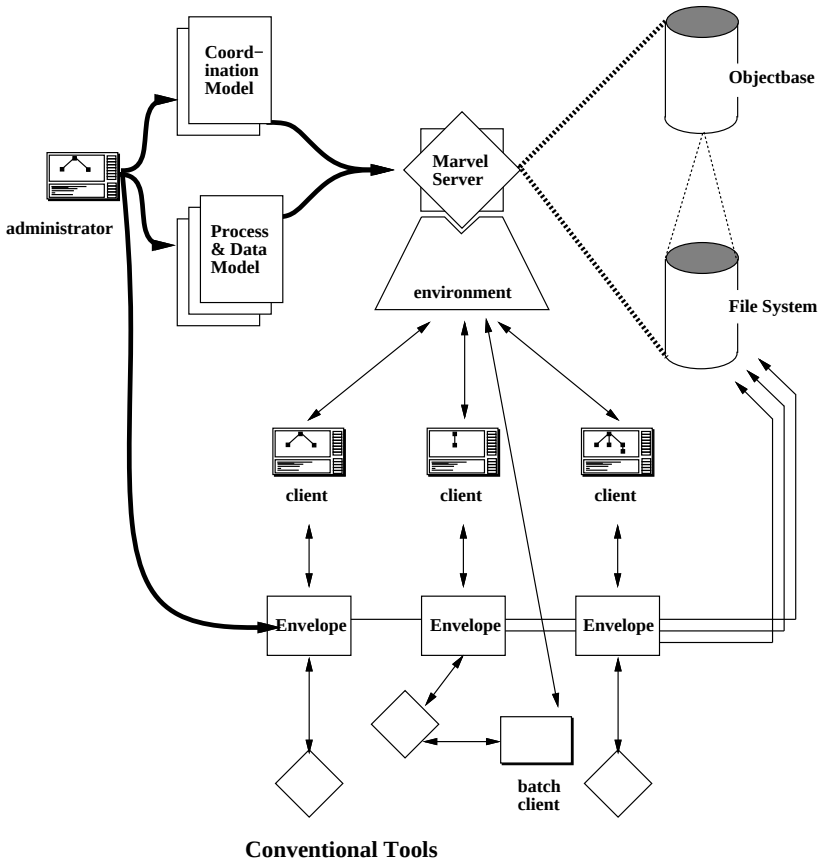

Figure 5: Generic MARVEL environment

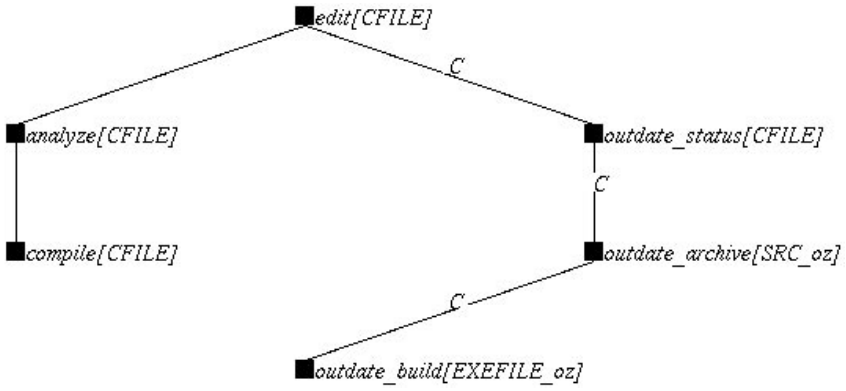

Figure 6: Sample rule network

M1	Develop a project-specific software process to accommodate the attributes of a particular project, such as its product, or organizational environment
M2	Reason about attributes of software creation or evolution
M3	Support development of plans for the project (forecasting)
M4	Monitor, manage, and coordinate the process
M5	Provide a basis for process measurement, such as definition of measurement points within the context of a specific process

Table 4: Goals for process management

version of an object is released by the configuration management system. Using this information, for example, the administrator can determine "hot spots" in the code where frequent changes are made. The enacted process can record as many statistics about itself as the administrator requires.

MARVEL does a good job of supporting cooperative work among small- to medium-sized groups of individuals, since it automates process-specific concurrency control policies while still supporting conventional transactions (that is, isolation while process fragments are in progress) as the default. But it does not address collaboration among autonomous teams (A5). The MARVEL team is currently working on a new architecture to support such collaboration among possibly geographically dispersed across a wide area network such as the Internet [12].

6 Process Management

The final goal cluster (Table 4) is concerned with the planning, control, and operational management of software processes. Process management is a specific discipline which supports Total Quality Management (TQM) [33] and is needed to support continuous improvement of defined processes. Process management relies on feedback—from measurements, assessments, and other analyses—to guide improvement activities. Process management must also nurture an organizational culture that subscribes to a process-driven approach to software engineering and continuous, on-going improvement.

The first step to process management is the selection of software processes most appropriate for the individual needs of a project and its organizational structure. In Section 4.2, we showed how OPT determines the match between a particular process and the company's organization. Boehm [14, 15] describes a "software process model generator" to aid in selecting the type of process for a given software effort based on a decision table and various life cycle processes. Another important element of planning involves the development of a suitable process from a repertoire of components. The AI planning paradigm used by GRAPPLE [40] provides an automated goal-driven

approach to this problem; in GRAPPLE, process components are selected from an existing set and organized automatically to satisfy stated goals. The SFINX software process model library [18] offers a hierarchical organization of processes into layers, allowing components to be "plugged" into "frames" provided by the next higher layer of the architecture. Other works, such as [11, 55], explore reuse-based mechanisms for developing project-specific software processes.

Once a generic process model is selected it needs to be tailored to fit individual projects. Kellner, Phillips, and Gates [55, 51] have characterized evolutionary approaches to process design; the two major categories are derivative and constructive, with further characterizations based upon the source of the primary starting process and of the changes being integrated (as improvements, for tailoring or customization).

Process models can be useful in suggesting measurement points and metrics that can be used as status criteria (M5). Indeed, it is valuable for a model to include definitions of precisely what is to be measured and when, to whom it is to be reported, how it is to be used, and so forth [81, 62]. Such data should be collected and retained over time to reflect past experience and be used as a foundation for future planning [49]. One problem with measurements in a process model is that the actual performance of a process might differ from its model. Consequently, it is important to record the performance of the process (for example, using a process trace), and consider all measurements against that trace. The history of past processes and results can be stored in a "process database" [77].

Software process models can be used for planning schedules and analysis. Kellner [42, 50] describes modeling support for management planning and control (including monitoring, recording progress, and re-planning), examining both general needs and specific capabilities available with Statemate-based modeling. Support for both planning and re-planning during process performance was found to be an essential capability for managerial support; it should enable planning schedules, costs, and resource needs with and without resource constraints, and accommodate deterministic or stochastic information.

In [50], the Statemate-based modeling approach is extended to incorporate automated, quantitative simulations that are used to derive schedules, required work effort, and required staffing profiles. Cases of both point estimates (deterministic modeling) and uncertain estimates (stochastic modeling) are discussed, and resource constraints are also considered. This modeling approach offers the distinct advantage of smooth integration of representation, analysis, and forecasting capabilities; the quantitative simulations can be run by adding relatively straightforward information to an existing model, with no need to otherwise modify the existing model and with no changes to the visual representation. This integration is important, since this approach has been successfully used to model and analyze various large-scale real-world software processes.

The Statemate-based modeling approach also offers distinct advantages over traditional project management approaches such as the critical path method and PERT. The process models are more general, provide enhanced visibility into behavior, and highlight the importance of feedback loops in software processes. Moreover, they are amenable to resource constraints and full Monte Carlo simulation analysis [50]. Akhavi and Wilson have reported practical applications of these techniques at Rockwell International [2].

Other work has also addressed some of these needs: SPMS [26] contains a project management tool using the critical path method for project planning, system dynamics have been productively employed to forecast project level plans and the impact of changes [1, 58], Articulator [72] uses AI scheduling techniques from production systems, and Prism [59] incorporates process simulations.

6.1 Key Process Areas

A major component of the SEI research at CAS is a study that examines the relationships and trade-offs among major dimensions of software total-quality: life cycle cost, field defects, and customer satisfaction. Various management-level questions are addressed, such as: *How do cost and satisfaction vary with defects? How do defects and cost vary with front-end investment during development?* Moreover, the research explores the factors underlying the differences and interrelationships seen. This work attempts to determine the key drivers of software total quality, and find how they impact these major dimensions. For example:

- What is the value/impact of CASE design tools, automated testing tools, peer reviews, configuration management, or methodology training?
- What is the value/impact of each of the Key Process Areas (KPAs) defined in the SEI CMM [76, 77]?

In terms very specific to software processes, practical questions regarding the value and impact of software process differences are addressed, such as: *What value has typically been obtained in practice from implementing a given KPA? Based on the actual experiences of others, what value can an organization expect to achieve from implementing a given KPA?*. Answers to these questions can be very useful in enabling management to obtain rough estimates of the types and magnitudes of the values they might achieve from putting selected KPAs into practice. KPAs can be prioritized by these estimates for strategic attention, based on anticipated value. This would be quite appropriate for an organization that does not yet possess deep insight and understanding into their current processes, and is therefore not in a position to use the detailed quantitative process modeling techniques discussed in Section 4.3.

It would be useful to determine typical values achieved from individual KPAs, even at a broad average level. However, such a broad average would be a relatively crude estimate to apply to any single, specific case. For the purposes of a single organization or project, a model that takes other important factors (for example, type of application, personnel capability, or size of product) into account would provide a more refined estimate of likely value to that specific organization.

Krishnan [27, 56] is conducting a field study based on data collected from the IBM Software Solutions Toronto Laboratory. It is important to recognize that this is an empirical investigation of actual results achieved in an actual organization. This directly addresses a vital real-world concern of reporting typical actual experience, not just theoretical possibility or best-case performance. Econometric statistical techniques (such as multiple regression and ordered probit analyses) are being used to analyze the data and test hypotheses.

The underlying conceptual model is relatively straightforward. Instead of considering software quality as being limited solely to conventional error or defect counts, software total-quality is based on "delighting the customers[13]". Second, this research hypothesizes that software total-quality is determined by the interactions of product, people, technology, process, and environment factors. These factor groupings comprise a "system" of drivers that determine total-quality [54]. Finally, the quality resulting from the driver interactions is measured along certain selected dimensions, which in this work include life cycle cost, field quality (based on defects), and customer satisfaction.

In Phase I of this effort, the focus has been on understanding the interrelationships and trade-offs among the measured result dimensions. Some specific research questions addressed are:

- Is quality "free" in the context of software? (Crosby [19] claims that it is)
- Does higher deployment of resources in the early phases of systems development pay off in terms of quality?
- How does product type affect the relationship between cost and quality?

Early Phase I results are as follows. The empirically observed relationship between cost and quality indicates that as quality increases (lower shipped defect density), long-run unit cost (life cycle cost per KLOC) decreases, but at a decreasing rate. This confirms Crosby's assertion, at least over the range of quality observed in the data set. The relationships among resource deployment strategy (higher or lower investment early in the life cycle), quality, and product size have also been examined. The results indicate that (1) higher front-end investment pays off in terms of higher quality, and (2) the payoff in quality due to higher front-end investment is more

[13]John Schwarz, IBM Software Solutions Toronto Laboratory.

pronounced for larger products. These results suggest a managerial action to invest in the early stages of the life cycle, especially for larger products; this appears to lead to higher quality and correspondingly lower costs.

Phase II will broaden the analysis to include key drivers of software quality and their impact on the measured dimensions of software quality. The measured result dimensions will be the same as in Phase I. The drivers will be specific factors reflecting the important characteristics of process, people, technology, product, and environment. The process factors will indicate degree of conformance with each of the KPAs identified in the CMM. The other factors will measure specific characteristics of the other four driver groupings; the COCOMO cost drivers [16] provide examples of the factors to be examined. Phase II will identify key driver factors, and the empirical relationships among them and the major dimensions of software quality.

From a managerial perspective, the Phase II model will allow a manager to examine the result dimensions and a proposed change (for example, emphasize increasing field quality), and then gain insight into what changes in the driver factors would yield the desired results. Those factors which the manager can control (such as the process factors) can then be manipulated to obtain the desired results in practice. This illustrates how the work will yield actionable managerial implications.

The Phase II work also yields broad estimates of value. The model itself will reveal the value of process (and people, technology, etc.) factors in terms of life cycle cost, fielded quality, and customer satisfaction. Using average values for the non-process factors will yield the average value seen in the data set for each KPA (on an individual, incremental basis). For a specific project, one would estimate the values to be seen for the non-process factors, and the model then predicts the value to be obtained for that project from each KPA.

Achievements in management

Software process models can enhance process management in a number of valuable ways, as discussed in this section. The feedback provided to management by measurements helps characterize current problems and areas of opportunity, and may also lead to further process improvements (M3); thus this process management goal cluster also relates to the process improvement cluster. Better process management will also be greatly aided by accomplishing some of the other goals, such as increased understanding, better training, conformity to process definitions, and evaluation of potential improvements. The process cycle [60] and OPT [84] can be used together to create project-specific software processes (M1) that are best suited for a particular product. MARVEL [13] addresses M4 by allowing the process manager to observe the behavior of the process and define coordination policies for the multiple users [6]. MARVEL provides mechanisms that allow the administrator to improve and evolve

Objective		*Publications and Results*
Understanding	U1-U5	Elicit [65, 64, 37], DMP, SEI Statemate approach [49, 52, 53, 20, 55, 48]
Improvement	I1	OPT [84]
	I2	P/MARVEL, [55]
	I3	[50, 80, 79, 42]
	I4	[80, 79], Evolver [45]
	I5	TIM [17], [52, 53, 49]
	I6	OPT, [55, 27]
	I7	Evolver [45], Prism [59], [51, 20]
Automation	A1-A7	MARVEL[13],OZ [12], P/MARVEL, [55]
Management	M1	[51, 55], OPT
	M2	Process Cycle [60], [20, 42, 49, 27, 56]
	M3	[42, 50, 27, 56]
	M4	MARVEL, [50]
	M5	[62]

Table 5: Summary table

the process as feedback from the performers is acquired.

7 Conclusion

The PRS consortium was initiated at a time when process-awareness was very low and has matured as the research on software processes has increased. By providing a holistic vision of software processes, PRS can affect many aspects of processes, as we have seen in this paper. Table 5 summarizes the individual publications and research results corresponding to the twenty-four subgoals. This table concisely shows the breadth of process research performed by PRS participants.

After some reflection on the nature of these objectives, we can create a hierarchy as shown in Figure 7. The base objective, Understanding, is a precursor to all other objectives. Improvement and Automation are independent of each other, and reside at the same level in the hierarchy, while Management, at the highest level, is dependent upon all other objectives. This hierarchy portrays the dependencies between the objectives and reflects the depth of PRS research. The presentation of the Understanding objective in Section 3, for example, is much more comprehensive than the discussion on Management in Section 6. We plan to continue PRS research addressing each of these four objectives, ultimately providing real-world solutions to real-world problems.

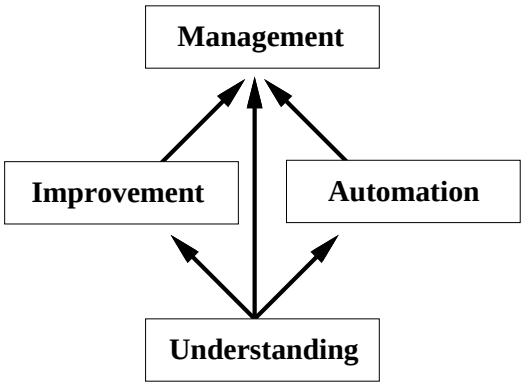

Figure 7: Hierarchy of Objectives

Acknowledgment

The authors would like to thank the current PRS students for their continuing hard work and research: Tilmann Bruckhaus, M. S. Krishnan, David Raffo, Carolyn B. Seaman. Previous CAS students involved with PRS include Dirk Höltje, Steven Popovich, and Andrew Z. Tong. Sincere thanks are also due to research students in projects other than PRS but who have contributed greatly to our process work in general: Israel Z. Ben-Shaul, Khaled El Emam, Won-Kook Hong, Graciela Perez, Khalid Sherdil, Peter D. Skopp, Kamel Toubache and Josee Turgeon. We would like to thank the CAS staff – in particular Jacob Slonim – for providing an open and challenging research environment.

References

[1] Tarek K. Abdel-Hamid and Stuart E. Madnick. *Software Project Dynamics: An Integrated Approach.* Prentice-Hall, 1991.

[2] Mina Akhavi and Wendy Wilson. Dynamic simulation of software process. In *5th Software Engineering Process Group National Meeting* (Costa Mesa, CA, April 1993). Software Engineering Institute; Carnegie Mellon University, 1993. Presentation in Session Three, Advanced Track.

[3] V. Ambriola, P. Ciancarini, and C. Montangero. Software process enactment in Oikos. In Richard N. Taylor, editor, *4th ACM SIGSOFT Symposium on Software Development Environments*, pages 183–192, Irvine CA, December 1990. Special issue of *Software Engineering Notes*, 15(6).

[4] Robert Balzer. Tolerating inconsistency. In *13th International Conference on Software Engineering*, pages 158–165, Austin TX, May 1991. IEEE Computer Society Press.

[5] Sergio Bandinelli and Alfonso Fuggetta. Computational reflection in software process modeling: the SLANG approach. In *15th International Conference on Software Engineering*, pages 144–154. IEEE Computer Society Press, May 1993.

[6] Naser S. Barghouti. *Concurrency Control in Rule-Based Software Development Environments*. PhD thesis, Columbia University, February 1992. CUCS-001-92.

[7] Naser S. Barghouti. Supporting cooperation in the MARVEL process-centered SDE. In Herbert Weber, editor, *5th ACM SIGSOFT Symposium on Software Development Environments*, pages 21–31, Tyson's Corner VA, December 1992. Special issue of *Software Engineering Notes*, 17(5), December 1992.

[8] V. R. Basili, Gianluigi Caldiera, and Giovanni Cantone. A reference architecture for the component factory. *ACM Transactions on Software Engineering and Methodology*, 1(1):53–80, January 1992.

[9] V. R. Basili and H. D. Rombach. The TAME project: Towards improvement-oriented software environments. *IEEE Transactions on Software Engineering*, 14(6):758–773, June 1988.

[10] V. R. Basili and H. D. Rombach. Tailoring the software process to project goals and environments. In *9th International Conference on Software Engineering*, pages 345–357, Monterey, CA, April 1987. IEEE Computer Society Press.

[11] V. R. Basili and H. D. Rombach. Support for comprehensive reuse. *Software Engineering Journal*, 6(5):303–316, September 1991.

[12] Israel Z. Ben-Shaul and Gail E. Kaiser. A paradigm for decentralized process modeling and its realization in the OZ environment. In *16th International Conference on Software Engineering*, Sorrento, Italy, May 1994. IEEE Computer Society Press. In press.

[13] Israel Z. Ben-Shaul, Gail E. Kaiser, and George T. Heineman. An architecture for multi-user software development environments. *Computing Systems, The Journal of the USENIX Association*, 6(2):65–103, Spring 1993.

[14] B. Boehm and F. Belz. Experience with the spiral model as a process model generator. In Dewayne Perry, editor, *5th International Software Process Workshop: Experience with Software Process Models*, pages 43–45, Kennebunkport ME, October 1989. IEEE Computer Society Press.

[15] Barry Boehm. What we really need are process model generators. In *11th International Conference on Software Engineering*, page 397, Pittsburgh PA, May 1989. IEEE Computer Society Press.

[16] Barry W. Boehm. *Software Engineering Economics*. Prentice-Hall, 1981.

[17] Tilmann Bruckhaus. The impact of inserting a tool into a software process. In Ann Gawman, W. Morven Gentleman, Evelyn Kidd, Per-Åke Larson, and Jacob Slonim, editors, *1993 CASCON Conference*, pages 250–264, Toronto, Ontario, Canada, October 1993. IBM Canada Ltd. Laboratory and National Research Council Canada.

[18] Giuseppe Bux and Giovanni Marzano. Library of predefined software process models as support for software factory design: The SFINX proposal. In *5th International Workshop on Computer-Aided Software Engineering* (Montréal, Quebec, July, 1992), pages 176–178. IEEE Computer Society Press, 1992.

[19] Philip B. Crosby. *Quality is Free*. McGraw-Hill, 1979.

[20] Bill Curtis, Marc I. Kellner, and James W. Over. Process modeling. *Communications of the ACM*, 35(9):75–90, September 1992.

[21] Wolfgang Deiters and Volker Gruhn. Managing software processes in the environment MELMAC. In Richard N. Taylor, editor, *4th ACM SIGSOFT Symposium on Software Development Environments*, pages 193–205, Irvine CA, December 1990. Special issue of *Software Engineering Notes*, 15(6).

[22] Khaled El Emam and Nazim H. Madhavji. A study of the factors that affect the success of the requirements engineering process. Technical Report SE-94.2, School of Computer Science, McGill University, Montréal, 1994.

[23] Khaled El Emam, Nazim H. Madhavji, and Kamel Toubache. Empirically driven improvements of generic process models. In Wilhelm Schäfer, editor, *8th International Software Process Workshop*, pages 61–65, Wadern, Germany, March 1993.

[24] Khaled El Emam, Nadir Moukheiber, and Nazim H. Madhavji. An empirical evaluation of the G/Q/M method. In Ann Gawman, W. Morven Gentleman, Evelyn Kidd, Per-Åke Larson, and Jacob Slonim, editors, *1993 CASCON Conference*, pages 265–289, Toronto, Ontario, Canada, October 1993. IBM Canada Ltd. Laboratory and National Research Council Canada.

[25] David Harel *et al.* Statemate: A working environment for the development of complex reactive systems. *IEEE Transactions on Software Engineering*, 16(4):403–414, April 1990.

[26] Herb Krasner *et al.* Lessons learned from a software process modeling system. *Communications of the ACM*, 35(9):91–100, September 1992.

[27] Mayuram S. Krishnan *et al.* Cost, Quality, and User Satisfaction of Software Products: A field study. In *Field Studies in Quality Management Conference* (Held at the Simon School, University of Rochester, Rochester, NY, March 26-27, 1993), In Press.

[28] Peter H. Feiler and Watts S. Humphrey. Software process development and enactment: Concepts and definitions. Technical Report SEI-92-TR-4, Software Engineering Institute, September 1992.

[29] Christer Fernström. PROCESS WEAVER: Adding process support to UNIX. In *2nd International Conference on the Software Process: Continuous Software Process Improvement*, pages 12–26, Berlin, Germany, February 1993. IEEE Computer Society Press.

[30] P. Fowler and S. Rifkin. Software Engineering Process Group Guide. Technical Report CMU/SEI-90-TR-24, Software Engineering Institute, Carnegie Mellon University, September 1990.

[31] D. A. Garvin. How the Baldridge Award Really Works. *Harvard Business Review*, 69(6):80–93, November-December 1991.

[32] Mark A. Gisi and Gail E. Kaiser. Extending a tool integration language. In Mark Dowson, editor, *1st International Conference on the Software Process: Manufacturing Complex Systems*, pages 218–227, Redondo Beach CA, October 1991. IEEE Computer Society Press.

[33] J. Hauser and D. Clausing. The House of Quality. *Harvard Business Review*, 66(3):63–73, May-June 1988.

[34] George T. Heineman. A transaction manager component for cooperative transaction models. CUCS-017-93, Columbia University Department of Computer Science, July 1993. PhD Thesis Proposal.

[35] George T. Heineman. Automatic translation of process modeling formalisms. Technical Report CUCS-036-93, Columbia University Department of Computer Science, November 1993.

[36] George T. Heineman, Gail E. Kaiser, Naser S. Barghouti, and Israel Z. Ben-Shaul. Rule chaining in Marvel: Dynamic binding of parameters. *IEEE Expert*, 7(6):26–32, December 1992.

[37] D. Höltje. Building Software Process Models Using the Elicit Meta Process: A Case Study. Master's Thesis, FernUniversität Hagen, Germany (This research was carried out at McGill University, Montréal, Canada, and at IBM Canada Toronto Lab), June 1993.

[38] W. Hong and N. H. Madhavji. A Method for Building Generic Process Models. Technical Report SE-94.1, School of Computer Science, McGill University, Montréal, January 1994.

[39] C. C. Huff. Elements of a realistic case tool adoption budget. *Communications of the ACM*, 35(4):45–54, April 1992.

[40] Karen E. Huff and Victor R. Lesser. A plan-based intelligent assistant that supports the software development process. In Peter Henderson, editor, *ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments*, pages 97–106, Boston MA, November 1988. ACM Press. Special issue of *Software Engineering Notes*, 13(5).

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

[42] Watts S. Humphrey and Marc I. Kellner. Software process modeling: Principles of entity process models. In *11th International Conference on Software Engineering* (Pittsburgh, PA, May, 1989), pages 331–342. IEEE Computer Society Press, 1989.

[43] *International Standard ISO 9000-3, Guidelines for the Application of ISO 9001 to the Development, Supply and Maintenance of Software*, Geneva, 1991. International Organization for Standardization.

[44] R. Kadia. Issues encountered in building a flexible software development environment. In Herbert Weber, editor, *5th ACM SIGSOFT Symposium on Software Development Environments*, pages 169–180, Tyson's Corner VA, December 1992. Special issue of *Software Engineering Notes*, 17(5), December 1992.

[45] Gail E. Kaiser, Israel Z. Ben-Shaul, George T. Heineman, and Wilfredo Marrero. Process evolution for the MARVEL environment. Technical Report CUCS-047-92, Columbia University Department of Computer Science, April 1993.

[46] Takuya Katayama. A hierarchical and functional software process description and its enaction. In *11th International Conference on Software Engineering*, pages 343–352, Pittsburgh PA, May 1989. IEEE Computer Society Press.

[47] R. K. Keller and N. H. Madhavji. A comprehensive process model for studying software process papers. In *15th International Conference on Software Engineering*, pages 78–88, Baltimore, MD, May 1993. IEEE Computer Society Press.

[48] Marc I. Kellner. Representation formalisms for software process modeling. In Colin Tully, editor, *4th International Software Process Workshop: Representing and Enacting the Software Process*, pages 93–96. ACM Software Engineering Notes, June 1989. Special issue of *Software Engineering Notes*, 14(4).

[49] Marc I. Kellner. Software process modeling: Value and experience. In *SEI Technical Review,* Software Engineering Institute, pages 23–54. Carnegie Mellon University, 1989.

[50] Marc I. Kellner. Software process modeling support for management planning and control. In Mark Dowson, editor, *1st International Conference on the Software Process: Manufacturing Complex Systems*, pages 8–28, Redondo Beach CA, October 1991. IEEE Computer Society Press.

[51] Marc I. Kellner and Linda Parker Gates. Evolution of software processes. In *International Workshop on the Evolution of Software Processes* (Mt. St. Hilaire, Quebec), January 1993.

[52] Marc I. Kellner and Gregory A. Hansen. Software Process Modeling. Technical Report CMU/SEI-88-TR-9, DTIC: ADA197137, Software Engineering Institute, Carnegie Mellon University, May 1988.

[53] Marc I. Kellner and Gregory A. Hansen. Software process modeling: A case study. In *22nd Annual Hawaii International Conference on System Sciences*, volume II, pages 175–188, Kona HI, January 1989. IEEE Computer Society Press.

[54] Marc I. Kellner and James W. Over. A software quality improvement framework. In *Software Engineering Symposium - 1992* (Held at Milan, Italy, June 10-11, 1992). Advanced Software Technology - Olivetti Information Services, 1992.

[55] Marc I. Kellner and Richard W. Phillips. Practical technology for process assets. In Wilhelm Schäfer, editor, *8th International Software Process Workshop*, pages 107–112, Wadern, Germany, March 1993.

[56] Mayuram S. Krishnan. Cost, Quality, and User Satisfaction of Software Products: An empirical analysis. In Ann Gawman, W. Morven Gentleman, Evelyn Kidd, Per-Åke Larson, and Jacob Slonim, editors, *1993 CASCON Conference*, pages 400–411, Toronto, Ontario, Canada, October 1993. IBM Canada Ltd. Laboratory and National Research Council Canada.

[57] Programming Systems Laboratory. Marvel 3.1 Administrator's manual. Technical Report CUCS-009-93, Columbia University Department of Computer Science, March 1993.

[58] Chi Y. Lin and Rueven R. Levary. Computer-aided software development process design. *IEEE Transactions on Software Engineering*, 15(9):1025–1037, September 1989.

[59] N. H. Madhavji. The Prism model of changes. In *13th International Conference on Software Engineering*, pages 93–96, Austin TX, May 1991. IEEE Computer Society Press.

[60] N. H. Madhavji. The process cycle. *Software Engineering Journal*, 6(5):234–242, September 1991.

[61] N. H. Madhavji. Environment evolution: The Prism model of changes. *IEEE Transactions on Software Engineering*, 18(5):380–392, May 1992.

[62] N. H. Madhavji, J. Botsford, T. Bruckhaus, and K. El Emam. Measurements based on process and context models. In V. R. Basili, D. H. Rombach, and R. Selby, editors, *International Workshop on Experimental Software Engineering*

Issues, pages 67–72, Dagstuhl, Warden, Germany, September 1992. Springer-Verlag.

[63] N. H. Madhavji, D. Höltje, W. Hong, and T. Bruckhaus. An Empirically Improved Process for Modelling Software Processes. Technical Report SE-93.3, School of Computer Science, McGill University, Montréal, August 1993.

[64] N. H. Madhavji, D. Höltje, W. Hong, and T. Bruckhaus. Eliciting a Formal Model of A Requirements Engineering Process. Technical Report SE-93.1, School of Computer Science, McGill University, Montréal, August 1993.

[65] N. H. Madhavji, W. K. Hong, T. Bruckhaus, and J. E. Botsford. Elicit: A meta process and supporting tool for eliciting software process models. Technical Report SE-92.4, McGill University, September 1992.

[66] N. H. Madhavji, K. Toubache, and W. Hong. Towards engineering reliable software processes. In *International Software Quality Exchange (ISQE92)*, pages 4B–1–30, San Francisco, CA, March 1992. Juran Institute, Inc. and the Rocky Mountain Institute of Software Engineering (rMise).

[67] N. H. Madhavji, K. Toubache, and W. Hong. A framework for process maintenance. In *IEEE 1992 Conference on Software Maintenance*, pages 245–254, Orlando, FL, November 1992.

[68] N. H. Madhavji, K. Toubache, and W. Hong. Communications and iterations in the process cycle. In *7th International Software Process Workshop*, pages 91–93, Yountville, CA, October 1991. IEEE Computer Society Press.

[69] N. H. Madhavji, K. Toubache, and E. Lynch. The IBM-McGill project on software process. In *1991 CASCON Conference*, pages 95–109, Toronto, Ontario, Canada, October 1991. IBM Canada Ltd. Laboratory.

[70] N. M. Madhavji and Wilhelm Schäfer. Prism – Methodology and process-oriented environment. *IEEE Transactions on Software Engineering*, 17(12):1270–1283, December 1991.

[71] Peiwei Mi and Walt Scacchi. A knowledge-based environment for modeling and simulating software engineering processes. *IEEE Transactions on Knowledge and Data Engineering*, 2(3):283–294, September 1990.

[72] Peiwei Mi and Walt Scacchi. Modeling articulation work in software engineering processes. In *1st International Conference on the Software Process* (Redondo Beach, CA, USA, October, 1991), pages 188–201. IEEE Computer Society Press, 1991.

[73] Naftaly H. Minsky. Law-governed systems. *Software Engineering Journal*, 6(5):285–302, September 1991.

[74] Leon Osterweil. Software processes are software too. In *9th International Conference on Software Engineering*, pages 2–12. IEEE Computer Society Press, March 1987.

[75] M. Paulk, B. Curtis, M. Chrissis, and C. Weber. Capability Maturity Model, version 1.1. *IEEE Software*, pages 18–27, July 1993.

[76] Mark C. Paulk, Bill Curtis, Mary Beth Chrissis, and Charles V. Weber. Capability Maturity Model for Software, Version 1.1. Technical Report CMU/SEI-93-TR-24, Software Engineering Institute; Carnegie Mellon University, February 1993.

[77] Mark C. Paulk, Charles V. Weber, Suzanne M. Garcia, Mary Beth Chrissis, and Marilyn Bush. Key practices of the Capability Maturity Model, Version 1.1. Technical Report CMU/SEI-93-TR-25, Software Engineering Institute, Carnegie Mellon University, February 1993.

[78] Graciela Perez, Khaled El Emam, and N. H. Madhavji. A method for the evaluation of congruence and its application to software process design and customization. Technical Report SE-93.4, School of Computer Science, McGill University, Montréal, 1993.

[79] David M. Raffo. Assessing the impact of potential process changes for large scale software development using process modeling. Working Paper WP#1993-14, Graduate School of Industrial Administration, Carnegie Mellon University, April 30, 1993.

[80] David M. Raffo. Evaluating the impact of process improvements quantitatively using process modeling. In Ann Gawman, W. Morven Gentleman, Evelyn Kidd, Per-Åke Larson, and Jacob Slonim, editors, *1993 CASCON Conference*, pages 290–313, Toronto, Ontario, Canada, October 1993. IBM Canada Ltd. Laboratory and National Research Council Canada.

[81] H. Dieter Rombach and Leo Mark. Software process & product specifications: A basis for generating customized software engineering information bases. In *22nd Annual Hawaii International Conference on System Sciences*, volume II, pages 165–174, Kona HI, January 1989. IEEE Computer Society Press.

[82] Kristine D. Saracelli and Kurt F. Bandat. Process automation in software application development. *IBM Systems Journal*, 32(3):376–396, 1993.

[83] Wilhelm Schäfer, Burkhard Peuschel, and Stefan Wolf. A knowledge-based software development environment supporting cooperative work. *International Journal on Software Engineering & Knowledge Engineering*, 2(1):79–106, March 1992.

[84] Carolyn B. Seaman. OPT: Organization and process together. In Ann Gawman, W. Morven Gentleman, Evelyn Kidd, Per-Åke Larson, and Jacob Slonim, editors,

1993 CASCON Conference, pages 314–324, Toronto, Ontario, Canada, October 1993. IBM Canada Ltd. Laboratory and National Research Council Canada.

[85] Michael H. Sokolsky and Gail E. Kaiser. A framework for immigrating existing software into new software development environments. *Software Engineering Journal*, 6(6):435–453, November 1991.

[86] S. M. Sutton, Jr. APPL/A*: A Prototype Language for Software-Process Programming*. PhD thesis, University of Colorado, August 1990.

[87] *Conference on transferring software engineering tool technology*, Piscataway, NJ, November 1987. Available from IEEE Service Center (cat # 88TH0218-8).

[88] U.S. Department of Defense. *(MIL-HDBK-347) Military Handbook: Mission-Critical Computer Resources Software Support*, May 1990.

About the Authors

George T. Heineman is a PhD candidate in the Computer Science Department at Columbia University. His research interests include cooperative transactions, software technology, and the intersection of process centered environments with database technology. He received his BA degree in computer science from Dartmouth College, and his MS degree from Columbia University. He is a member of IEEE and ACM. He can be reached at the Department of Computer Science, 450 CS Building, Columbia University, New York, NY 10027, email: heineman@cs.columbia.edu.

John E. Botsford is a Research Staff Member at IBM Canada Ltd.'s Centre for Advanced Studies in the Toronto Laboratory. He joined IBM in 1964 and worked in two branch offices before moving to the Toronto Lab in late 1967. Since then he has worked on a large number of development projects and was an instructor in the Education Department for three years.

Gianluigi Caldiera is on the faculty of the University of Maryland Institute for Advanced Computer Studies where he coordinates projects on software quality and reuse. He is also associated with the Software Engineering Laboratory of NASA Goddard Space Flight Center, Greenbelt, Maryland. He has participated, as both project leader and project reviewer, with the European Strategic Program for Information Technology (ESPRIT). His research and professional activities are in software engineering, with special focus on software quality assurance and management, software metrics, software reuse and software factories. He has worked on these topics with industry and government in both the USA and Europe. He has authored and co-authored many papers in international journals and conferences.

Gail E. Kaiser is an Associate Professor of Computer Science and Director of the Programming Systems Laboratory at Columbia University. She was selected for an

IBM Research Initiation Grant in Complex Information Systems in 1988. Prof. Kaiser has published over 80 papers in a range of areas, including software development environments, software processes, extended transaction models, object-oriented languages and databases, and parallel and distributed systems. Prof. Kaiser is an associate editor of the journal ACM Transactions on Software Engineering and Methodology, and serves on numerous program committees for conferences as well as reviewing for conferences, journals, NSF and NSERC. She received her PhD and MS from CMU and her ScB from MIT. She is a member of AAAI and ACM and a senior member of IEEE.

Marc I. Kellner is employed as a senior scientist at the Software Engineering Institute (SEI) at Carnegie Mellon University in Pittsburgh, Pennsylvania, USA. Kellner has pioneered and led much of the research on software process modeling conducted at the SEI, and has published several papers on software process issues. Prior to joining the SEI, Kellner was a professor at Carnegie Mellon University, where he established and directed a degree program in Information Systems. He received his Ph.D. in Systems Sciences (specializing in MIS) from the Graduate School of Industrial Administration at Carnegie Mellon University. His research interests include software processes, software process modeling, software maintenance, and quality management for software. He is a member of the Association for Computing Machinery, the IEEE Computer Society, and The Institute of Management Sciences. Kellner can be contacted at the Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA 15213-3890. TEL: (412) 268-7721 FAX: (412) 268-5758 email: mik@sei.cmu.edu

Nazim H. Madhavji obtained his Ph.D. degree in 1980 from the University of Manchester (U.K.). He joined McGill University, Montréal, Quebec, Canada in 1983, where he is a professor at the School of Computer Science. In 1993, he was appointed as Research Director of the Software Process Programme at *Centre de recherche informatique de Montréal*, (CRIM). His research interests are in software engineering, software processes, project management, software environments, and programming languages. He leads ProM Canada, the Canadian component of a Canada-Germany joint research project in software processes involving McGill/CRIM, GMD (German National Research Centre in Computer Science) and FernUniversität, Hagen. He is a principal investigator in a Reverse Engineering Project, involving McGill University, University of Toronto, University of Victoria, and IBM Canada. He has chaired, and co-chaired, numerous program committees, workshops, and conference sessions. He is on the Advisory Editorial Board of the Journal of Software Maintenance. He has led Quebec and Canadian missions to foreign countries on the subject of software engineering, software processes, and CASE technologies. He is a consultant to several organizations in the field of software engineering and process technology.