The Uses of Process Modeling: A Framework for Understanding Modeling Formalisms

William H. Kleppinger
Doris Y. Tamanaha
Leon J. Osterweil

UCI-ICS Technical Report 92-116
Arcadia Technical Report UCI-92-01

April 14, 1992

This research was supported by Hughes Aircraft and their Command and Control Systems Division under the 1LA02C IR&D Project. This research was also supported by the Defense Advanced Research Projects Agency, through DARPA Order #6100, Program Code 7E20, which was funded through National Science Foundation grant #CCR-8705162. This work was done while the first author was supported by a Hughes Aircraft Staff Doctoral Fellowship.

The Uses of Process Modeling: A Framework for Understanding Modeling Formalisms

William H. Kleppinger
Hughes Aircraft Co.
and
University of Calif., Irvine

Doris Y. Tamanaha
Hughes Aircraft Co.

Leon J. Osterweil
University of Calif., Irvine

UCI-ICS Technical Report 92-116
Arcadia Technical Report UCI-92-01

April 14, 1992

Abstract

There is wide-spread recognition of the urgent need to improve software processes in order to improve the performance of software organizations. Process models are essential in achieving understanding and visibility of processes and are important for other uses including the analysis of processes for improvement. It has been increasingly difficult to compare and evaluate the variety of process modeling formalisms that have appeared in recent years without a clear understanding of precisely for what they will be used. The contribution of this paper is to provide an understanding and a fairly comprehensive catalog of the applications of process modeling for which formalisms may be used. The primary mechanism for doing this is a guided tour of the literature on process modeling supplemented by recent industrial experience. In the paper, basic definitions concerning processes, process descriptions and process modeling are reviewed and then uses of process modeling are surveyed under the following headings: communication among process participants, construction of new processes, control of processes, process analysis, and process support by automation. Comments are offered on paradigms for process modeling formalisms and directions for future work to permit evolution of a discipline of process engineering are given.

1 Introduction

1.1 Importance of Process

In [Ost91] one of the authors (LO) summed up the significance of processes in our lives:

> Ineffectiveness in the design, analysis, and execution of processes lies at the heart of many of the most important problems faced by many of our most important

> institutions. Processes are used to guide manufacturing, construction, management, government, and military activities. Existing processes are often inefficient, of poor quality, or incorrect. Deficient processes lead to poor products. When processes are inefficient, products are late to market, roads and buildings are completed late, military forces are deployed tardily, and project deadlines are missed. Poor quality processes generally lead to poor quality products, uncontrollable activities, and unpredictable outcomes. Incorrect processes can lead to disasters such as misplaced trust in faulty products, malcoordinated military campaigns, and government institutions that simply do not work.

A disciplined, uniform approach to the development of high quality processes, and efficient execution of such processes is needed. The most significant benefits of this are: increased predictability of an organization and its products, improved organizational efficiency, increased productivity for individuals and groups, and enhanced contributions by computers to the work of organizations.

1.2 Need for Software Process Improvement

Since, as Mark Dowson [Dow91c] points out, the software industry manufactures large, complex artifacts — software systems, it is important that its products be of high quality and reasonable cost, and be delivered on time and within budget. To achieve this, it is correspondingly important that we improve the manufacturing process — the software process. In the same connection, Watts Humphrey points out:

> The software process provides the most practical framework for making both long and short-term improvements in the performance of software organizations. Because of the growing importance of software to humankind, software process improvement must be a high priority. The issues needing research and development attention range from the management of large software operations to fostering and supporting the creative and effective performance of individual software professionals. [Hum91]

It has been noted [Dow91c] that "historically, the software engineering community has focused mainly on the products of software processes. Recent advances in the understanding of these processes, focusing on the activities involved in creating software products, present an opportunity to solve many of the problems underlying software creation and evolution." Understanding and improving the software process is directly related to two major problems with which software engineering is concerned: (1) High cost due to long execution time of the development process; and (2) low quality of products in the development process. Lehman [Leh91] defines software quality as the measure of "user satisfaction with a software product over its lifetime." Cycle time reduction for software products is tied to the process so there is a need to understand development processes, and meeting quality objectives in the delivered product requires a suitable quality-oriented process. The current quality of the development process is acknowledged to be of too low a level to be satisfactory.

The recent concern with Software Engineering Institute (SEI)-defined process maturity levels and the capability maturity model [Hum91] for software acknowledges that organizations must learn from their own and others successes and the realization that improving the process implies continual and continuous need to change for the better. We must therefore assure and manage the activities of evolution and change in a constant regimen of self-improvement.

Humphrey [Hum91] concludes:

> There is an urgent need to improve software processes at both the organizational and individual levels. In the future, we need major improvements in the capability of the software industry. These will generally come, not from a few block-buster inventions but from the countless small process improvements that compound to produce well managed organizations that effectively use their software professionals.

Lehman [Leh91] states that software development processes must be deliberately designed and that for process design one must be able to describe and reason about processes. This requires descriptive models and models that provide a context reflecting the objectives, development and usage environments and technologies to be used including methods, procedures, tools and measures. In particular "...process models play a vital role in achieving *understanding* and process *visibility*. Both are necessary for development of quality, cost effective, processes and their evolution as software technology advances."

1.3 Role of Process Formalisms

Process models play an important role in process guidance, "elucidating process and project relationships, action sequences, interaction and flow of information [Leh91]." In addition, the mechanization of some process steps by means of CASE tools can only be effective if applied on a process and project basis, if benefits are assessed over product lifetime, and if methods and tools are integrated leading to "a coherent process that can be readily introduced, adopted and pursued [Leh91]." Process models are at the core of having a disciplined process, mechanization and process guidance to ensure correct use of methods and following the defined process.

Process formalisms have evolved rapidly in recent years, attested to by a series of seven annual International Software Process Workshops [Pot84, WD86, Dow86, ISP88, ISP89, ISP90, ISP91], a European Software Process Modeling Workshop [FCA91], and an International Conference on the Software Process [Dow91a]. The goal of the formalism developers is for the formalisms to be used to achieve the objectives outlined above. It has been difficult to evaluate and compare these formalisms without a clear understanding of precisely for what they will be used. The contribution of this paper is to provide an understanding and a fairly comprehensive catalog of the applications of process modeling for which formalisms may be used. The primary mechanism for this will be a guided tour of the literature supplemented by recent industrial experience. In what follows, basic definitions concerning processes, process descriptions and process modeling will be reviewed and then uses of process modeling

will be surveyed under the following headings: communication among process participants, construction of new processes, control of processes, process analysis, and process support by automation. Finally, comments will be made on process modeling formalisms and directions for future work to permit evolution of a discipline of process engineering will be given.

2 Processes

2.1 The Variety of Processes

An elementary definition of process (given in [Ost87]) is: a systematic approach to the creation of a product or the accomplishment of some task (this characterization includes the notion of process commonly used in operating systems — a computational task executing on one computing device.) The concept of process occurs in a number of subject matter domains: cooking, chemical engineering, office procedures, value engineering, VLSI design, and software among them. The varieties of process encountered in these domains have a common thread: they represent an interplay of products and processes and transformations on the products against the flow of time. The transformations are often described by a set of actions performed by various agents. The flow of time is often organized into events that are characterized by changes in the state of products or of the world external to the process under discussion. We shall further limit the processes to which the comments in this paper are meant to apply to include only those that involve human and (possibly) computer participation. (For example, we exclude processes featuring chemical agents.)

It will pay us to try to be somewhat more precise in defining process, however. Generalizing the definition given in [SO91] for a "design methodology", we may say that the commonly used term "methodology" refers to a collection of methods (which specify how to arrive at decisions needed in the process), chosen to complement one another, along with rules for applying them. Concepts, artifacts, measures, guidelines, rules of thumb, notations, and procedures are parts of a method (the method components). A process, then, can be described as a type of activity that adapts a methodology in response to local factors, and uses it to devise artifacts that are to satisfy specific requirements. Thus, a process can perhaps be viewed as an execution of an instantiation of a methodology.

There is a hierarchy of concepts, varying in their degree of generality, that are related to the notion of process. In efforts to define a corporate software development process at Texas Instruments, five levels of process concepts were distinguished (in order of increasing specificity): meta-process, process framework, generic process, process model, and process (an execution of the process model) [Fra91]. We will be concerned here primarily with process models.

2.2 Basic Process Description Primitives

The following distinction between processes and process descriptions was made in [Ost87]. A key difference between a process and a process description is that while a process is a vehicle for doing a job, a processs description is a specification of how the job is to be done.

Cookbook recipes are process descriptions while the carrying out of the recipes are processes. Office procedure manuals are process descriptions, while getting a specific office task done is a process. From a computer science perspective, the difference can be seen to be the difference between a type or class and an instance of that type or class. The process description defines a class or set of objects related to each other by virtue of the fact that they are all activities which follow the dictated behavior.

Processes consist of artifacts, concepts, representations, as well as actions. Tully [Tul88b] states that the things that process models must be able to represent include at least some of the following: actions, activities, agendas, agents, configurations, deliverables, events, messages, methods, obligations, permissions, pre- and post-conditions, roles, rules, tools, triggers, types, versions, views. Depending on the purpose to which a process description is intended to be put, all these things may need to be captured in the description.

Definitions are given here for the more specialized of these:

- process step: an action performed during a software process [DNR90],
- resource: an available asset supportive of carrying out tasks (resources must be allocated to tasks before they can be used to support task execution) [DNR90],
- agent: an entity which performs process steps (an agent may be either human or software) [DNR90],
- role: defines the behavior of a human resource (i.e., of an individual with well identified skills who devotes a given amount of time to perform a task inside the software process) [Amb91],
- event: means of synchronizing process step performance [DNR90],
- constraint: a restriction upon process steps and the data they operate on (agents, events, and their interrelationships) [DNR90].

Other things that may need to be captured are Song and Osterweil's method components [SO91]: concept, artifact, measure, guideline, rule-of-thumb, notation, and procedure.

Feiler [Fei90] discusses how event/action modeling primitives of different formalisms and environments vary in the set of events and the set of actions they accommodate. Events may consist of completion of various operations on objects or of events on tasks. Actions may register an event with other objects, instantiate new tasks, send E-mail or execute arbitrary command scripts. A model can offer events on objects (a product-oriented view), on tasks (a task-oriented view), or on both (mapping events on objects into actions on tasks). The latter is common.

One intrinsic difficulty in process modeling comes from the static nature of process descriptions contrasted with the dynamic nature of executing processes. The process description may specify a very wide and diverse collection of dynamic processes, some of which may not perform "correctly". Processes (as with Dijkstra's observation concerning program executions) are hard to comprehend and reason about while process descriptions (as with programs) may

be far easier to comprehend due to their static nature. The closer our process descriptions can resemble the processes that might be instantiated from them, the more useful our reasoning about the descriptions will be in understanding the processes [Ost87].

Humphrey [HK89] notes some resulting difficulties with traditional task-oriented views in process models. Task-oriented views (such as the waterfall model) are "appropriate and relatively easy to understand when the tasks are simply connected, [they] become progressively less helpful when the number of possible task sequences increases." In attempting to make these models more comprehendable, the formalisms may overconstrain task scheduling by limiting the number of action sequences that are permissable.

2.3 Process Modeling

Process modeling as a research area "studies software process, (i.e., those activities, such as methodology guided design activities, involved in software development and maintenance)." [SO91] It "supports the rigorous and explicit descriptions of static software processes and structures of their components" [SO91]. Process modeling as an engineering technique involves the creation and use of an appropriate abstraction to describe the nature or behavior of a real-life process (paraphrasing the definition of a model given by [Kri69]). Like other engineering modeling techniques, it is used for thinking, communication, prediction, control, and training related to the real-world thing it abstracts.

It has been pointed out that to be complete, a process model needs to contain functional, behavioral, structural, and conceptual data modeling views and that a complete model of a complex process is inherently complex [HK89]. Dealing with this complexity through the use of multiple views is also characteristic of systems engineering, and, as we shall see, many of the issues associated with process modeling are also issues associated with building system models during system development. A process viewed as a system, is decomposable into levels of subsystems and eventually simple components, connected by a variety of interfaces. (Tully notes, in fact, that "systems analysis, systems design, and systems engineering are in essence interface analysis, interface design and interface engineering [Tul90b]".) The subsystems and components are subprocesses and tasks, and the interfaces are intermediate products.

The level of detail included in a model is another such issue. It has been stated that process models must be refinable to whatever level of detail is needed and that previous models have been guilty of not providing sufficient detail to support process optimization. [HK89].

Understandability versus precision is a central tradeoff in deciding how to model a process. For some uses of process modeling, understandability is the most important attribute of a model, for other uses, precision is paramount. Informal semantics and flexible constructs in a modeling formalism allow an expressive representation to be designed that is easy for human readers to understand and requires little familiarity with the formalism on the part of the reader. Well-defined or formal semantics supporting an extensive set of constructs may permit process model execution or enable fine distinctions to be made regarding a rich array of process aspects. For the most part, however, formalisms well-suited for producing understandable models may not be well-suited for producing precise ones and vice-versa.

In the course of developing, adapting, using, and maintaining process models, various operations (meta-operations, to be precise) need to be performed on the process models themselves. The following set of primitive operations on process models has been proposed [Rue90]: specialization, decomposition, instantiation, deduction, generalization, and aggregation.

3 Uses of Process Modeling

The authors firmly agree with Watts Humphrey when he says, "The question, therefore, is not 'What is the right way to model the process?' but 'What is the most appropriate way to model this process for this purpose? [HK89]' " Furthermore, the selection or design of appropriate process modeling formalisms that expose the appropriate view of a process and that can represent the process with a useful degree of precision depends on a firm understanding of the various purposes to which process modeling can be put. (In fact, for a given purpose, a mix of a number of formalisms may be necessary.) The discussion in this section is intended to contribute to such an understanding.

The various uses of process modeling have been discussed by a limited number of authors. Tully [Tul88a] has stated the need to "understand and compare software processes, evaluate and reason about them, and design and replicate (reuse) better ones." Lehman is another who has recognized that "desirable characteristics of models will vary with the purpose for which they are to be used, the benefit one hopes to derive from their development or use and the value attached to benefit to be gained from such activity". He has provided an extensive list of roles for process models [Leh89]. To a large extent, however, research in process modeling has been centered on the development of modeling formalisms and experience in creating models for particular purposes, or, in some cases, on the development of modeling formalisms independent of any particular purpose for which they might be used. There is a need to characterize the various applications to which process modeling formalisms have been put so that the formalisms can be evaluated and compared. It is necessary to rationalize our knowledge about the nature of the various formalisms in order to begin to create a scientific discipline of process engineering.

In what follows, the uses of process modeling are discussed under the following headings: communication among process participants, construction of new processes, control of processes, process analysis, and process support by automation. The interrelationships of these five categories of process model use are shown schematically in Figure 1. As the figure suggests, process analysis has a relationship with all the four major application areas since analysis techniques provide the manipulations of the process model that allow the model to solve problems in each area. For each use, the central problems that process modeling is being called on to solve are explained and example applications from the literature are cited.

3.1 Communication Among Process Participants

Here, we consider the use of process models to communicate information about a software process and its instantiation. (We discuss the need to construct process models that represent

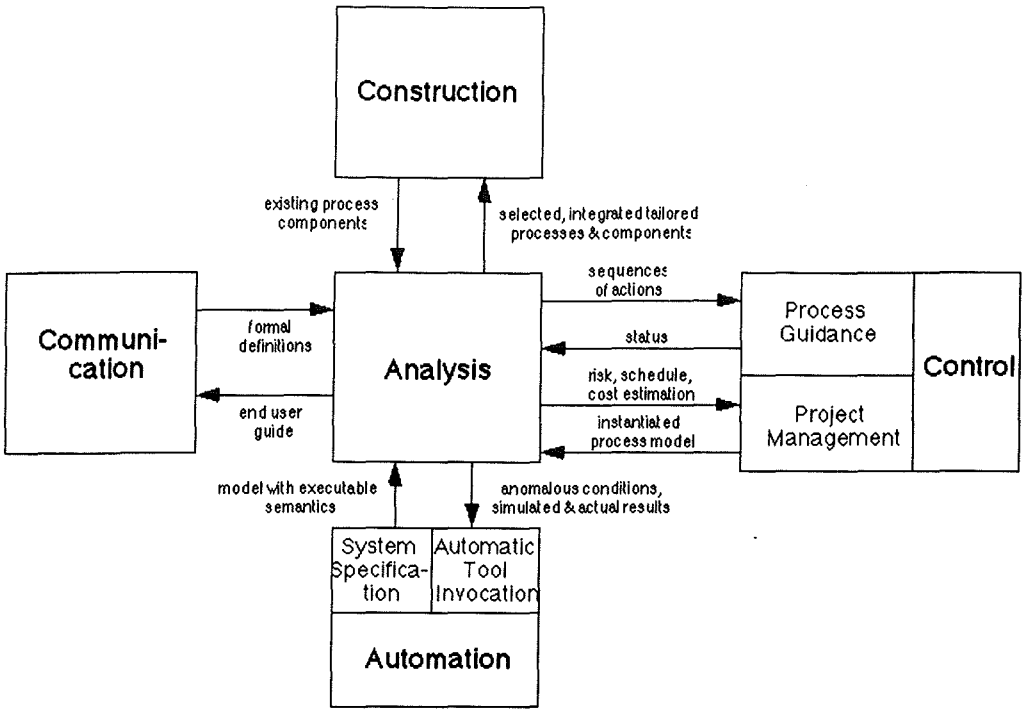

Figure 1: Uses of Process Modeling Formalisms

general technical communication among participants in the section on construction of new processes.) Situations where two or more groups on a project need to communicate but do not are commonplace. This can be caused by failing to realize the need for communication or by neither group perceiving that it is their responsibility to initiate it [KCI87]. Not only are there obvious synchronization failures in software processes (e.g., failures to communicate, missed deadlines, duplicated effort, omitted tasks), but there are misunderstandings and disagreements about contents of products (e.g., software requirements specifications) due to failure to understand how they will be used in the project by various participants. For example, when major phases of a large project are performed by different groups of people, there are problems in the transition between phases when groups hand off intermediate products to the succeeding group. Furthermore, major problems can occur when iteration back through these phases is necessary and the previous groups have been disassembled or reassigned [KCI87]. Process models are used to enable effective communication among process users, process developers, or managers in the various engineering groups involved in a project.

Another aspect the use of process models for communication is the process experience transfer problem [Rom89]. It is difficult to transfer the organization's informal process knowledge. Industrial experience shows process models can help to train new personnel by making that knowledge more formal. An example of this use of process models is found in the efforts of Texas Instruments Incorporated to define a preferred software engineering process for

use throughout the company [Fra91]. In the hierarchy of process representations mentioned in section 2.1, Texas Instruments set out to create a *process framework* that would act as a "blueprint" to be instantiated for particular applications. In Frailey's words, "the process framework provides leverage for learning from the experience of others, and a common ground for organization-wide goals [Fra91]." The primary users would be software project planners and managers, and the software improvement teams that support their efforts at process improvement. The framework could also be used to train software engineering and support personnel, however.

In selecting a modeling formalism for their framework, Texas Instruments needed a method of presentation that could readily be understood by the majority of their end users. Due to their wide acceptance and intuitive appeal, data flow diagrams were selected. In retrospect, however, they probably would have made a distinction between the model used to define the process and the model used to present it to end users. Frailey offers the following opinion:

> Our compromise proved workable but not the most desirable for either definition or representation. It was not formal enough to form a solid, provably consistent definition; and it was not intuitive enough to serve as the end-user's guide about what to do [Fra91].

The distinction is illustrated well by Watts Humphrey's observation (from a private communication with Frailey) that the definition model is analogous to source code whereas the end user model is analogous to object code for a human processor. This suggests that communication models might be defined formally, then tailored for a particular project and then presented in a user-accessible form.

Efforts are also being made [MYKS90] to use a description of a software development process as an instructional tool. The process description is used to provide a learner at a workstation with a list of inputs, outputs, and process constraints for a human-executed task. The learner is then presented with an actual set of inputs and must use the described process to produce an actual set of outputs. Efforts must be made to keep the development work performed within specified constraints for schedule, budget, quality, etc. The goal is to use the prompting of the instruction tool to both enable the learner to understand the development process and master the skills involved in doing development that way.

3.2 Construction of New Processes

Process development is expensive. Yet, every organization, and even every project within an organization, generally develops its own and develops it from nothing more concrete than the intuition of the project's managers. Watts Humphrey acknowledges the situation and concludes: "For software development to progress from a tediously unproductive craft, we must learn to build on the results of others. This starts with a defined working process [Hum90]." Osterweil suggests that "...the most important benefit of process [modeling] is that it offers the hope that software processes themselves can be reused" [Ost87].

There are at least three important subproblems in the construction of new processes from existing ones. First is the selection of appropriate existing processes or of components from

applicable processes to be recombined in possibly new ways to create a new process. Second is what Colin Tully refers to as "method integration" which

> ...aims, by selecting and adapting methods appropriately, to avoid counterproductive conflicts which might otherwise arise from the use of different notations to represent similar information, from the use of different techniques to perform similar tasks, or from gaps and overlaps in their coverage of the process" [Tul90a].

Third is what Dieter Rombach has referred to as a process tailoring problem: namely that "tailoring of processes to changing project goals and project environment characteristics is based on subjective rather than objective knowledge regarding the project differences and the effectiveness of candidate methods and tools" [Rom89].

All of the above subproblems can benefit from good comparisons of existing generic processes for software development. For example, in the case of design methodologies, [SO91] notes that a systematic and objective comparison would aid in codifying, enhancing and integrating the design methodologies that have been developed over the last 20 years. It is noted that existing assessments and comparisons of methodologies are overly affected by differences in application domains and project personnel as well as varying levels of understanding of method components by various authors. Such design methodologies can be viewed as a generic and static process definition that can be instantiated to create a process model for a particular project. [SO91] proposes an approach to comparing methodologies based on process modeling. The methodologies are decomposed into method components and artifacts using a set of process modeling formalisms. The components are classified by comparison to a methodology-independent model of the design process. Objective comparisons of analogous components and assessment of overall process coverage by the various methodologies are now possible. This approach has been used to compare a number of contemporary design methodologies.

It is clear that the existence of explicit models of existing processes are a step in the solution of all three of the above subproblems. We should note some of the characteristics that the formalisms for expressing these explicit models should possess.

Process modeling formalisms must allow the construction of processes that possess certain key attributes. One of these is the ability to represent technical communication among participants. That communications among process participants is an important consideration can be seen from studies showing job communications accounts for as much as 32% of a programmer's time [KCI87]. Curtis, et. al. note in [CKSI87] that organizational structures used in software development generally assume that the artifacts produced are sufficient to convey all the information the software developers will need to complete their assignments. Curtis believes, and most software developers will confirm, that this is not the case. In addition to supporting less formal communications, a process should not require recipients to wait for complete information; a process should allow information to reach its audience in time to be used for making subsequent design decisions. Often times these decisions are being made in concurrent tasks in the process. It has been noted [KM91] that in many situations where process steps can be executed concurrently, they are executed concurrently by exchanging information about their progression through a number of points of interaction that are difficult

to identify in advance. The model of communication will need to address this kind of concurrency. In the opinion of many [CKSI87, HK89], a process model needs to address the need for constant technical communication among participants. Indeed, according to Curtis, since common organizational structures are frequently not designed to enhance communications among engineering groups, the process model should be able to "imply recommendations for the organizational design best suited for supporting large systems projects" [CKSI87].

One attempt to address the representation of information communication in a project is the work of Peuschel, Schäfer and Wolf [PSW90, PSW91], implemented in the MERLIN environment. Here the process model describes the "cooperation model" between a number of small teams working together on a project. Information is assigned to one of three levels of working context based on whether it is needed for coordination of the entire project, the coordination of just a team, or only necessary for the job of a single individual. Objects may be manipulated by either fine-grained representations or coarse-grained representations (or both); may be manipulated using different transaction models; or may be distributed to sites depending on the working context within which they are being accessed.

As a second key attribute, process modeling formalisms must allow the construction of processes that possess domain understanding. As Zave [Zav89] points out "based on our current level of understanding of the software process, there are major unexplained differences between application domains." She states that in a well understood application domain, there exists a great deal of public knowledge about how the required class of functions can be mapped successfully onto software structures. This knowledge can be most easily captured, used, and communicated in the form of "domain understanding", a specific framework for specification and implementation of a class of functions. Domain understanding is conventionally captured in the form of subsystems, layers, languages, techniques, constraints, or by humans in the form of "exceptional designers" [CKI88]. Zave states that domain understanding should be a major factor in any model of the software process. Among other reasons, this is because "domain understanding cuts across (and undermines) the traditional divisions between requirements, design, and implementation" [Zav89].

Balzer [Bal90] points out another key attribute when he states that process modeling formalisms should permit the construction of processes that tolerate and deal with inconsistency and incompleteness. Rather than requiring inconsistency and incompleteness to be treated informally (outside the system) or as absolute constraints, the processes constructed should spot such violations, treat them as problems, organize resources to resolve them, recognize when this has occurred, and limit access to the inconsistent data by agents not involved in the resolution. One approach to modeling processes that handle unexpected events that interrupt software development activities has been proposed by Mi and Scacchi [MS91]. It incorporates a knowledge-based model of resolution or recovery activities into the modeling formalism. Such activities (referred to as *articulation work*) and the breakdowns to which they respond are an inherent part of the software process. The approach uses *problem-solving* and *selection* heuristics to direct how articulation work is carried out, how a solution is formulated and selected, and how a solution is realized (perhaps incompletely). The indicated solution is represented by using the process modeling environment to make changes such as replacing

the resource allocation, adding an activity, or replacing an existing activity.

Humphrey suggests: "As process model development and customization costs decrease and as the benefits of an orderly process framework become more apparent, it will be more reasonable for each project to define and/or adapt its process before launching into a new product development [Hum90]."

3.3 Control of Processes

Process control may be exerted at both the individual level (process guidance) and the organizational level (project management). The two applications are intimately related, however, and are related through their use of process modeling. As Hubert points out, "...the project management activity can be viewed as the instantiation mechanism for process guidance [HFB90]." Project management takes the process model, with its implied methodological constraints, and uses managerial constraints such as schedules and available resources that are part of the project plan to instantiate the process for execution [HFB90]. This project management activity is what Watts Humphrey refers to as the planning process that "typically breaks large problems into manageable tasks and assigns individuals to perform them [Hum90]." He goes on to point out, however, that plans not only mobilize the organization's resources to the task in hand but also provide individual guidance. From a process perspective, process guidance/monitoring takes the instantiated process and the policies and constraints it implies and controls the process at the level of the individual participant through execution of a process model according to the policies and constraints.

The notion of execution of process descriptions (sometimes referred to by the phrase process "enactment") is central to control of processes using process models. Implicit in execution is the idea that human beings involved in the software process receive computer guidance and assistance in what is generally an extremely complex activity [Tul88b]. To do this, process models are used, as [Tul88b] points out, "on-line" while processes are being carried out, as a means of directing, controlling, monitoring, and instrumenting them. A more careful definition of the notion has been supplied by Christer Fernström [FO91]: " 'process enactment' refers to the simultaneous and synchronized execution of a human-oriented development process and an executable model of this process in order to enhance the computer-based support given to the human-oriented process."

3.3.1 Project Management

The conventional wisdom is that project managers undertake three main types of activities: planning, controlling, and monitoring projects [Dow91b]. In a survey of Department of Defense program management offices, the Defense Systems Management College's Decision Support Systems directorate further refined the program/project manager's responsibilities and identified the need for four capabilities: risk management, cost estimating, scheduling, and monitoring [Def91]. Project managers, then, strive to create plans that control risk, minimize and reflect accurate cost estimates, and schedule resources and activities accordingly. One tool for this is the construction of an explicit process model.

Dowson [Dow91b] points out that project managers do planning "in the light of a model of the process that should be followed by the project, and attempt, perhaps incrementally and reactively, to produce plans that conform to that model." The model may be implicit, based on the manager's personal experience, or explicit, represented in the language or formalism of a process model. Dowson further points out that "if both the process model and the plans produced for a project are explicit and unambiguous, it is possible, at least in principle, to verify that the plans conform to the process model".

Huff's GRAPPLE system [Huf89], in an attempt to support the dynamic, context-dependent nature of planning through the use of powerful instantiation facilities; adopts a general approach to reasoning about actions, based on the artificial intelligence theory of actions. In the GRAPPLE view of planning, knowledge of a domain is expressed in terms of predefined *operators* (having defined preconditions and resulting effects), a state schema (consisting of predicates describing the world for the domain in question), and *goals* (logical expressions composed of state predicates). A *plan*, then, is defined as "a hierarchical, partial order of operators (with bound parameters) that achieves a goal given an initial state of the world [Huf89]." Through decomposing a plan's complicated goals into simpler subgoals by replacing operators with lower level operators one at a time, the plan is incrementally developed and "executed". Using such plan-based process definitions, a form of analysis is performed that permits reasoning about sequences of actions without actually executing the actions. In fact, one of the main reasons for constructing an explicit process model is to allow reasoning about and analysis (in the form of risk management, cost estimation, and scheduling) of processes so that they may be improved, irrespective of the project-specific decisions of project managers and other project staff. Monitoring of project performance can provide feedback which will allow models to be improved and evolved.

Work has been done to develop analysis techniques based on explicit process models to support the project manager's risk management, cost estimation, and scheduling. Statechart-based simulations of processes with manpower levels and durations assigned to the process steps (states) is an example of automated analysis of a computer-readable model to determine manning levels (and hence cost) and schedules. (Statechart simulations can be used to automatically produce Gantt charts.) Resource-constrained statechart simulations can be used to show schedule implications of manning constraints. Steepness of staffing curves can be used to associate risk levels with plans. The cost drivers for various software cost estimation models are essentially means of quantifying risk and applying it to cost estimates. Manual analysis to determine bottlenecks, redundant effort, level of automation, and efficiency of interfaces are often applied to less formal but still explicit models. These techniques for supporting project manager planning and others are discussed below under process analysis.

For use in monitoring a project, Fernström [FO91] notes that an executing process model is an instance of a process model whose current status is maintained by the "enactment system". This allows the project manager to get a complete picture of the state of the project in real time, enabling decisions to be made based on actual facts rather than optimistic or pessimistic estimates. He further states the executing process model "can even, to some extent, do the monitoring automatically and work out the actions to be taken at certain decision points in

the workplan". Use of statechart-based process models to do monitoring and replanning when actual progress differs from the plan [Kel91b] is an example of this.

Defining an organization's software process explicitly not only provides to management visibility of and standards for individual tasks as they are performed, but may also enable the process to be supported so that routine tasks (e.g., change management, design control, interface coordination) are handled routinely [Hum90]. This is described later under process automation.

3.3.2 Process Guidance

In discussing process control problems, Watts Humphrey observes:

> Policies which are overly specific constrain action, limit freedom, and demotivate the people. If guidance is too vague and imprecise, however, the professionals will likely be confused and unwilling to act. When people are uncertain about management's desires, they often fail to exercise initiative or blindly follow some rule or procedure. In software, clear policy is thus an essential prerequisite to both high productivity and quality.[Hum90]

Along a similar vein, Dieter Rombach states that the lack of explicit and formal process specifications creates a process control problem in that "controlling the adherence of a process execution to an inadequate specification is necessarily based on subjective judgement rather than objective criteria [Rom89]". The problem thus described is an area that can be addressed by means of process modeling.

It has been asserted that within a software development environment, a computer-readable process model can be used to help assure that a project is conducted within the framework of a disciplined process making errors less likely and making them easier to detect and correct when they do occur [DNR90]. In addition, after instantiation of a process model according to the constraints of a project plan using the project management capabilities discussed above, the instantiated model can update project status information as it is executed [HFB90].

Experimental environments constructed as part of the ESF project, the OPIUM [HFB90] and ARCHIPEL [FO91] environments, have explored these issues with some success. In ARCHIPEL accessing tools and data through an Agenda rather than the conventional windowing environment gives the user full support from the executing process model. Access with process guidance (for planned activities) or without is realized through user activation of a task stored in the Agenda. An Agenda's tasks may be visualized as lists, icons, or graph vertices. Tasks may also be created, modified, deleted, or delegated. Work contexts defined by the model accompany tasks. It can be seen, then, that a variety of restrictions may be imposed on the user to control and guide the process. One of the most valuable forms of guidance, however, that a process model may provide is a simple indication of options or alternative means of accomplishing a task are available to the user at a given point in time.

Humphrey's call for process models that are clear and precise, without being overly precise, is acknowledged in recent work that concerns different styles of computer-readable process

descriptions. Dennis Heimbigner states that "the degree to which a process-centered environment is accepted will depend both on its ability to enforce a specified process and on its ability to support a non-restrictive style of interaction with programmers using that environment [Hei90]." To accommodate these seemingly disparate goals, he discusses the prescriptive style of process model versus the proscriptive style of process model. The term "prescriptive" indicates that the environment (driven by the process model) closely controls the means by which a task is to be completed and the order in which tasks are to be performed. Such a model (or "process program") tends to be procedural and serial requiring one task to be completed before beginning another. In contrast, the term "proscriptive" indicates that the environment operates by prohibiting inappropriate manual actions and ensuring conformance with specified consistency constraints. It does not constrain the means or order in which tasks are performed. Choices of prescriptive versus proscriptive process models are design choices responding to the nature of the process and the formalism should support both [Sut91a].

Recognizing this same tension in the style of guidance provided, Dowson, Nejmeh and Riddle [DNR91] talk about the need for *constrained cooperation software processes*, defined as "software processes which flexibly maintain a dynamic balance between self-guided cooperative problem solving and the discipline imposed by constraints upon this cooperation". They further note that

> The successful application of constrained cooperation software processes additionally depends on the degree to which constraint observance can be monitored and software processes can be dynamically modified in response to events occurring during the project.

3.4 Process Analysis

In general, process analysis is making inferences about processes. This may be for global optimization or other improvement, for verification, or for evaluation. Dowson [Dow91b] states that improvement is, in fact, one of the main reasons for constructing an explicit process model. Elsewhere, Humphrey and Kellner have stated that process models must support comprehensive analysis of the process through the model, and allow predictions to be made regarding the consequences of potential changes and improvements [HK89]. It has also been proposed that analysis be directed against detailed, executable process models called "process programs" to arrive at measures of complexity of the corresponding software processes [Ost87].

The lack of explicit and formal process descriptions creates the following process improvement problem:

> The desired improvement of software processes requires an understanding of their current status, the identification of weaknesses regarding their current status, a systematic eradication of those weaknesses, and a validation that the new processes indeed represent an improvement. Without adequate process specifications this process will be random rather than systematic [Rom89].

Curtis warns, however, that models that focus on how a software product evolves through a series of artifacts poorly describe such crucial processes as learning, technical communication, requirements negotiation, and customer interaction. Models that merely prescribe a series of development tasks provide no help in accomplishing a number of the objectives of process analysis identified in the next subsection. Such evolutionary process models need to be integrated with models that emphasize factors affecting psychological, social, and organizational processes to create a comprehensive model of the software development process. A model that overlays these various processes begins to show causes for bottlenecks and inefficiencies in development [CKI88]. Unfortunately, existing formalisms and automated analysis techniques are inadequate for representing and reasoning about psychological, social, and organizational processes. Consequently a good portion of the most significant analysis techniques may need to be intuitive, informal reasoning performed by a human analyst inspecting information collected in notes supporting a process model.

The process analysis that has been described to date has been associated with one or more of the following goals:

- Analysis to support model building. This includes finding errors in the process model such as incompleteness or inconsistency of the description due to blunders or lack of understanding of the process [KH88].
- Analysis to identify real-world process anomalies (e.g., non-determinism — situations where there is more than one way for the described system to respond) [KH88].
- Analysis to show the behavior and reaction of the model to changing events (e.g., the changes related to one person: sick for two weeks, replaced, removed) [KH88, Kel91b, Red90].
- Analysis of process attributes of interest (e.g., time-to-complete, manpower requirements, quality measures) [KH88].
- Analysis to determine if the model complies with constraints such as required documents, formats, review points [KH88].
- Analysis to determine relationship of an explicitly represented manual activity with other activities (manual or automated) for the purpose of facilitating and directing manual activities [Sut91a].
- Analysis to determine properties of processes (e.g., in process X, a programmer cannot complete a build without archiving the source [Huf89]; security properties [Red90]; impact on final work products of human error rates [Red90]).

Tools have existed for some time to analyze instantiated processes for certain specific characteristics. Software cost models exist (e.g., SEER [Jen81] and COCOMO [Boe81]) for predicting the cost to complete of instantiated software development processes utilizing a number of predictors. Tools have also been written to predict schedule risk of an instantiated process based on the steepness of staffing curves required to meet schedule constraints [Orm83].

Dowson [Dow91b] makes the point, however, that it is important to separate consideration of inherent properties of a process or class of processes from consideration of project-specific decisions and actions of project managers and staff. Analysis of a *model* rather than an initiated process can lead to "improvements which will be reflected in all projects that are performed in conformance with it [Dow91b]." Monitoring *project* performance can, of course, provide feedback which will help models to be improved and evolved.

3.4.1 Objectives of Analysis

We can conclude from the general goals listed above that analysis is itself not done for its own sake and that models are used to support analysis. Each analysis effort should be designed to solve a particular problem, however, and it will be instructive to list some specific objectives at which analysis of process models has been directed.

- Eliminate delays in task initiation [KH88] due to time consumed in data transfer, queuing of tasks, dependence on multiple inputs [Tam91].
- Introduce parallelism into the process flow [KH88].
- Enhance coordination and communication in order to reduce surprises [KH88].
- Identification of bottlenecks [Tam91].
- Identification of contributors to excessive rework [Tam91].
- Identification of hindrances to communication between functional areas [Tam91].
- Identification of redundancies (both process and data) [Tam91].
- Determine resource requirements (e.g., CPU resources needed by programmers) as part of "what if" planning, also called loading analysis [i-L89].
- Determine if the process will reach a situation in which a certain time limit or resource limit is reached [i-L89].
- Detect deadlocks in an enacted process [MTH91].
- Detect excessive idle time in the midst of scheduled peak periods in an enacted process [MTH91].
- Discovering complementarity problems between process descriptions and the capabilities of agents [MTH91].

In addition, Curtis [CKI88] gives several questions whose answers constitute additional objectives for analyzing. These concern broader issues than are normally represented in process modeling formalisms:

- How much new information must be learned by a project staff?

- How discrepant requirements should be negotiated.
- How design teams resolve architectural conflicts.
- How these and similar factors contribute to a project's uncertainty and risk.

3.4.2 Techniques of Analysis

Techniques used for process analysis seem, so far at least, to be largely specific to the particular formalism used to model the process to be analyzed. For this reason, we will organize the discussion of techniques according to the type of formalism for which they are intended.

One traditional technique for process analysis is directed at processes modeled as precedence networks and known as CPM or PERT. Here events or milestones are connected in a graph by arcs that represent tasks of varying durations that must be completed to achieve the milestone. The objective is to find those tasks that are on the critical path to know where task duration reduction will result in a decrease in overall duration of the entire process. It has been pointed out that the same results can be derived by inspecting Gantt charts that have in turn been produced from a statechart-based "entity process model" [HK89].

When processes are modeled by the Statemate tool [i-L89] as they are in the entity process models mentioned above, their description includes an activity-chart (a data-flow diagram) and a statechart (a modified state-transition diagram with hierarchy, concurrency and broadcasting). Automated analysis may be done to verify that the activity-chart and statechart are consistent and syntactically correct. For example, an event that causes a transition in a statechart needs to be shown as an input on the activity-chart. Appropriate types of consistency and completeness checks may be done with a variety of process modeling formalisms. Modeling formalisms such as statecharts that have semantics based on executable models of computation (in this case finite state automata) can be executed and, as we discuss below, permit dynamic analysis through simulation.

Petri nets have been modified to form a vehicle for process modeling and analysis. One such type of high-level Petri net, called a FUNSOFT net [EG91], consists of channels (object stores), agencies (activities) and edges (the relations between object stores and activities). Activation predicates attached to agencies model the dependence of activity execution on explicit conditions concerning the values of the tokens that flow through the net. FUNSOFT nets have semantics based on an executable computation model known as predicate/transition nets. Since there is an executable semantics, the analysis supported by FUNSOFT nets includes validation of process models against actual processes by simulation [Gru91]. Standard analysis facilities for predicate/transition nets are used for verification of interesting process model properties such as: the existence of object types for which no objects can be produced; the existence of object types for which no objects are processed; whether arbitrarily many versions of an object can be produced; identification of activities that can be executed permanently; whether objects stored in a particular channel cannot be produced (deadlock); and other static properties related to traps and activity conflicts as well as the well-formedness of FUNSOFT nets [Gru90]. Using predicate/transition net representation of FUNSOFT nets, it is possible "to show that the number of objects in a certain software process model part

remains invariant [Gru90]." For proving dynamic properties of FUNSOFT nets, a related notion known as quantitative coverability trees is derived. These trees can be used for deciding about properties such as: software development activities which can periodically or which cannot be activated (deadness and liveness of agencies); number of objects being in a certain state (boundedness of channels); maximal number of persons that can concurrently work in a software process; and potential occurrence of any software process state if interest [Gru90].

A separate effort based on petri nets has shown that plans and schedules for project management may be derived from a petri net based process model through "net theoretic manipulation steps" [BGM91]. They note that PERT charts correspond to the notion of "process" in net theory and that Gantt charts correspond to the firing sequence of a timed petri net.

Language of Temporal Ordering Specification (LOTOS), originally developed for formal specification of communication systems, has been used to model processes [SKS91]. It is claimed that this approach for process modeling allows not only task behavior to be represented (as with Statemate) but also represents the behavior of resources. There are a number of simulators available for LOTOS as well as tools for analyzing syntactic correctness of LOTOS programs. The behavior of LOTOS programs can be validated against real-world processes by program simulation (concurrency is simulated). The simulation of just the task behavior part of the description can be used to determine the event sequence of tasks as if one person were performing them.

Huff [Huf89] has noted that plan-based process descriptions (such as those used by her GRAPPLE system that were described above under Project Management) "appear to be amenable to analysis both because the formalism is logic-based and because the effects clause allows simulation of actions without actual execution" [Huf89]. In the case of GRAPPLE, the analysis takes the form of repeated application of predefined operators according to their required preconditions and the state of the system as altered by the effects of previous operators. To overcome some of the difficulties in debugging process descriptions using such rule-like formalisms, work on the MARVEL environment kernel for enacting such a formalism has included work on a tool to uncover direct and transitive dependencies among rules by constructing a dataflow graph of currently loaded rules [Kai89]. This analysis is designed to cut down on unanticipated interactions between rules. The EPOS environment kernel models planning activities with production rules and features a planner that interleaves planning and execution [CLW90]. The EPOS planner is a production system that uses both forward and backward reasoning on a knowledge-base of tasks to transform incomplete plans into "flaw-free" plans in a technique referred to as hierarchical planning. The goal is to provide intelligent assistance guiding a process participant.

Functional descriptions and other less formal representations are commonly used to perform analysis of processes in the sense of the dictionary definition of analysis: the separating of any material or abstract entity into its constituent elements. HFSP, which describes software processes as mathematical functions that map their inputs into their outputs, has been used describe a real industrial process to determine its appropriateness and completeness [KM91]. This was done by decomposing functions hierarchically until the resulting functions could be

performed by humans through either tool invocation or mental activities.

Structured Process Flows (SPFs) are a leveled, end-to-end threaded view of the major processes in a system and provide the necessary first view of a system while enabling decomposition for details [Tam91]. SPFs are annotated with process inputs and outputs at each level of decomposition. In addition to providing a vehicle for process understanding in the same way as HFSP, SPFs may be annotated with the values of various process metrics to perform informal analysis of overall processes. Areas of process analysis that have been addressed with SPFs include: identification of bottlenecks, items contributing to the problem of excessive rework, hindrances in interfaces between functional areas, and redundant processes and data. An example of the kind of informal reasoning-by-inspection techniques supported by such less formal types of descriptions is the following procedure:

1. An aggregate estimate for the elapsed time taken by a process at one level is secured from a supervisor and used to annotate its symbol on that level's SPF.

2. The process is then described in detail at a lower level of SPF with a series of subprocesses.

3. Elapsed time estimates for these subprocesses are obtained from the appropriate performing workers and entered on the lower level SPF as an annotation.

4. The aggregated time estimates can then be compared to the estimate obtained at a higher level to see if there are any gross mismatches.

Such mismatches may be symptoms of overlooked activities (e.g., ad hoc communication or reviews) or of hidden lags (e.g., time consumed in data transfer, queueing of tasks, or dependencies on multiple inputs) that may be opportunities for process improvement.

3.4.3 Dynamic Analysis Through Execution or Simulation

One frequently used technique for performing process analysis is not specific to any specific formalism. That technique is execution or simulation of the model and only demands that the formalism have well-defined semantics based on some executable model of computation. It has been stated that such simulation capabilities are crucial to achieving the desired benefits of software process modeling. This is because, in addition to identifying flaws and problems in models and modeled processes, simulations provide both qualitative and quantitative forecasting capabilities. The former include the behavior of the process in response to various events and circumstances. The latter include prediction of numerical outcomes such as time-to-completion, manpower requirements, or quality measures. These capabilities provide a vehicle for answering "what if" questions about such activities as procedural changes and technology insertion [KH88, Kel90].

The Statemate tool mentioned earlier has been used to perform simulation supporting a number of types of analysis of process models [i-L89]. (Use of the same features to analyze *system* models is described in [Har92].) The essential ability to carry out a single step of a process' dynamic operations may be used to execute a model in a step-by-step interactive

fashion or iteratively, taking a sequence of external events and signals off of a batch file and creating a trace file of the model's execution for later evaluation off-line. Executing expected behaviors permits verification that the process will behave as expected. Such model executions can also (just as they can for systems) uncover unexpected patterns of process behavior [i-L89, Har92].

Programming simulation runs, by means of an execution control language, to include breakpoints and actions to be taken automatically when the breakpoint conditions are reached and satisfied permits further types of analysis to be undertaken using the model. In order to see the model running under circumstances that we don't care to specify in detail, events might be selected randomly. As an example, we may want to know how many times an event occurs during a process. The model may select typical scenarios by generating random numbers to select new events according to predefined probability distributions. Statistics are then gathered using appropriate breakpoints and arithmetic operations. Performance analysis may also be carried out using execution control programs. To find out if certain resource consumption limits (manpower loading limits, elapsed time, development workstations, computer time) are exceeded by a process during its execution, known resource amounts are associated with the activities that consume them. Typical scenarios can then be run and total resources consumed calculated for all activities either active at a moment in time or over history of the run. As well as determining if consumption limits are exceeded, breakpoints can be used to determine the set of circumstances causing the limits to be exceeded [i-L89].

If all possible scenarios could be run through, the presence of deadlocks or non-determinism in the model could be detected. Reachability tests could also be performed, determining whether the process could ever reach a situation in which some specified condition (either desirable or undesirable) becomes true. Such analysis presents problems since the number of possible states and the time required to reach them may increases exponentially as the number of components and component states does. Useful analysis can be done, however, within practical limits by restricting scenarios to feasible ones and simulation to selected portions of a model [Har92].

A variety of logical conditions (expressed perhaps as sentences in a temporal logic) can be represented as "watchdogs" and cast as reachability questions. Harel describes a watchdog as "a small special-purpose 'piece' of behavioral specification that is carefully set up to enter a special state if and when the offending situation occurs" (it has been shown that under certain conditions, any temporal logic formula can be systematically translated into a watchdog) [Har92]. Reachability tests are run to see if the state is ever entered or under what conditions it is entered. Harel notes that promising research is under way into automatic verification of very large finite-state systems against properties in temporal logic using other means than exhaustive execution. In addition to Statemate, Harel gives references in [Har92] to tools that have been created to permit simulation of models using other computational models (e.g., petri nets).

A different type of simulation technique is used by Abdel-Hamid and Madnick [AHM91] for process analysis. The technique uses a continuous simulation model that integrates management and production functions of the software development process by the application of

feedback control systems principles. The software development process is represented in terms of levels and rates. A "level" is an accumulation of some object involved in the process (e.g., resource, product, or rework item), and a "rate" is a flow increasing or decreasing a level. As an example, a manpower would be represented as a level of people that is increased by the rate of hiring and decreased by the rate of firing and quitting. Levels are given initial values and rates are usually defined as functions of auxiliary variables, constants, and levels. The system's equations are expressed in DYNAMO, a simulation language for non-linear feedback models.

The DYNAMO model permits continuous re-estimation of the effect of changes in the work force or requirements. In particular, the model has been used for:

- parametric analysis such as where to put more quality assurance or where the schedule can be compressed;
- continual re-estimation to permit earlier convergence on effort level changes that are required to meet schedule;
- controlled experiments of the effect of different management decision dynamics;
- training of managers; and
- analysis of lessons learned from historical projects.

3.4.4 Process Measures to Support Analysis

Part of process analysis is the derivation of quantitative measures of both process models and instantiated processes. Basili implies the need for analysis when he states, "We need models of the [software] development process, measures of its characteristics, and practical mechanisms for obtaining those measures. [BM91]" Kellner [Kel90] states that process analysis using models often suggests process metrics, measurements, and status indicators that would aid in process management. For example, we need mechanisms that will identify process measures that contribute to product reliability, mechanisms that will identify skills that a process will impart to its participants (for satisfying skill-up requirements), and mechanisms that will identify software builds that can be separately coded and tested in order to decrease project risk. Kellner asserts that these measures can then be concretely defined in terms of the model's components so as to be meaningful for the process at hand. Rombach made the point that only integration of the to-be-measured process and the measurement process "will provide the desired engineering control for project-specific process execution and organization-specific [process] improvement [Rom89]." Further, he stated that experience showed the need to formalize the descriptions of both the to-be-measured and measurement processes, including "all aspects of measurement, ranging from data collection and validation to evaluation and feedback." The resulting models should be consistent as should be the formalized measurement goals, and appropriate automated support should be provided for designing and specifying the processes [Rom89]. The *Amadeus* system [SPSB91] provides such a framework for integrating empirically based analysis techniques with mechanisms for

enabling empirically guided to-be-measured processes. It does this by providing mechanisms for instrumentation of a process model and invocation of activities such as statistics gathering based on time and progress in execution of the process.

The process measures will provide valuable feedback for use in process control. Kellner notes that the prescriptive model embodying the project plan, along with associated simulation runs of the model, can serve as a yardstick for evaluating actual behavior and results. Actual measurements can be compared to the planned outcomes and appropriate action taken based upon any significant discrepancies. Such actions may include replanning efforts during which quantitative simulations of the process at the macro level may be employed to evaluate and select appropriate adjustments such as re-allocation of resources to various activities [Kel90].

3.4.5 Analysis to Support Project Management

In addition to the use of process measurements to control processes that was described above, another aspect of process analysis that supports project management is the analysis needed to develop an executable process from a process model. This may be characterized as applying resource constraints to an unconstrained process model to produce a final constrained process model [HK89]. An example of this type of analysis is the TSURU scheduling tool that assigns individual people to the tasks of a modeled software development process based on schedule constraints and the resulting need for parallel effort and on the organization's skill-up requirements for assigning individuals to tasks in which they need training [WO90]. This process development process is what we have referred to as process instantiation.

One example of separate application of resource constraints to instantiate an unconstrained process model can be seen in the use of the formal specification language LOTOS to model processes [SKS91]. In LOTOS, resource behavior (such as the numbers and interactions of participants) is described separately from the task behavior (i.e., unconstrained process model). The process model may be partially executed by executing either one of the two distinct viewpoints separately. This provides a natural way of describing a generic process independently of any specific resource allocation. It has been suggested informally by Bob Balzer that the general program transformation technique of *partial evaluation* (see [ACM91] for examples of partial evaluation) may be used in specializing process models for particular settings without actually instantiating them. A simple example of partial evaluation in a program translation context would be optimizing by loop unfolding when the value of the loop variable is known at compile time. A process model might be partially instantiated in stages by fixing values of various resources used by the process (e.g., numbers of participants, elapsed time permitted, total man hours permitted) one resource at a time.

3.5 Process Support by Automation

Basili notes that in addition to models of the development process, we need models of how users will employ the system being developed [BM91]. We divide these models into two categories: the first category consists of models that specify portions of well-understood

processes to be automated. Such processes permit the participation of human agents in certain pre-defined places in the processes. Examples of such systems include air traffic control systems, airline reservation systems, and military command and control systems. Modeling formalisms are commonly used in creating operations concepts, in systems analysis dealing with both system-level and software requirements, and in performance analysis of such systems.

The second category consists of models that specify processes that are not well-understood or by their nature are open-ended and not deterministic. The partial automation of such processes involves tools that may be invoked flexibly in sequences and combinations that are not pre-specified. It has been proposed [TBC+88] that the environment that controls the invocation of tools and access to data be driven by an enactable process model. Examples of such environments exist in software development, office automation, CAD, and other application domains.

3.5.1 System Specification

Colin Tully observes [Tul90b]: "Any system may be regarded as a process; and any process may be regarded as a system.... systems built by software engineers are processes and the systems within which they function are processes." Models of the latter are frequently referred to as system models rather than process models and the people who build and analyze them as systems engineers rather than process engineers. We may expect to find, therefore, formalisms in use in systems engineering that are of use in process engineering, and vice versa. In fact this is the case. Tully holds out the hope that

> ... building new kinds of processes and process models to support our own activity may lead to new insights into how we can build application systems, which are often to do precisely with the more effective management of *clients'* processes: in other words, we may find we have a rich new paradigm which we can apply outwardly in developing products, as well as inwardly for managing that development process [Tul90b].

A number of methodologies have been developed to define and design systems by modeling them. These models involve partitioning big systems into smaller systems and repeating the partitioning until small, implementable systems are reached. The methodologies are related by the relative emphasis they (and their modeling formalisms) place on each of three views (data, process, or control) in focusing in on the particular problem aspect the methodology addresses. No single development methodology addresses all problem domains and complex systems may need multiple models to fully describe their operation [Bra91].

A popular method of modeling used to perform systems analysis and specify functional requirements for information systems emphasizes first process (data transformation) aspects, then data (information modeling), and lastly behavioral (sequencing of states and actions). It is known as structured analysis and makes use of data flow diagrams and, to a lesser extent, entity-relationship diagrams. It has been noted [FGMM] that frequently the meanings of the data flow diagrams and the connection between the flows and entity-relationship diagrams

are not formally stated and that "the interpretation of these and other ambiguous or vague aspects is hidden within unstated assumptions made by the user of the notation". To overcome the lack of rigorous semantics for the model (particularly the inability to execute the specifications for analysis or to generate code automatically from the specifications), several approaches have been proposed to add the missing control information to data flow diagrams and produce a model better suited for specifying automation of a system. [FGMM] specifies a set of precise rules for basic synchronization and control activation in data flow diagrams, thus making them executable. [MDR87] makes data flow diagrams executable and permits code generation by providing an interpreter that interacts with the user in cases where the execution might proceed in more than one way. [War86] extends data flow diagram notation to represent control and timing by introducing signals for activation of data transformations emanating from separate logic specified using a finite state machine. The approach stops short of sufficient formality for execution but is actually an extension of the highly successful approach described next.

Harel [Har92] points out that over the last seven years several separate efforts [WM85, HP87, HLN+90] to extend information hiding and structured analysis concepts to reactive systems have resulted in surprisingly similar conclusions. Their joint result is what Harel refers to as a "vanilla" set of modeling concepts that represent system requirements in terms of both a functional and a behavioral description of the system. The functional description models the system in terms of data flow and the behavioral description models the system in terms of control flow and state changes. These models are developed in combination with and mapped to a structural or architectural model of the system that deals with subsystems, modules, channels, physical links, and storage components. The three vanilla modeling strategies, along with a fourth [BJKW88] resulting from an attempt to merge the strengths of each, are compared in [WW89]. The comparison found the methods to be largely similar and succeeded in identifying only small-grained differences.

The vanilla modeling approaches have been used extensively for specifying and analyzing the automation of reactive systems. For example, [LHH+91] describes use of STATEMATE modeling to develop a requirements specification for an aircraft collision avoidance system. It was found that behavioral modeling produced a specification of the system judged by experts to be easier to understand and to find errors in than a pure functional model.

Variations on the vanilla modeling approach substituting petri net representations for the finite state machine approach used as a behavioral description in the vanilla techniques have been proposed. [FS91] is one such example.

Other modeling techniques have proved useful in analyzing existing manual or partly automated systems prior to more completely automating them. Structured Process Flows (SPFs) [Tam91] feature imperative programming-like structured control flow constructs, such as conditionals and various looping constructs, annotated with data flow information, performing agents, and various metrics. They have been used, in connection with entity-relationship diagrams in efforts such as: a quality auditing support system that unified and integrated a set of PC-based, scattered operations; a new document processing system planned to handle anticipated millions of pages of documentation including people, interfaces, processes, and

automated support; and development of a functional specification of a command and control system decomposed for three echelons of command, control and information processing.

3.5.2 Automatic Tool Invocation in Design Environments/Frameworks

We turn now to automated support for processes that rely heavily on human creativity for their accomplishment. Due to their open-ended nature, the models which support the automation of these processes have different characteristics. These processes are frequently characterized by design or problem solving activities. The process that has received the most attention and the one that has driven work on this use of process modeling is the software development process.

Watts Humphrey states that the basic reason for defining the software process is "not to improve creativity but to free it by efficiently handling the details [Hum90]." He goes on to say that a well controlled and supported process should handle routine tasks routinely. Mundane tasks such as change management, design control, and interface coordination should be handled smoothly and efficiently so that designers do not become preoccupied with the administrative consequences of their work rather than devoting their energies to more creative activities. In addition to reducing mental load and tedium for team members, it has been suggested that automated support for the software process should impose constraints to improve consistency as well as enforce project and organizational policies [DNR90]. The same policies, mechanisms, and structures realized in a process model to support and enforce multi-person coordination and cooperation should also be used to bound the effects of complexity in the evolution of large-scale systems [Per90]. Another characteristic of the automation to support the software process is the facilitation and direction of manual activities [Sut91a].

Feiler [Fei90] distinguishes between *fully enacted* processes in a software development and *manually* enacted processes. Software configuration management (SCM) is cited as a portion of the software process that is manually enacted (enacted by humans following procedures and supported by the environment by a set of configuration management operations). Since there are a number of models of configuration management (checkout/checkin, transaction, change set, composition) that may be appropriate for different settings, the environment should support this aspect of the software process with a set of low-level control primitives, carefully designed to allow a range of higher-level coordination functions to be implemented. These functions are the operations "executed by humans who map the high-level SCM process reflected in the SCM procedures into sequences of instructions executed by invoking series of SCM operations".

Feiler continues that fully enacted portions of the software process are exemplified by the build process (this applies to the "derivation of objects through automatic application of tools [Fei90]"). In the build process, a series of predefined tool invocations is automatically executed based on a model of the fully enacted portion of the process that constitutes structural information and derivation rules for the various objects involved. In addition to providing automated support, Feiler suggests that software process modeling formalisms can be applied to better understand the semantics of the services offered by an environment and their impact on the software process in the combined environment/human system [Fei90].

An early example of support for the build process is the Make tool [Fel79] for UNIX whose "Makefiles", containing explicit object names and detailed identification of derivation steps, are in Feiler's sense a model of part of the development process that has the environment as execution agent. Odin [CO90] builds upon Make concepts but stores (and subsequently interprets) a database of derivation information in the form of a derivation graph that applies to a type of objects rather than just explicitly named individual objects. By providing for automatic derivation requiring "the complex synthesis of many diverse tools and the creation of many intermediate objects", Odin provides a more sophisticated model of the build process that features the improved flexibility and ease of use furnished by enhanced object management capabilities in the modeling formalism. By doing this, the Odin modeling formalism defines a new use for process modeling formalisms: the integration of existing and proposed tools in a development environment.

One of the mechanisms for providing partial automation in support of software and other processes is the simultaneous and synchronous execution of a human-oriented development process and an executable model of the process (a process program). For example, the ARCHIPEL environment uses generic process models (in the form of templates for roles, tasks, work contexts, etc.) to prepare tools by integration of components and activates tools into work contexts when it receives appropriate task activation events [FO91]. The process models are expressed using a petri-net based formalism augmented with data flow and structure information [Hub91]. (ARCHIPEL is a pilot system developed within the Eurêka Software Factory project (ESF) [ESF89, Fer91], a large research effort funded by the multinational European Eurêka program.)

The Software Designer's Associate (SDA) [KKM+88] represents a research effort (conducted by a consortium of Japanese and American researchers) that is likewise directed toward establishing a framework for environment architectures with a specific focus on consolidation and integration of existing tools. Integration in SDA is to be achieved by use of software product, software process, and tool collection conceptual models. The process model describes software development as a set of activities (functions) that are decomposed into constituent activities. The activity's action may be performed by invoking a particular tool or sequence of tools or may be carried out by a developer without automated assistance. Tools for carrying out a particular function are modeled by a data flow model that gives a tool's input and output object types.

Arcadia is an American (DARPA-sponsored) research effort investigating software development environment architecture issues. These issues include environment architectures for organizing large collections of tools and facilitating their interaction with users and each other supported by user interface management and object management components as well as tools to facilitate testing and analysis of software [TBC+88]. To address the apparently conflicting goals of providing flexibility/extensibility and a high degree of integration, the Arcadia consortium's efforts have focused on the notion of driving the environment's interactions by a formal, executable representation of the user's development and maintenance processes (process programs). In such a process-driven environment, flexibility is obtained by supporting alterations to process programs. Extensibility is achieved by writing new process programs

or by modifying existing process programs to incorporate new tools, subprocesses, types or objects.

In Arcadia research the process modeling formalism, the process programming language, is a full programming language. One of the process programming languages proposed is APPL/A, the Ada language extended with abstract, persistent relations with programmable implementations, relation attributes that may be composite and derived, triggers that react to relation operations, optionally-enforcible predicates on relations, and composite statements with transaction-like capabilities [SHO90]. Another proposed process programming language is P^4 [Hei89], based on Prolog. A process program indicates how the various software tools would be coordinated to support a process [TBC+88]. Originally, it was proposed (abstractly) that the process program drive the environment through interpretation by a "process program interpreter" [TBC+88]. Recent work, however, has proposed the separation of process programming language from the state of the process itself. The latter would be stored by a "process server" [Hei91]. In this way, multiple styles of process program (prescriptive and proscriptive) and multiple process programming languages can coexist in an environment.

Other efforts applying process modeling to the domain of software development include the ALF project's work to develop a process-centered software environment under the European ESPIRIT research and development program [Oqu90]. A formalism known as the Model for Assisted Software Processes (MASP) concept rigorously describes computer-assisted software process models and enacts them in an environment based on the PCTE public tool interface.

Examples of process-driven environments exist in other application domains than software development. One example is the Process Support System (PSS) [BPR91] built by a collaboration including STC Technology, Ltd., the University of Manchester, and ICL in the United Kingdom. PSS was applied to the problem report-handling activities of a Customer Service Department. Diagnosticians select the oldest request for service, assess whether it can be addressed immediately or must be sent elsewhere for analysis, and take the appropriate action. A database of known problems is available for consultation and work pools of the diagnosticians must be monitored to ensure that queues do not become too long. Line managers resolve overloading problems by reassigning staff. The process was coded using the PML process programming language [Rob88, RJ89]. (PML and PSS are outgrowths of the British IPSE 2.5 project. PML has been augmented with features to facilitate imperative programming rather than the declarative style of programming for which the original was intended.) Mundane activities like time-stamping the requests and monitoring work pools were automated. Automatic selection of the next request to be serviced provides an example of ensuring conformance to process. Access to up-to-date information about the state of the process allowed the line manager to make better decisions on how to allocate the available resources. The intricacies of retrieving information from the reference database were handled by the process program. The net effect was to allow the diagnostician to concentrate on solving problems and to simplify the diagnostician's training [BPR91]. Other applications of PSS have been made to support software development, health services, investment management, and project management [War91].

The developers of the process program described above, in evaluating their experience,

warn that the decision to introduce process execution must be based on sound business analysis and an analysis of the current process. Depending on differing levels of process maturity and other issues, different solutions to process improvement may be more cost effective or otherwise more appropriate. Rather than conformance to process, however, the developers believe that

> ... the true importance of process enactment is that it makes a new set of processes viable. These are processes that take advantage of the close integration of tools and participants within a typed environment. They analyse and act on the process data in a way that would interfere unacceptably with the participants' work, had the process not been automated.

They conclude that "the designer of a new process should seek out such benefits rather than merely replicating the existing process."

Based on PSS experience, Warboys [War91] has concluded that this use of process models may itself be broken into two categories of usage. For the first category, the process model is used for the fine-grained modeling of specific processes. The execution agents that are integrated are people, members of a team in an organization. In the second category, the model is used as an integration framework for disparate tools and databases. Concrete concerns such as format conversions, protocols, communication must be addressed. The latter, "architectural" view of process modeling, however, also extends to the human organizational domain. Just as a windowing environment can contribute to perceived integration of tools as end-users cut and paste between application windows, the process model can contribute to perceived integration by "essentially acting as a guidance system for a set of community-wide windows into some shared corporate model" [War91]. After discussing the major differences in the features of models for these two purposes, Warboys then reiterates the theme of this paper: namely that "the role of the process model in these terms is probably the most significant decision to be taken prior to the introduction of [a] given process model".

4 Understanding the Process Modeling Formalism Life Cycle

We have given a description of each major category of process model use and described some of the applications that fall under that use at the same time indicating some of the formalisms used for those applications. A wide range of formalisms have been designed for or applied to process modeling. These include process modeling languages of various programming language paradigms, charts and diagrams having the expressive power of regular expressions, and formal specification languages (including state machines, petri nets, and LOTOS). As with any artifact built by humans, they could be compared in a variety of ways according to a number of different sets of criteria (the dimensions of their design space). One could consider *characteristics* of the formalisms that are *stressed* such as human readability, precision, or ability to represent the various concepts identified at the beginning of the paper. One could also consider characteristics that are *addressed* regardless of the particular design choice

made such as object management issues, human-machine interaction issues, etc. One could also consider particular *capabilities* possessed by formalisms such as executability, ability to be analyzed formally, ability to account for unforeseen events, ability to handle rework/error conditions, ability to support abstraction, ability to support parallelism, or possession of a graphic representation. These high-level design choices in turn lead to a finer-grained set of choices that include the *paradigm* employed by the formalisms (for example, rule-based/logic programming, functional/hierarchical decomposition, entity-relationship data modeling, or state-based representations). Following these considerations are still finer choices such as the particular *constructs* provided by a formalism in its chosen paradigm. In another paper, Ziv and Osterweil [ZO92] discuss a lifecycle for process modeling formalisms and present a survey of a number of formalisms with respect to: the uses the formalism was intended to serve (its requirements), its chosen paradigm (its design), and its linguistic features (its implementation). For the purpose at hand, we construe the term "requirement" rather broadly, including the levels of abstraction we referred to above as requirements, characteristics, and capabilities.

As one might conclude from the preceding discussion, there are a number of different characteristics found in the formalisms described in the literature, and, correspondingly, a variety of different requirements for the design of process modeling formalisms have been given. The nature of these requirements is sensitive to the characteristics of the users of the model (e.g., Kellner [KFF+91], in the case of software processes, has identified different usage characteristics for software managers, software engineers, and process engineers) and to the model's domain of application (e.g., software development has been said to require models capable of representing unusual complexity and dynamism). We claim, however, that the chief reason for this wide range of requirements is the need for different capabilities in order to support the various different uses to which the formalism in question is intended to be put. In other words, one should design a formalism quite differently for one use than for some other. The intended use for a formalism is sometimes but not always explicitly noted by the designer; often there is only an implicit use for the formalism. In this section we attempt to rationalize the diverse requirements that have been expressed by relating them to the categories of use given in the paper. The exercise of classifying an assortment of requirements according the use categories will also serve to assess the completeness and usefulness of the set of categories we have chosen. While the list of requirements we give is not comprehensive, we have attempted to make them at least representative.

Requirements for process modeling formalisms have been given in [EG91, SHO90, KH88, HK89, Kel91a, Kel88, KH89, Per90, KKM+88, War91, Fei90, Tul88a, Tul88b, KFF+91, Leh89]. We will identify each requirement and then classify it by the uses of process modeling to which we feel it applies.

1. Tully in 1988 identified the following requirements:

   (a) Process formalisms should be enactable, providing human beings involved in the software process computer guidance and assistance on-line while processes are being carried out, as a means of directing, controlling, monitoring and instrumenting

them [Tul88a]. (This is practically a definition of the control by process guidance use of formalisms.)

(b) Process programs are written neither to mechanize software production or wholly prescribe what humans in the process are to do. They are written to "define possible (allowable) patterns of behavior between non-deterministic human beings and systems constructed of computer programs" [Tul88a]. (Also representing a control by process guidance use of formalisms.)

(c) It is generally agreed that models must have the following capabilities: active support, hierarchy, inheritance, multiple forms of display (especially including graphics), nondeterminism, and parallelism [Tul88b]. (This list of capabilities seems consistent with both communication and control with the exception of "active support" which is applicable to control and support by automation.)

2. Also in 1988, the Software Designer's Associate consortium identified the following requirements in [KKM+88]:

(a) A software process conceptual model should "support the definition of a set of tools to populate a Software Designer's Associate."

(b) It should "provide for independence between tools and details of specific software methods."

(c) It should "incorporate a distributed, team-based view of software creation and evolution."

(d) It should "reflect a broad range of current software methods."

(e) A process description should specify "what activities should be performed and how they are related".

(f) The process model should possess facilities to "determine dependencies among activities and use this information to determine which activities must be performed serially and which can be performed in parallel".

(g) The process model should make it possible to "show which activities are to be performed at a given time together with information on which tools are to be used. If several activities can occur, concurrency control is provided...."

(h) The process model must make it possible for its implementation to "keep track of the status of various activities and provide feedback on the status of the design process."

All of the requirements expressed here are clearly advantageous to the use of process models for automatic tool invocation. Several of the requirements are also useful for other purposes. Determining activities that can be performed in parallel could also be a requirement for formalisms used for constructing improved processes from existing (more serial) ones. Keeping track of process status and providing feedback is a requirement for process guidance and for project management.

3. Lehman's extensive 1989 list of purposes for process models [Leh89] consists of requirements that may be assigned to one or more of all the various categories of process model usage identified in this paper.

4. Kellner and associates at SEI have identified requirements for process modeling formalisms in an array of papers [Kel88, HK89, KH89, KFF+91, Kel91a] over the last four years. They have been summarized in the following list of "requirements for an ideal approach to software process modeling", found in both [KH89] and [KH88]:

   (a) The formalism should use a highly visual form of information representation (e.g., diagrams). (An important requirement for the use of process modeling formalisms to communicate and in vizualizing current status and progress for project management.)

   (b) It should enable compendious descriptions by being comprehensive in scope yet concise in presentation so that complex aspects of a process can be represented easily. (This is useful for both communication and process construction.)

   (c) It should support multiple, complementary views of the process. In particular, the following views are quite useful:

   - A functional view, focused on the main activities and the data that flows between them.
   - A behavioral view (describing when and how these activities are accomplished), capable of representing feedback loops, iteration, complex decision-making conditions, entry criteria/trigger conditions, exit criteria, and precedence relationships. It should also represent parallelism at the level of individual software objects (e.g., both testing/debugging and documentation revision can occur concurrently for the same software module) and parallelism at a level that crosses objects (e.g., some modules may be in test while others are still in coding).
   - An implementation view (describing by whom and where the activities are implemented), connecting activities with the organizational subunits performing them and describing communication channels.
   - A conceptual data modeling view (providing an abstract, global of data regarding the software objects being produced and the process itself), connecting process and product metrics to the process representation.

   (This requirement is needed for capturing a complete process and then providing views for multiple uses: for example, a functional view for communicating it; a behavioral view for process improvement, process guidance, and automatic tool invocation; and implementation view for process control; and a conceptual data modeling view for improvement.)

   (d) It should support multiple levels of abstraction/hierarchical decomposition for each view, providing global to low levels of detail. (Process construction from components and system specification uses of a formalism require a variable level of detail

in the process description. This requirement is also necessary in order to support multiple uses with the same model such as communication and project management at the high end and process guidance and automatic tool invocation at the low end.)

(e) It should offer a formally defined syntax and semantics so that it can be machine parsed and analyzed semantically. This also opens the posssibility of automatic simulation or execution of the description. (This requirement is the essential one for process guidance and for automatic tool invocation.)

(f) It should provide comprehensive analysis capabilities for evaluating consistency, completeness, and correctness. (Detecting these types of anomalies is primarily useful in model building itself but may also be used in process construction.)

(g) It should facilitate the direct qualitative and quantitative simulation of process behavior from the description. (This type of analysis is useful for process improvement, project management, and system specification.)

(h) It should support the creation and management of variants, revisions, and reusable components of process descriptions, making reuse of portions of descriptions and identification of differences between descriptions easy. (This is clearly important to process construction.)

(i) It should support the representation and analysis for compliance of constraints on the process such as standards, required documents, and required review points. (This analysis is important in constructing a process or in instantiation of a model as part of project management.)

(j) It should enable the representation of purposes, goals, rationales, and intentions for process components and the overall process. (This is crucial to process improvement efforts.)

(k) The process model should integrate easily with other approaches to process representation which may be deemed useful such as PERT or CPM for critical path analysis or entity-relationship modeling for a conceptual data model. (This is important for acceptance of the model by its potential users in any of the categories of use that involve direct interaction with the model, as opposed to indirect through a process-driven software development environment.)

(l) The model should take an active role in process execution, possibly by automatically recording the steps taken and objects manipulated during actual execution or by providing guidance during execution based on the descriptions. (This is practically the definition of use of a model for process guidance.)

(m) The formalism's environment should offer automated tools supporting the approach. (Applicable to model building itself.)

5. Perry [Per90] identified the following requirements in 1990:

(a) "What is needed to support the highly dynamic, interactive and incremental [software] evolution and integration process is a model ... with several subprocesses that are dynamically instantiatable and modifiable for each of the particular instance or class of instances." The application of the activities described in the subprocess model, the instantiation of its various instances, and the determination of process state depends on "aspects of the project management structure, the system architecture and design structure, and the various mechanisms and structures being used to implement the process model".

(b) "Appropriate process model/instantiation support is needed for ... evolution of the process model and various instantiations [that] may occur as a result of redefining policies, changing the policies concerning various activities, changing the degree of enforcement or support, or changing the underlying process support structures or mechanisms."

These requirements seem applicable to process construction as well as process guidance and automatic tool invocation.

6. Feiler, in his 1990 discussion of needs for software process support [Fei90], requires the infrastructure of software development environments to support:

   (a) Software processes fully enacted by an environment (e.g., the build process).

   (b) Manually enacted processes (e.g., software configuration management).

   (c) Dynamic enactment of processes (i.e., possessing flexible primitives for events and actions on objects and tasks). The primitives used for this may involve different degrees of control for performing change, different degrees of coordination, different scopes of visibility of change, different degrees of consistency to be asserted by the environment, and object modification rights that may be reassigned to permit changes in the resource allocation of the software process.

   These are requirements on the modeling formalism that are concerns largely of automatic tool invocation.

7. Warboys of the IPSE 2.5 project has recently identified the following requirements for process modeling environments but also for the languages they use in [War91]:

   (a) "...the process modeling environment must "naturally integrate" with the rest of the Information System being used.... We are indeed faced already with the problem of introducing a Process Modeling Environment which interworks with disparate and distributed (through different tools) fragments of process models." (This is primarily a consideration for automatic tool invocation.)

   (b) "...the process modeling environment must incorporate the means of changing itself whilst executing and also allow for end-user specialisation of the process under identifiable constraints." (This is important for accurate process guidance as well as automatic tool invocation.)

(c) A key role of process models (and hence a requirement on their formalisms) "is as a framework for the integration of disparate tools and databases." (This is practically the definition of the use of process models for automatic tool invocation.)

(d) Another principle role of process models is as an auditing mechanism to record the course the process' execution took, including the rationale for decisions that were made and lessons learned. If, as it appears in many cases, this audit role is more important than the control of what will happen next, this clearly has a significant effect on the language used for process modeling. (This is one of the key types of analysis used for process improvement.)

8. Emmerich and Gruhn of the MELMAC project at the University of Dortmund have identified the following requirements in [EG91]:

   (a) "In software process models several situations exist in which it is unimportant in which way something is done. Instead it is important that it is done in one way or the other. Modeling this kind of non-determinism is a crucial issue." (This is important for process construction and for creating accurate plans to support project management and guidance.)

   (b) " It is necessary to model that several activities can be executed concurrently. This must be expressible in software process modeling languages. The representation of concurrent activities, for example, can help to find out how many people can be deployed, this it can be the basis for personnel management." (This is clearly relevant to project management but also to process improvement.)

   (c) "Analysis of software process models can contribute to the early detection of errors. By analyzing software process models it is possible to prove specific properties of these models, to detect errors, and to gain deeper insights into the nature of the analyzed software process model." (Such analysis is needed to support many uses of formalisms but is especially needed for process construction.)

   (d) "A software process modeling language must enable a tight representation of software process models, since software process models are usually quite complex. In order to keep this complexity manageable it is necessary to represent basic entities as single units of description." (This is important for both communication and in specifying tools for automatic invocation.)

9. Finally, Sutton, Heimbigner, and Osterweil of the Arcadia consortium suggest the following desireable characteristics of software process programming languages for support of change to environments in [SHO90]:

   (a) "Explicit representation of both objects and inter-object relationships."

   (b) "Explicit representation of the semantics of objects and relationships, including constraints and derivations."

(c) "Automation of as much of the change process as is feasible, including propagation of data, maintenance of consistency, and invocation of tools."

(d) "Abstraction of processes, objects, and relationships from the underlying implementation system. At the logical level change management should be independent of the implementation, and changes to the implementaiton should not affect the abstract representation of development processes and products."

These requirements are all applicable to the automatic tool invocation aspect of process support by automation.

In table 1 we summarize the classification of each requirement in the above list by its applicable categories of process modeling usage. From the correspondence noted between requirements and categories of process modeling usage, we can see that selection of requirements for process modeling formalisms made by different authors is generally uniform in the usages to which the any single author's requirements apply (i.e., all of each individual author's requirements apply to a very small set of usages). We also found it easy to find one or more places for each requirement within the set of uses for process formalisms that we had defined. We conclude that the categories of usage draw meaningful distinctions (corresponding to the intended application of the author's work) and that the set of categories is complete with respect to requirements expressed in the literature for modeling formalisms. We feel that these comments also hold for the other artifacts of the process modeling formalism life cycle such as design features and implementation constructs of formalisms.

5 Future Work

Process engineering will involve the selection and tailoring of appropriate formalisms and their use either separately or in combination to realize the applications outlined in this paper. Several steps necessary in the development of the knowledge required by process engineering are already apparent. They are discussed below.

5.1 Developing Guidelines for Choice of Formalism

How should we choose a formalism for a given project? It should be clear from the survey of uses of process modeling formalisms given in this paper that a central issue in process engineering is the choice of a formalism appropriate for the particular use at hand. Guidelines relating characteristics and capabilities to specific uses or combinations of uses would greatly aid in the systemization of process engineering and its evolution into an engineering discipline. One approach to developing such guidelines could be accomplished in two steps: first, categorize design choices made in the design of formalisms according to the choices that would support each of the uses identified; second, catalog the design choices that have been made in the various formalisms that have been described in the literature independently of their intended use. Formalisms' profiles of design choices could then be matched against those

Requirements for Formalisms		Communication	Construction	Control		Automation	
				Proj. Mgt.	Guidance	Sys. Spec.	Tool Inv.
1	a Tully				X		
	b				X		
	c	X			X		X
2	a Katayama						X
	b						X
	c						X
	d						X
	e						X
	f						X
	g		X				X
	h			X	X		X
3	Lehman	X	X	X	X	X	X
4	a Kellner	X		X			
	b	X	X				
	c	X	X	X	X		X
	d	X	X	X	X	X	X
	e				X		X
	f		X				
	g		X	X		X	
	h		X				
	i		X	X			
	j		X				
	k	X	X	X	X	X	
	l				X		
	m						
5	a Perry		X		X		X
	b		X		X		X
6	a Feiler						X
	b						X
	c						X
7	a Warboys						X
	b				X		X
	c						X
	d		X				
8	a Emmerich		X	X	X		
	b		X	X			
	c		X				
	d	X					X
9	a Sutton						X
	b						X
	c						X
	d						X

Table 1: Requirements for Process Modeling Formalisms and the Uses to which they apply.

identified as desirable for a particular use. The result would be somewhat objective identification of formalisms that should be useful for various purposes. This result would require testing by appropriate modeling activities.

A second approach to developing guidelines would be to develop a process description for each of the process modeling uses identified in the paper. Following the approach of [SO91] for software design methodologies, the various formalisms that might be applicable to a particular use might be fit into a process of applying them and decomposed and compared to the generic process for the use. The strengths and omissions of the formalisms for supporting a particular use could then be evaluated.

5.2 Tailoring a Formalism Using Complementary Approaches

It is a general characteristic of the discipline of engineering that models are used that each make a cost-effective tradeoff of available modeling features to understand salient points and reach a conclusion about what to build. This is true in particular of process engineering. Many of the features provided by a given formalism are dependent on the particular paradigm employed by the formalism for representing processes. The wide variety of paradigms have been employed in the formalisms we have surveyed intended to address various uses of process modeling. It may happen, then, that for a particular use of a modeling formalism, the features of more than one paradigm should be employed in combination. Certainly no single paradigm provides the modeling capabilities required for all the uses we have surveyed. In fact, for each given use, different formalism capabilities may be desirable. As Schäfer notes: Petri-nets support graphic visualization and analysis but do not support frequent dynamic changes or efficient execution. Rules, on the other hand, lack a graphic visualization, and so on ... [Sch91]. As a further example, we previously noted that system modeling for automation involves formalisms having features of state-based ("control"), data flow ("process"), and structural ("data") paradigms to different extents. If we extend the methodology used for system modeling, we might want to tailor the formalism we use (e.g., Hatley-Pirbhai and SPFs; DFDs with E-Rs or state diagrams). Liu and Conradi [LC91] point out that several of the process modeling formalisms we have mentioned previously have such "hybrid" paradigms: the SPECIMEN system tries to merge FUNSOFT nets and the rule-based MERLIN process modeling language; the environment-relation (ER) nets of Bandinelli, Ghezzi, and Morzenti [BGM91] feature a Petri net model of control with activities modeled in the transitions by relations implemented in a logic programming language and an environment of structured data carried in the places by tokens; EPOS combines a static rule-based (AI planning) paradigm with dynamic triggering in a task network or graph/net paradigm. For a given purpose, then, multiple paradigms might be employed to advantage to describe different aspects of the same process. This is Schäfer's view of a multi-paradigm process: namely, the end-to-end process model is decomposed into smaller parts which are described using different paradigms [Sch91].

It should also be clear from the discussion in this paper that multiple objectives for modeling may lead to the same process (and the same parts of the process) being described by alternative models that coexist yet employ different paradigms. For example, various kinds of process analysis demand use of process descriptions employing different paradigms as dif-

fering views of the same process. Meyers and Reiss [MR91], addressing software development using different paradigm-specific views of software, state that the software development environment "should support this natural plethora of views, should allow developers to create new views without undue difficulty, and should automatically maintain consistency between disparate views of the same system, even while the views are being modified." To avoid the problem of needing to write N^2 inter-view translations for N types of views, they propose a single canonical, semantics-based representation for software at the core of the environment with just N translators required for N views. Exactly the same considerations hold for the various views of a process maintained by what Sutton refers to as a Software Process Environment [Sut91b]. Heimbigner [Hei91] discusses the use of a process server component to enable multiple styles of process programs and multiple process programming languages (possibly employing different paradigms) to usefully coexist and drive a software development environment. In this approach, the process instance would be independent of the particular process programming language (or languages) used to represent the process. The process server stores the paradigm-independent state of the process as it is executed.

Heimbigner notes that even single-paradigm process descriptions may use multiple distinct representations of a process (e.g., APPL/A tasks for directly executing process code and APPL/A relations as an explicit representation of the attributes and structure of the tasks that can be manipulated dynamically by the process. These single-paradigm descriptions use embedded maintenance operations in the code representation of the process and triggers associated with the explicit structural representation to maintain consistency among the representations. The process server idea elaborates on this use of an explicit process state, describing it and maintaining it independently of languages that manipulate it.

Both Scott Meyers' and Heimbigner's proposals/experiments suggest the need for and feasibility of a central canonical form that both forms the basis for static translation and maintains execution state. Meyers has experimented with the use of a data flow graph similar to the ones used as object code for the MIT tagged-token dataflow architecture [AN90]. The data flow graphs are used as a canonical form for translation from Petri-net to Statechart formalisms and back. The data flow-based and highly parallel language Id (for *I*rvine *d*ataflow) [AGP78] and the data flow graph formalism into which Id is designed to be compiled deserve investigation as a candidate canonical representation to facilitate use of multi-paradigm process representations. The development of suitable architectural components and canonical representations, then, are areas of future work that would help give process modelers the ability to tailor formalisms from existing approaches to support the unique requirements of each process engineering problem.

6 Conclusions

From the survey of its uses we have given above, we can see that process modeling technology is:

- Rapidly developing

- Broadly applicable both within and outside software engineering
- An enabling technology that, if properly applied, can result in significant benefits to a wide range of activities. Its proper application, however, is a matter of engineering judgement, and there is no substitute for the skill of the engineer applying it to solve real-world problems.

Further, we have developed a list of uses for process modeling formalisms that is both useful and sufficient for understanding their differences.

References

[ACM91] ACM SIGPLAN. *Proceedings of the Symposium on Partial Evaluation and Semantics-Based Program Manipulation*, Yale University, New Haven, Connecticut, June 1991. ACM Press. Appeared as *SIGPLAN Notices 26*(9).

[AGP78] Arvind, Kim P. Gostelow, and Wil Plouffe. An asynchronous programming language and computing machine. Technical Report TR-114a, University of California, Irvine, California, December 1978.

[AHM91] Tarek Abdel-Hamid and Stuart E. Madnick. *Software Project Dynamics: An Integrated Approach.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1991.

[Amb91] Vincenzo Ambriola. The oikos process modeling paradigm. In A. Fuggetta, R. Conradi, and V. Ambriola, editors, *First European Workshop on Software Process Modeling*, Milan, Italy, May 1991. CEFRIEL, Politecnico di Milano, A.I.C.A. Working Group on Software Engineering.

[AN90] Arvind and Rishiyur S. Nikhil. Executing a program on the MIT tagged-token dataflow architecture. *Transactions on Computers*, 39(3):300–318, March 1990.

[Bal90] Robert Balzer. What we do and don't know about software process. In *Proceedings of the 6th International Software Process Workshop*, pages 61–64, Hakodate, Japan, October 1990. IEEE Computer Society Press.

[BGM91] Sergio Bandinelli, Carlo Ghezzi, and Angelo Morzenti. A multi-paradigm petri net based approach to process description. In *7th International Software Process Workshop — Preprints*, Yountville, CA, October 1991.

[BJKW88] W. Bruyn, R. Jensen, D. Keskar, and P. Ward. ESML: An extended systems modeling language. *ACM SIGSOFT Software Engineering Notes*, 13(1):58–67, January 1988.

[BM91] Victor R. Basili and John D. Musa. The future engineering of software: A management perspective. *IEEE Computer*, 24(9):90–96, September 1991.

[Boe81] B. W. Boehm. *Software Engineering Economics.* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1981.

[BPR91] R. F. Bruynooghe, J. M. Parker, and J. S. Rowles. PSS: A system for process enactment. In *Proceedings of the First International Conference on the Software Process*, pages 128–141, Redondo Beach, CA, October 1991. IEEE Computer Society Press.

[Bra91] Dennis L. Brandl. Modeling and describing really complex processes. *Texas Instruments Technical Journal*, 8(3):21–27, May–June 1991.

[CKI88] Bill Curtis, Herb Krasner, and Neil Iscoe. A field study of the software design process for large systems. *Communications of the ACM*, 31(11):1268–1287, November 1988.

[CKSI87] B. Curtis, H. Krasner, V. Shen, and N. Iscoe. On building software process models under the lamppost. In *Proceedings of the Ninth International Conference on Software Engineering*, pages 96–103, Monterey, CA, March 1987.

[CLW90] Reidar Conradi, Chunnian Liu, and Per H. Westby. Epos pm: Planning and execution. In *Proceedings of the 6th International Software Process Workshop*, pages 73–75, Hakodate, Japan, October 1990. IEEE Computer Society Press.

[CO90] Geoffrey M. Clemm and Leon J. Osterweil. A mechanism for environment integration. *ACM Transactions on Programming Languages and Systems*, 12(1):1–25, January 1990.

[Def91] Defense Systems Management College, Decison Support Systems Directorate, Fort Belvoir, Virginia. *The Program Manager's Support System (PMSS): An Executive Overview and Description of Functional Modules*, fourth edition, February 1991.

[DNR90] Mark Dowson, Brian Nejmeh, and William Riddle. Concepts for process definition and support. In *Proceedings of the 6th International Software Process Workshop*, pages 87–90, Hakodate, Japan, October 1990. IEEE Computer Society Press.

[DNR91] Mark Dowson, Brian Nejmeh, and William Riddle. Fundamental software process concepts. In A. Fuggetta, R. Conradi, and V. Ambriola, editors, *First European Workshop on Software Process Modeling*, Milan, Italy, May 1991. CEFRIEL, Politecnico di Milano, A.I.C.A. Working Group on Software Engineering.

[Dow86] M. Dowson, editor. *Proceedings of the Third International Software Process Workshop*, Breckenridge, Colorado, November 1986.

[Dow91a] Mark Dowson, editor. *Proceedings of the 1st International Conference on the Software Process*, Redondo Beach, CA, October 1991. IEEE Computer Society Press.

[Dow91b] Mark Dowson. Process and project management. In *7th International Software Process Workshop — Preprints*, Yountville, CA, October 1991.

[Dow91c] Mark Dowson. Why is process important?: Panel introduction. In *Proceedings of the First International Conference on the Software Process*, page 2, Redondo Beach, CA, October 1991. IEEE Computer Society Press.

[EG91] Wolfgang Emmerich and Volker Gruhn. Funsoft nets: A petri-net based software process modeling language. In *Proceedings of the Sixth International Workshop on Software Specification and Design*, pages 175–184, Como, Italy, October 1991. IEEE Computer Society Press.

[ESF89] Eureka Software Factory Consortium ESF. Technical reference guide, June 1989.

[FCA91] A. Fuggetta, R. Conradi, and V. Ambriola, editors. *First European Workshop on Software Process Modeling*, Milan, Italy, May 1991. CEFRIEL, Politecnico di Milano, A.I.C.A. Working Group on Software Engineering.

[Fei90] Peter H. Feiler. Software process support in software development environments. In *Proceedings of the 6th International Software Process Workshop*, pages 91–94, Hakodate, Japan, October 1990. IEEE Computer Society Press.

[Fel79] Stuart I. Feldman. Make—a program for maintaining computer programs. *Software — Practice & Experience*, 9(4):255–265, April 1979.

[Fer91] Christer Fernström. The eureka software factory: Concepts and accomplishments. In A. van Lamsweerde and A. Fuggetta, editors, *ESEC '91: 3rd European Software Engineering Conference*, Milan, Italy, October 1991. Springer-Verlag. Lecture Notes in Computer Science.

[FGMM] Alfonso Fuggetta, Carlo Ghezzi, Dino Mandrioli, and Angelo Morzenti. Executable specifications with data flow diagrams. unpublished.

[FO91] Christer Fernstrom and Lennart Ohlsson. Integration needs in process enacted environments. In *Proceedings of the First International Conference on the Software Process*, pages 142–158, Redondo Beach, CA, October 1991. IEEE Computer Society Press.

[Fra91] Dennis J. Frailey. Defining a corporate-wide software process. In *Proceedings of the First International Conference on the Software Process*, pages 113–121, Redondo Beach, CA, October 1991. IEEE Computer Society Press.

[FS91] Helmut Franzen and Günther Siegel. Die methode der strukturierten analyse mit petri-netzen (SA/PN) als echtzeitweiterung. In M. Timm, editor, *Proccedings Requirement Engineering '91*, pages 178–190, Marburg, Germany, 1991. Springer-Verlag. Informatik-Fachberichte, Band 273.

[Gru90] Volker Gruhn. Analysis of software process models in the software process management environment MELMAC. In Fred Long, editor, *Software Engineering Environments*, volume 3, pages 67–90. Ellis Horwood, Ltd., New York, 1990.

[Gru91] Volker Gruhn. Validation and verification of software process models. In A. Endres and H. Weber, editors, *Software Development Environments and CASE Technology: European Symposium*, pages 271–286, Königsberg, Germany, June 1991. Springer-Verlag. Lecture Notes in Computer Science, Volume 509.

[Har92] David Harel. Biting the silver bullet: Toward a brighter future for system development. *IEEE Computer*, 25(1):8–20, January 1992.

[Hei89] Dennis Heimbigner. P^4: A logic language for process programming. In *Proceedings of the 5th International Software Process Workshop*, pages 67–70. IEEE Computer Society, 1989.

[Hei90] Dennis Heimbigner. Proscription versus prescription in process-centered environmwents. In *Proceedings of the 6th International Software Process Workshop*, pages 99–102, Hakodate, Japan, October 1990. IEEE Computer Society Press.

[Hei91] Dennis Heimbigner. A process server. In *7th International Software Process Workshop — Preprints*, Yountville, CA, October 1991.

[HFB90] Laurence Hubert, Frederic Fournier, and Maryse Bourdon-Le Brasseur. Eureka software factory: OPIUM an environment for software process modeling integrated with a project management tool. In *Proceedings of the 6th International Software Process Workshop*, pages 103–107, Hakodate, Japan, October 1990. IEEE Computer Society Press.

[HK89] Watts Humphrey and Marc Kellner. Software process modeling: Principles of entity process models. In *Proceedings of the Eleventh International Conference on Software Engineering*, Pittsburgh, May 1989.

[HLN+90] David Harel, Hagi Lachover, Amnon Naamad, Amir Pnueli, Michal Politi, Rivi Sherman, Aharon Shtull-Trauring, and Mark Trakhtenbrot. STATEMATE: A working environment for the development of complex reactive systems. *IEEE Transactions on Software Engineering*, 16(4):403–414, April 1990.

[HP87] Derek J. Hatley and Imtiaz A. Pirbhai. *Strategies for Real-Time System Specification.* Dorset House Publishing, New York, 1987.

[Hub91] Laurence Hubert. Eurêka software factory: OPIUM An environment for software process modeling integrated with project management and product management facilities. In A. Fuggetta, R. Conradi, and V. Ambriola, editors, *First European Workshop on Software Process Modeling*, Milan, Italy, May 1991. CEFRIEL, Politecnico di Milano, A.I.C.A. Working Group on Software Engineering.

[Huf89] Karen E. Huff. Software process instantiation and the planning paradigm. In Dwayne E. Perry, editor, *Proceedings of the 5th International Software Process Workshop*, pages 71–73, Kennebunkport, Maine, October 1989. IEEE Computer Society Press.

[Hum90] Watts S. Humphrey. People considerations in process models. In *Proceedings of the 6th International Software Process Workshop*, pages 113–115, Hakodate, Japan, October 1990. IEEE Computer Society Press.

[Hum91] Watts S. Humphrey. Why is process important?: Panel position statement. In *Proceedings of the First International Conference on the Software Process*, page 3, Redondo Beach, CA, October 1991. IEEE Computer Society Press.

[i-L89] i-Logix Inc. The STATEMATE approach to complex systems. 22 Third Ave., Burlington, MA, 1989.

[ISP88] *"4th Proceedings of the Software Process Workshop"*, Moretonhampstead, UK, May 1988.

[ISP89] *"5th Proceedings of the Software Process Workshop"*, 1989.

[ISP90] *Proceedings of the 6th International Software Process Workshop*, Hakodate, Japan, 29-31 October 1990. (to appear).

[ISP91] Rocky Mountain Institute of Software Engineering. *7th International Software Process Workshop — Preprints*, Yountville, California, October 1991.

[Jen81] Randall W. Jensen. A macro-level software development cost estimation methodology. In *Fourteenth Asilomar Conference on Circuits, Systems and Computers*, New York, 1981. Institute of Electrical and Electronic Engineers.

[Kai89] Gail E. Kaiser. Experience with Marvel. In *Proceedings of the 5th International Software Process Workshop*, pages 82–84, Kennebunkport, Maine, October 1989.

[KCI87] Herb Krasner, Bill Curtis, and Neil Iscoe. Communication breakdowns and boundary spanning activities on large programming projects. In Gary Olson, Sylvia Sheppard, and Elliot Soloway, editors, *Empirical Studies of Programmers: Second Workshop*, pages 47–64, Norwood, NJ, 1987. Ablex Pub Co.

[Kel88] Marc I. Kellner. Representation formalisms for software process modeling. In *Proc. 4th International Software Process Workshop*, pages 93–96, Moretonhampstead, UK, May 1988.

[Kel90] Marc I. Kellner. Supporting software processes through software process modeling. In *Proceedings of the 6th International Software Process Workshop*, pages 125–129, Hakodate, Japan, October 1990. IEEE Computer Society Press.

[Kel91a] Marc I. Kellner. Multiple-paradigm approaches for software process modeling. In *7th International Software Process Workshop — Preprints*, Yountville, CA, October 1991.

[Kel91b] Marc I. Kellner. Software process modeling support for management planning and control. In *Proceedings of the First International Conference on the Software Process*, pages 8–28, Redondo Beach, CA, October 1991. IEEE Computer Society Press.

[KFF+91] Marc I. Kellner, Peter H. Feiler, Anthony Finkelstein, Takuya Katayama, Leon J. Osterweil, Maria H. Penedo, and H. Dieter Rombach. ISPW-6 software process example. In *Proceedings of the First International Conference on the Software Process*, pages 176–186, Redondo Beach, CA, October 1991. IEEE Computer Society Press.

[KH88] Marc I. Kellner and Gregory A. Hansen. Software process modeling. SEI Technical Report CMU/SEI-88-TR-9, Carnegie-Mellon University Software Enginnering Institute, Pittsburgh, Pennsylvania, May 1988.

[KH89] Marc I. Kellner and Gregory A. Hansen. Software process modeling: A case study. In *Proceedings of the Twenty-Second Annual Hawaii International Conference on System Sciences*, pages 175–187, Kailua-Kona, Hawaii, January 1989.

[KKM+88] Kouichi Kishida, Takuya Katayama, Masatoshi Matsuo, Isao Miyamoto, Koichiro Ochimizu, Nobuo Saito, Hojn H. Sayler, Kiji Torii, and Lloyd G. Williams. Sda: A novel approach to software environment design and construction. In *Proceedings of the Tenth International Conference on Software Engineering*, 1988.

[KM91] Takuya Katayama and Sumio Motizuki. What has been learned from applying a formal process model to a real process. In *7th International Software Process Workshop — Preprints*, Yountville, CA, October 1991.

[Kri69] Edward V. Krick. *An Introduction to Engineering and Engineering Design.* John Wiley & Sons, New York, second edition, 1969.

[LC91] Chunnian Liu and Reidar Conradi. Process modeling paradigms: An evaluation. In A. Fuggetta, R. Conradi, and V. Ambriola, editors, *First European Workshop on Software Process Modeling*, Milan, Italy, May 1991. CEFRIEL, Politecnico di Milano, A.I.C.A. Working Group on Software Engineering.

[Leh89] Manny M. Lehman. The role of process models in software and systems development and evolution. In *Proceedings of the 5th International Software Process Workshop*, pages 91–94, Kennebunkport, Maine, October 1989.

[Leh91] M. M. Lehman. Why is process important?: Panel position statement. In *Proceedings of the First International Conference on the Software Process*, page 4, Redondo Beach, CA, October 1991. IEEE Computer Society Press.

[LHH+91] Nancy G. Leveson, Mats Heimdahl, Holly Hildreth, Jon Reese, and Ruben Ortega. Experiences using statecharts for a system requirements specification. In Nancy G.

Leveson and Richard W. Selby, editors, *Proceedings of the 1st Irvine Software Symposium*, pages 29–46. The Irvine Research Unit in Software, University of California, Irvine, June 5, 1991.

[MDR87] Reginald Meeson, Jr., Michael B. Dillencourt, and Audrey M. Rogerson. Executable data flow diagrams. In *Proceedings of the 1st International Workshop on Computer-Aided Software Engineering*, pages 445–454, Cambridge, Massachusetts, May 1987.

[MR91] Scott Meyers and Steven P. Reiss. A system for multiparadigm development of software systems. In *Proceedings of the Sixth International Workshop on Software Specification and Design*, pages 202–209, Como, Italy, October 1991. IEEE Computer Society Press.

[MS91] Peiwei Mi and Walt Scacchi. Modeling articulation work in software engineering processes. In *Proceedings of the First International Conference on the Software Process*, pages 188–201, Redondo Beach, CA, October 1991. IEEE Computer Society Press.

[MTH91] Nazim H. Madhavji, Kamel Toubache, and Won-Kook Hong. Communications and iterations in the process cycle. In *7th International Software Process Workshop — Preprints*, Yountville, CA, October 1991.

[MYKS90] Sumio Mochizuki, Akira Yamauchi, Takuya Katayama, and Masato Suzuki. Applying the software process to the instruction tool in system design. In *Proceedings of the 6th International Software Process Workshop*, pages 141–143, Hakodate, Japan, October 1990. IEEE Computer Society Press.

[Oqu90] Flávio Oquendo. Building object and process-centered software environments on the PCTE public tool interface. In *Proceedings of the 6th International Software Process Workshop*, pages 149–154, Hakodate, Japan, October 1990. IEEE Computer Society Press.

[Orm83] Donald O. Ormand. Private communication. March 1983.

[Ost87] Leon J. Osterweil. Software processes are software too. In *Proceedings of the Ninth International Conference on Software Engineering*, pages 2–13, Monterey, CA, March 1987.

[Ost91] Leon J. Osterweil. Why is process important?: Panel position statement. In *Proceedings of the First International Conference on the Software Process*, page 5, Redondo Beach, CA, October 1991. IEEE Computer Society Press.

[Per90] Dewayne E. Perry. Policy and product-directed process instantiation. In *Proceedings of the 6th International Software Process Workshop*, pages 163–166, Hakodate, Japan, October 1990. IEEE Computer Society Press.

[Pot84] C. Potts, editor. *Proceedings of the Software Process Workshop*, Egham, Surrey, UK, February 1984.

[PSW90] B. Peuschel, W. Schäfer, and S. Wolf. A flexible environment architecture as a basis for distributed software development. In *Proceedings of the 6th International Software Process Workshop*, pages 167–169, Hakodate, Japan, October 1990. IEEE Computer Society Press.

[PSW91] B. Peuschel, W. Schäfer, and S. Wolf. A consistency model for team cooperation. In *7th International Software Process Workshop — Preprints*, Yountville, CA, October 1991.

[Red90] Samuel T. Redwine, Jr. Organizational properties and software process models. In *Proceedings of the 6th International Software Process Workshop*, pages 171–173, Hakodate, Japan, October 1990. IEEE Computer Society Press.

[RJ89] Clive Roberts and Alun Jones. Dynamics of process models in PML. In Dwayne E. Perry, editor, *Proceedings of the 5th International Software Process Workshop*, pages 124–126, Kennebunkport, Maine, October 1989. IEEE Computer Society Press.

[Rob88] Clive Roberts. Describing and acting process models with PML. In Colin Tully, editor, *Proc. 4th International Software Process Workshop*, pages 136–141, Moretonhampstead, UK, May 1988. ACM Press.

[Rom89] H. Dieter Rombach. Specification of software process measurement. In *Proceedings of the 5th International Software Process Workshop*, pages 127–129, Kennebunkport, Maine, October 1989.

[Rue90] Michel Rueher. Formalizing operations and relationships on objects to support dynamic refinement of process models instances. In *Proceedings of the 6th International Software Process Workshop*, pages 185–189, Hakodate, Japan, October 1990. IEEE Computer Society Press.

[Sch91] Wilhelm Schäfer. Towards a multi-paradigm approach for software process design. In A. Fuggetta, R. Conradi, and V. Ambriola, editors, *First European Workshop on Software Process Modeling*, Milan, Italy, May 1991. CEFRIEL, Politecnico di Milano, A.I.C.A. Working Group on Software Engineering.

[SHO90] Stanley M. Sutton, Jr., Dennis Heimbigner, and Leon J. Osterweil. Language constructs for managing change in process-centered environments. In *Proceedings of ACM SIGSOFT '90: Fourth Symposium on Software Development Environments*, pages 206–217, Irvine, CA, December 1990.

[SKS91] Motoshi Saeki, Tsuyoshi Kaneko, and Masaki Sakamoto. A method for software process modeling and description using LOTOS. In *Proceedings of the First International Conference on the Software Process*, pages 90–104, Redondo Beach, CA, October 1991. IEEE Computer Society Press.

[SO91] Xiping Song and Leon J. Osterweil. Comparing design methodologies through process modeling. In *Proceedings of the First International Conference on the Software Process*, pages 29–44, Redondo Beach, CA, October 1991. IEEE Computer Society Press.

[SPSB91] Richard W. Selby, Adam A. Porter, Doug C. Schmidt, and James Berney. Metric-driven analysis and feedback systems for enabling empirically guided software development. In *Proceedings of the Thirteenth International Conference on Software Engineering*, Austin, TX, May 1991.

[Sut91a] Stanley M. Sutton, Jr. Accomodating manual activities in automated process programs. In *7th International Software Process Workshop — Preprints*, Yountville, CA, October 1991.

[Sut91b] Stanley M. Sutton, Jr. Software process environments: Some requirements and cponcepts. Arcadia Technical Report CU-91-01, Department of Computer Science, University of Colorado, Boulder, Colorado 80309, June 1991.

[Tam91] Doris Y. Tamanaha. Structured process flows (SPFs): A process model for metrics. In *1991 IEEE Aerospace Conference Digest*, Crested Butte, CO, February 1991.

[TBC+88] Richard N. Taylor, Frank C. Belz, Lori A. Clarke, Leon Osterweil, Richard W. Selby, Jack C. Wileden, Alexander L. Wolf, and Michal Young. Foundations for the Arcadia environment architecture. In *Proceedings of ACM SIGSOFT '88: Third Symposium on Software Development Environments*, pages 1–13, Boston, November 1988. Appeared as *SIGPLAN Notices 24*(2) and *Software Engineering Notes 13*(5).

[Tul88a] Colin J. Tully. Introduction. In Colin Tully, editor, *Proc. 4th International Software Process Workshop*, pages 3–4, Moretonhampstead, UK, May 1988. ACM Press.

[Tul88b] Colin J. Tully. Software process models and programs: Observations on their nature and context. In *Proc. 4th International Software Process Workshop*, pages 159–162, Moretonhampstead, UK, May 1988.

[Tul90a] Colin Tully. The implications of process support software for environment architectures. In *Proceedings of the 6th International Software Process Workshop*, pages 207–210, Hakodate, Japan, October 1990. IEEE Computer Society Press.

[Tul90b] Colin J. Tully. Software process issues in software engineering environments. In Fred Long, editor, *Software Engineering Environments*, volume 3. Ellis Horwood, Ltd., New York, 1990.

[War86] Paul T. Ward. The transformation schema: An extension of the data flow diagram to represent control and timing. *IEEE Transactions on Software Engineering*, SE-12(2):198–210, February 1986.

[War91] Brian Warboys. The practical application of process modelling — some early reflections. In A. Fuggetta, R. Conradi, and V. Ambriola, editors, *First European Workshop on Software Process Modeling*, Milan, Italy, May 1991. CEFRIEL, Politecnico di Milano, A.I.C.A. Working Group on Software Engineering.

[WD86] J. C. Wileden and M. Dowson, editors. *Proceedings of an International Workshop on the Sofwtare Process and Software Environments*, Coto de Caza, California, March 1986. Published as *Software Engineering Notes*, vol. 11, no. 4, 1986.

[WM85] P. Ward and S. Mellor. *Structured Development for Real-Time Systems.* Prentice-Hall, Inc., New York, 1985.

[WO90] Gen Watanabe and Leon J. Osterweil. Software process scheduling based on process programming. Arcadia Technical Report CU-90-02, Department of Information and Computer Science, University of California, Irvine, August 1990.

[WW89] David P. Wood and William G. Wood. Comparative evaluation of four specification methods for real-time systems. SEI Technical Report CMU/SEI-89-TR-36, Carnegie-Mellon University Software Enginnering Institute, Pittsburgh, Pennsylvania, May 1989.

[Zav89] Pamela Zave. Domain understanding and the software process. In Dwayne E. Perry, editor, *Proceedings of the 5th International Software Process Workshop*, pages 145–147, Kennebunkport, Maine, October 1989. IEEE Computer Society Press.

[ZO92] Hadar Ziv and Leon J. Osterweil. Evaluation and comparison of selected process-modeling formalisms. Arcadia Technical Report UCI-92-00, University of California, Irvine, April 1992.