Accounting for Purpose in Specifying Requirements for Process Programs

Stanley M. Sutton, Jr.

CMPSCI Technical Report 95–76
July 1995

Laboratory for Advanced Software Engineering Research
Computer Science Department
University of Massachusetts
Amherst, Massachusetts 01003

This work was supported by the Advanced Research Projects Agency under grant F30602-94-C-0137.

Abstract

Software process programs, as a group and individually, may be called on to serve multiple purposes, including process specification, implementation, simulation, proxy, and others. This variety of purposes is unusual in comparison to many familiar kinds of application, for which the purpose is typically assumed to be implementation. The variety of purposes and their influence on requirements makes the explicit treatment of alternative purposes important in process-program requirements specification.

One important influence that purpose has is in determining useful properties of the software-process system. These systems may be characterized potentially in terms such as consistency, fidelity, precision, verifiability, correspondence, coordination, determinability, and dependence. However, the properties that are actually important to a particular process program can vary significantly depending on the purpose of the program. These properties, in turn, induce specific functional and non-functional requirements on the process program. Conversely, when specific requirements conflict, the analysis of purpose provides a basis for conflict resolution.

Methods for process-program requirements specification can be based on the relationships among process-program purpose, process-system properties, and specific kinds of requirements. Such methods may have several potential benefits, including support for the identification, rationale, organization, and analysis of process-program requirements. Additionally, they may facilitate the prevention, detection, and resolution of conflicts among process-program requirements and purposes.

1 Introduction

The requirements for a program depend naturally on its purpose. On one level, programs serve a wide variety of purposes, related to their functionality and the situations in which they are used. On another level, however, most programs in fact serve one of a small number of general purposes. Specifically, the general purpose of most programs is to provide an implementation, typically for solving some problem or providing some service. The implementation-orientation of most software is reflected in typical in textbook treatments of requirements engineering, for example, "Once we have established *what* we want, we can proceed to the next level of modeling and define *how* to implement it" ([4], p. 83). Comparable statements are found in [17] (p. 132), [1] (p. 1), and [20] (p. 278).

In contrast, many different kinds of uses, reflecting different high-level purposes, have been suggested for software-process programs and related forms of process representation. Potential uses of process programs include:

- *Process specification*, in which the process program provides a formal definition of the process. These support *process understanding* in general. They can also support *static process analysis*, in which the program text is analyzed to determine properties of the process, and *process recognition* ([11]), in which the behavior of a development activity is evaluated for potential match against the defined process. Processes specified formally as programs may also provide the basis for process treaties and summits (described in [3]).

- *Process simulation*, in which the program is executed to provide a dynamic model of process behavior. Simulations can be used for *process prototyping* and for *dynamic process analysis*, that is, the analysis of patterns of behavior. These may be applied in *process discovery* ([6]) and in *process planning* for management purposes [12, 14].

- *Process implementation*, in which the program is executed in support of creating a software product. Such programs may differ widely in the degree and kind of support for process execution [9], ranging from passive *process guidance* to *process control* to *process automation*.

- *Process proxy*, in which the program is executed to support the process, but where the program is also intended to represent the process for purposes such as *process monitoring*, *process measurement*, *process analysis*, and *process discovery*. (Process proxy is discussed further in Section 4.4.)

This diversity of purposes applies not just across the domain of process programs as a whole. Differing combinations of purposes have been given for individual process programs or models:

- Static representation, static analysis, and comparison [19, 23]
- Static representation and simulation [18]
- Simulation and implementation [7]
- Simulation and dynamic analysis [12, 14]
- Guidance and recognition [11]
- Static representation and implementation [16, 22, 7]
- Manual and automated implementation [10, 5, 2]

This diversity of purposes, which may be manifest in a single process program, can have a profound influence on the requirements for a program. For example, many of the possible uses of process programs require that the program be executed, but some make use of the program text in a static form. Additionally, of the uses that depend on program execution, some support an implementation of the process, while others require additional sorts of functionality, either instead of or in addition to the functionality required for implementation. Furthermore, when multiple purposes are assigned to a single program, the number and complexity of requirements on the program increases, with a corresponding increase in the probability of conflict among them.

This paper examines the issue of purpose for process programs and analyzes its impact on process-program requirements specification. In particular, it shows how the purposes of a process program determine important properties of the software-process system, which in turn motivate particular kinds of functional and non-functional requirements on the program. This paper also shows how issues of purpose provide a conceptual framework for the development of methods for process-program requirements specification. Section 2 argues that the need for programs that serve multiple purposes is inherent in the domain of software processes; consequently, there will be an ongoing need to understand and accommodate the influences of multiple purposes. Section 3 presents a model that is used to illustrate characteristic properties of software-process systems; Section 4 then shows how the relevance and importance of these properties varies according to process-program purpose. Section 5 provides examples of how these properties, in turn, depend on or imply various functional and non-functional requirements on the process program. Additional factors affecting process-program requirements, the compatibility of different purposes, and the implications for a method for process-program requirements specification are discussed in Section 6. Finally, Section 7 presents a summary and conclusions.

2 Why Process Programs Serve Multiple Purposes

Process programs as a group have been addressed to a diversity of purposes since software processes became a domain for programming [16]. This diversity of purpose seems to be inherent in the domain, making the explicit recognition and accommodation of multiple purposes essential in specifying their requirements.

Support for process execution is arguably the ultimate goal of process programming (since execution may be taken as the ultimate goal for processes). However, even where "implementation" in some sense is the general goal of a process program, the particular sort of support for process execution can vary widely. For example, four different categories of support are identified in [9], including loosely coupled enactment, active process support, process enforcement, and process automation. These have very different ramifications for process-program requirements, and, moreover, they may be used in combinations and variations. Thus, for software processes, the concept of implementation is more diverse and perhaps less straightforward than it is for many domains.

At this (still early) stage in the development of software-process engineering, the use of process programs for process representations (definitions or specifications) may even be more important than their use for process implementation. This may be expected for the domain of software processes, where applications are large, complex, and variable, and where our understanding of them is limited. Process programs are useful for capturing software processes that are otherwise poorly understood or represented. Process programs afford the opportunity to simulate processes that are otherwise too cumbersome and costly to test directly. Additionally, programming can provide a common syntactic and semantic framework in which to analyze and compare processes that have been represented by various less formal means [19]. In such ways the process domain poses a number of different sorts of problems to which programming is applicable.

Characteristics of the domain of software processes also motivate the assignment of multiple purposes to individual process programs. For example, simulation is useful for analyzing, verifying, and predicting the behavior of software processes. However, given the complexity and variability of software processes, there is a significant concern as to how accurately any simulation of a process will match an actual execution. To help close the gap between simulation and actual execution, it may be helpful to have a single program support both process simulation and process execution. A related issue arises with respect to the testing of a software process and process program. A software process can be difficult to test in the abstract, i.e., without using it (at significant cost) to develop software. Consequently, the process program may be called upon both to help support the execution of the process and to support the testing of that process.

The problem of testing software processes is an example of a larger class of problems that arise from the difficulty of grappling with software processes generally. For a variety of reasons, software processes are difficult to monitor and measure directly: they are large, complex, distributed, long-lived, dynamic, and evolving. Process programs can be used to help with the monitoring and measurement of software processes to the extent that the process program can serve as a *proxy* for the software process. In other words, the execution of the process program can be studied as a surrogate for the execution of the software process that it supports. This requires that the process program provide not only an implementation of the software process but also a dynamic representation of it (that is, a view or depiction of the behavior of the process).

Finally, process programs may often be required to meet the differing needs of different categories of users. The typical "end users" of a process program will be software developers who are engaged in the processes of developing software. They will be concerned primarily with participating in the process, and it will be incumbent on the program to facilitate their participation. However, development activities are usually managed, and so process programs will also typically have to facilitate managerial oversight and control. Furthermore, process programs are also subject to meta-processes, not excluding monitoring, but also including dynamic maintenance and evolution, reuse, verification, and so on. Consequently, process programs must be developed so as to facilitate the performance of these meta-processes as well.

One domain that appears comparable to software process in the need for multi-purpose programs is that of manufacturing processes. Research issues for the use of information technology in manufacturing processes include the development of process descriptions that allow processes to be analyzed, enforced, and simulated, and the development of process-description languages that are both man- and machine-intelligible [15]. The motivations are typically the same as for software processes, for example, the need to support understanding and analysis, and the goal of supporting simulation of processes that are difficult or costly to prototype and test.

3 Software-Process Systems

This section presents a model of software-process systems including process programs. The model is used to illustrate recognized properties of processes and process programs. As shown in Section 4, various of these properties support various of the purposes for which process programs may be used. Conversely, the relevance and importance of particular properties for a particular program depends on the purposes of the program.

3.1 A Model of Software-Process Systems

A software process that is supported by a process program can be viewed as a kind of *process-in-process* system. In other words, the overall software process comprises numerous sorts of activities that contribute to the production of a software product, including both automated and manual activities. Among these activities is the execution of the process program (this is a particular kind of automated activity, although one that is generally interactive). The execution of the process program, as one activity (or set of activities) in the software process, guides or supports other activities and automates still others. However, the execution of the process program does not make up the whole of the execution of the software process; some process activities will lie beyond the scope or control of the process program, either because they are off-line, or because they are assumed by other programs (e.g., tools or cooperating process programs). The execution of the process program is thus a subset of the activities of the overall software process, albeit one that plays a distinguished role within the process.

We can further refine this view of a process-in-process system to differentiate static and dynamic aspects. A process program, like any program, has a static and dynamic aspect: the static aspect is the code viewed as a text, while the dynamic aspect is the behavior that results from the interpretation of the code. A similar distinction can be drawn analogously between the static and dynamic aspects of a software process. For example, static process representation by definitions or models is distinguished from the dynamic interpretation of those models [8, 13].[1]

A simple model of the software-process system where process programs are involved thus has two dimensions. It represents both the overall software process and the process program,

[1] Of course, a process program can also be a kind of process model or definition. However, a process program will not in general be the sole static representation of a software process. The process program, as process *code*, should be based on process-program specifications and designs. Those, in turn, will be derived at least in part from process specifications and definitions.

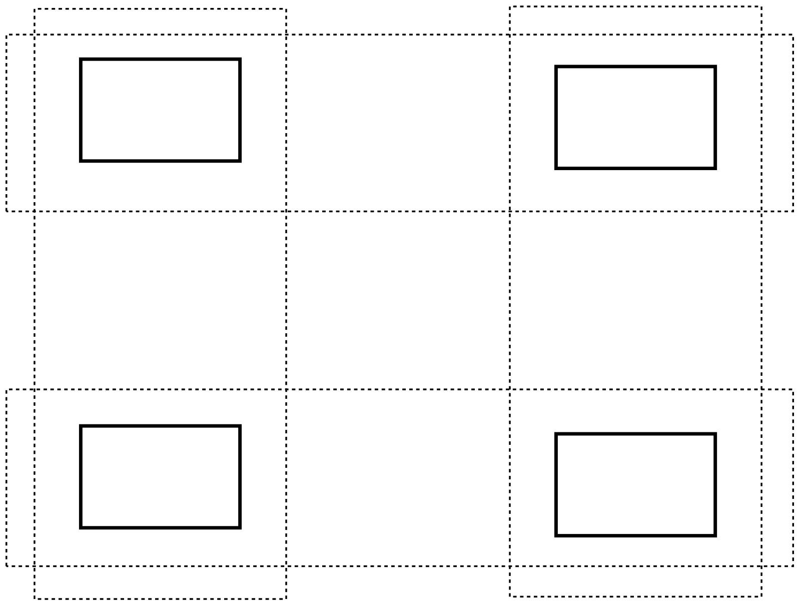

Figure 1: A Schematic Framework for Modeling the Software-Process System

and it differentiates the static and dynamic aspects of each. A schematic representation of such a model is shown in Figure 1.

3.2 Properties of Software-Process Systems

A number of properties of software-process systems have been identified and (sometimes variously) defined [8, 13, 21]. Among the properties that are especially relevant to the use of process programs are *consistency*, *precision*, *fidelity*, *verifiability*, *coordination*, *correspondence*, *dependence*, and *determinability*. These properties can be represented as relationships between the different parts of the model shown in Figure 1.[2] The meanings of these properties and some observations about them are presented below. The relevance of the properties to various purposes of process programs is discussed in the following section.

For purposes of this paper, the following definitions are adopted:

[2]This list of properties is intended to be representative rather than exhaustive. The fact that these particular properties can be modeled as relationships in Figure 1 makes the effect of program purpose especially easy to see. Certainly, though, the significance of other properties, such as efficiency, predictability, and fitness ([8]) is also related to program purpose.

- *Consistency* reflects agreement between the process program code and process specifications. It addresses the question of whether a process program is the correct one for the specified process.

- *Precision* is the degree to which the process specification details process activities that will yield accurate, i.e., intended, results [8]. It relates static specifications to dynamic behaviors, most typically at the level of the overall process.

- *Fidelity* is the faithfulness with which a defined process is followed [8]. It relates the actual behavior of an executing process to the process specification.

- *Verifiability* is the ability to measure the degree to which behavior conforms to a specification or definition. It is of primary concern here between the behavior and specification of the software process.[3]

- *Coordination* is the extent to which concurrent behaviors in the software process and process program are mutually appropriate, i.e., that the process is behaving correctly with respect to the execution of the program and vice versa [21]. For example, when a manager assigns process tasks to engineers, the process program (in coordination) may be expected to send out notices to the engineers of their assignments.

- *Correspondence* is the degree to which individual steps and patterns of behavior in the software process match steps and patterns in the behavior of the process program [21]. For example, a *Modify_Code* step in the process may be matched by a corresponding MODIFY_CODE procedure in the process program. A related concept proposed here is *prospective correspondence*. This relates the behavior of a software process to a description in a process program; it implies that the behavior of the process would correspond to the behavior of the program, if the program were executed.

- *Dependence* characterizes the reliance of process execution on process program execution [21]. A process with a high dependence on its process program will not be able to execute significantly unless the program is able to execute.

- *Determinability* [21] reflects the extent to which the behavior of the process is directed and constrained by the behavior of the process program. For example, a process program that only weakly determines the software process cannot be counted on to support the fidelity of the process.

[3]In [13] verification is defined as directed towards the formal proof of properties of a process model. In this sense it can be considered as contributing to *consistency* as defined above, in that proofs of process-program properties can be used to assess conformance to process specifications. Our concern, though, is with the additional problem of verifying process behavior.

These properties are illustrated in Figure 2. Precision, fidelity, and verifiability are properties that relate static and dynamic elements of the model, typically at the level of the software process. However, if the process program is viewed as a process specification, then these properties may relate process behavior to process-program code (as illustrated by the dashed diagonal arcs from lower right to upper left in Figure 2). Verifiability and fidelity can be defined between an executing process program and a software process specification. (Precision is not an issue in this case because the process program, apart from the software process, does not yield a product.) Fidelity and verifiability are related but distinct; fidelity concerns the actual behavior of a process, whereas verifiability concerns the ability to measure or assess that behavior.

Determinability, dependence, correspondence, and coordination all relate the behavior of the process and the behavior of the process program. Dependence does not necessarily imply determinability, since a process may depend on a process program for purposes other than control (e.g., for access to tools or data, or for authentication services). Determinability does typically imply some critical dependence, however (by which the determined process can be imposed). Coordination does not imply correspondence (since coordinated processes may perform different activities), but neither does correspondence imply coordination (since the process and process program may be executed asynchronously, as when the program serves as a process simulation). Prospective correspondence is a generalization of the notion of correspondence that relates actual program behavior to the implied behavior of an interpreted program text.

4 Dependence of System Properties on Program Purpose

This section assesses the relevance of different properties to various purposes that a process program may serve. Four cases are considered: process simulation, process definition (but not process implementation), process implementation (but not definition), and process proxy.

4.1 Process Simulation

If the process program is to be used to simulate the software process, then it should more or less behave like the process, especially in those aspects that are to be simulated. For this purpose, consistency and correspondence are most important. Consistency (of process program and process specification) implies that the process simulated is the correct one. Correspondence (of process program and process behavior) implies that the behavior of

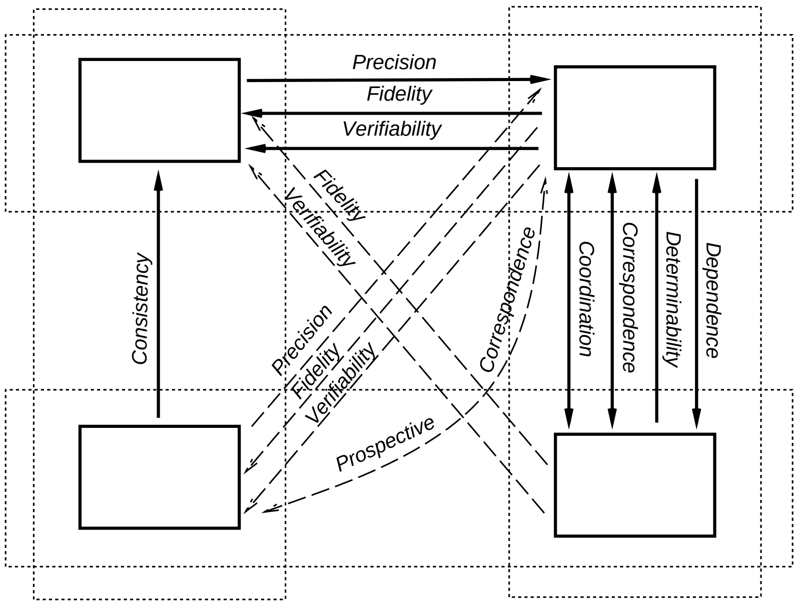

Figure 2: Properties of the Software-Process System

the simulation should reflect the behavior of the process. Verifiability may be important, especially if consistency is not established, to help assure that the simulated process is the process specified. (In this case, fidelity of the simulation to the process specification is the goal.)

Coordination, determinability and dependence are less important. These characterize the relationship between an executing process and a process program, so they are not relevant in a simulation, where the process itself is not executed. Similarly, precision is largely independent in a simulation. That is because the simulation will not yield a product, and the purpose of the simulation may be to evaluate properties that affect the expected precision of the process.

4.2 Process Definition

When a process program is used as a process definition, without being executed to support the process, the properties of consistency, (prospective) correspondence, and precision are most important. Consistency, as usual, is important to assure that the programmed process is the correct one. Prospective correspondence is important because it implies that the behavior of the program, if executed, would conform to the behavior of the process; in turn that implies that the program, which encodes the behavior, is also a good description of the process. Precision is important because the process described should be one that produces an accurate result.

The other properties are largely unimportant for a process program used just for purposes of definition, primarily because the program in that role is not executed. For example, the execution of the process cannot depend on the execution of the program, the behavior of the program cannot determine the behavior of the process, and process and program behavior cannot actually correspond. Fidelity and verifiability may remain a concern for the process, but they must be supported by means other than execution of the process program (e.g., by independent analyses and process monitors).

4.3 Process Implementation

When a process program is used for implementation, but not definition, then the most important properties are consistency, fidelity, and coordination. In various ways these support or reflect correct behavior on the part of the process that the program is to implement.

Precision in the strict sense is not a relevant property of the process program since the program in this role does not specify the process (although the program should still support the execution of a precise process). Correspondence of process and process-program behavior is also not important since the purpose of the process program requires only that the program support the specified process, not that it behave like the specified process (although this may be an effective implementation strategy).

Determinability, verifiability, and dependence may or may not be important, depending on other considerations. These are all properties that can be facilitated through an implementing process program. However, whether they are required of the program depends on concerns such as whether the process should be executed strictly or loosely, whether there is a need to certify process behavior, and whether the process should be robust from failures of the process program.

4.4 Process Proxy

A program that serves as a process proxy represents a specialized kind of process implementation, one in which the behavior of the process program is to be viewed as a surrogate for the behavior of the process. The motivation for this use of process programs stems from the difficulty of capturing process behavior directly [24]. However, if the behavior of the process program is representative of the behavior of the process, then the behavior of the process can be studied indirectly by studying the behavior of the process program. The advantage of this is that it should be possible to substantially automate monitoring, visualization, and measurement of the process program, thus providing significant leverage in the study of the software process.

The important properties for process proxy are coordination, correspondence, determinability, and dependence. Coordination and correspondence help to establish a meaningful mapping between program behavior and process behavior (e.g., order, number, and length of process steps as they are executed). Determinability and dependence help to assure that the actual process behavior will not deviate from program behavior. (The goals for process proxy may sound similar to those for process definition. Certainly, the behavior of a process should be constrained, if not determined, by its definition. However, a process definition provides a static representation that establishes, within limits, what process behavior should be. In contrast, a process proxy is intended to provide a dynamic representation that reflects what process behavior actually is.)

Verifiability, precision, fidelity, and consistency are not so important for process proxy, since the goal of a process proxy is defined with respect to actual process behavior, independent of whether or how that behavior conforms to or is otherwise related to any process specification. However, the requirement that a program serve as a proxy may be motivated by a need to support process verification or to measure process fidelity. (Process simulation may also be addressed to concerns such as verification and fidelity. However, with simulation, the goal is to be able to predict the behaviors of a prospective process rather than to measure the behaviors of an actual process.)

5 Supporting System Requirements through Program Requirements

The previous section shows how the purpose of a process program can determine the importance of particular properties in a software-process system. The need to support these properties can be viewed as high-level requirements on the process program, specifically, requirements regarding the relationship of the process program to other elements in the overall software-process system illustrated in Figure 1. These properties are still relatively abstract, though, with respect to capabilities or features of the process program. The need to support various properties of a process system thus implies a variety of more concrete requirements on a process program.

The relationship between desired high-level properties and specific functional and non-functional requirements on a process program is not one-to-one, however. As indicated below, a given high-level property such as fidelity or determinability can be supported through a variety of features or capabilities in a specific program. Conversely, a particular feature or capability can be applied in support of more than one high-level property. Some representative examples are given below.

- *Process control and guidance*: Requirements for process control and guidance may be imposed in support of process determinability, fidelity, or coordination.
- *Process-program understandability*: The understandability of the process program by humans is important for evaluating its consistency with respect to process specifications. It is important in support of process fidelity, if the faithfulness of process execution is to be evaluated against the program interpreted as a process definition. It can also contribute to the fitness of process executions, where the program is manually interpreted in support of process execution.
- *Process monitoring and visualization*: Requirements regarding process monitoring and visualization may be imposed to support process verification (by allowing process behaviors to be tracked against specifications), process fidelity (by enabling process participants to see where they are in the process and where they should be going), and process coordination (by helping process participants to orchestrate off-line and on-line process activities).
- *Process-program analyzability*: Requirements that a process program be automatically analyzable with respect to certain properties may be imposed to help assure consistency of process programs with respect to process specifications, or to help assess the correspondence (or prospective correspondence) of process and process-program behavior (e.g., orders or partial orders of process steps). Analysis of a process program can also help in identifying program properties that affect the process system. For example, analysis may help to determine whether each process step is accompanied by a corresponding logging operation as required in support of process verification.
- *Process fitness*: Fitness is the ease with which a process specification actually can be followed [8]. Fitness in general depends on a number of factors such as the understandability of the process specification, sensibility and correctness of the process, and factors that affect the implementability of the specification. Requirements for fitness may be imposed, for example, to support process fidelity (making faithful execution

easier) or to facilitate process coordination (e.g., enabling off-line activities to better follow the model with which on-line activities are to be orchestrated).

This list of process-program requirements is hardly exhaustive; additional kinds of requirements that might be considered include automation, reliability, security, and performance, among others. However, the above examples indicate that many kinds of process-program requirements may be imposed specifically to support desired properties in the software-process system. These requirements can be related, albeit indirectly, back to the original purposes of the process program. To give just three examples:

- Analyzability of the process program, which can be used to establish process correspondence, contributes to the quality of process program that are used as simulations.
- Automation, which helps to support process dependence and determinability, contributes to process implementations.
- Monitoring and visualization, which facilitate process fidelity and coordination, contribute to the use of a program as a process proxy.

It is perhaps unfortunate but to be expected that the relationship between process-system properties and categories of process-program requirements is not entirely systematic. Consequently, neither is the indirect relationship between process-program purpose and specific process-program requirements. However, the fact that these relationships are not entirely systematic does not imply that there is no connection between properties and requirements or purposes and requirements. On the contrary, there are fundamental semantic dependencies between specific program requirements and particular system properties, and through system properties, back to program purposes. These semantic dependencies can be identified, elaborated, motivated, organized, justified, rationalized, and evolved in the context of the conceptual framework provided by the analysis of program purposes and desired system properties. This prospect is discussed further in Section 6.3.

6 Discussion

This section addresses several topics that help to put process-program requirements specification in context, including other sources of requirements, the compatibility of different purposes, and prospects of a specification method.

6.1 Other Sources of Requirements

This paper presents the purpose of a process program as being a primary motivation for particular kinds of requirements on the program. The categories of purpose considered are very general: implementation, specification, simulation, and so on. The properties of software-process systems that support these general purposes should thus be applicable to large classes of process programs, for which the specific requirements that support the properties will also be relevant.

However, not all of the kinds requirements that are relevant to a process program are derivable from the general purposes of the program. For example, the development of classified applications may require a secure process. If the process is implemented by a process program, the need for security may impose a requirement for dependency, e.g., that the process depend on the process program for access control and monitoring. Dependence is not necessarily associated with process programs that serve as implementations; it is imposed in this case as a consequence of an independent requirement for security. Examples such as this suggest that process-program requirements will derive not only from the general purposes of the program but also from sources such as the particular characteristics of a project, product, organization, or other contingent circumstances.

It may be argued that a more refined view of the purposes of a process program could account for many of these more specific considerations: the purpose may be to provide a secure implementation of a software process. On the other hand, in many cases, organizational or other context-dependent issues will be determined before any particular process program is developed, such that the question of purpose is raised only within this context, which may be implicitly assumed for process-program development. In any case, whether as refinements on general purposes, or as independent context, the specific circumstances of a process program must nevertheless be accounted for.

6.2 Compatibility of Purposes

That a process program may be called on to serve multiple purposes raises the question of the compatibility of those purposes. The compatibility of purposes to which process programs may be put rests ultimately on the specific requirements that those purpose impose on a program. The compatibility of purposes must thus be analyzed in terms of the compatibility of these requirements.

Two purposes that may impose very different requirements on a process program are those of defining a software process and of implementing a software process. To define a software process, the control and data elements of the process should be represented directly and clearly, including their static relationships and potential dynamic behaviors. The level of abstraction should be generally high, and extraneous details should be kept to a minimum. To implement a software process it is first of all essential to provide the required functionality for the process. The form of the process program may or may not clearly represent the form of the process; conditions related to the implementation substrate or issues related to runtime behavior may have a more important effect on the program. For example, a single step in the process may be represented in the program by a group of implementation-oriented operations related to beginning transactions, acquiring access to objects, sending notifications to monitoring agents, and so on. Additionally, an implementation must address many practical details that are best abstracted from a process definition, and these must penetrate to levels of abstraction that are typically below the levels of process semantics. Consequently, a process program that emphasizes process definition is likely to be less than ideally matched to many implementation settings, while a process program that addresses implementation is likely to provide information that is inappropriate or irrelevant for process definition.

Other combinations of purposes face potential incompatibilities as well. For example, programs for process implementation, proxy, and simulation must all provide executable functionality, often in related areas. However, the use of a program as a proxy for the process requires that the behavior of the program be measurable in terms of the process definition. If the implementation of the program is not organized according to the definition, then it may be relatively difficult to support measurement in the required terms. Additionally, the specific functionality of an implementation and a simulation will differ These differences may be cumbersome to accommodate in program development even if they are not directly incompatible.

Not all purposes are entirely incompatible. Programs for process simulation and process proxy may be compatible with respect to their views of the process, i.e., a program acting as a proxy for the process may provide a basis for measurement of the same properties that are represented by a program acting as a simulation. Both proxy and simulation may be substantially compatible with the use of the program as a process definition, since it is typically important to simulate or measure the defined properties of a process. However, there are still potential conflicts in these cases, since the representation of functionality to support measurement or simulation may compromise the clarity of the program as a definition. Even the combination of functionality for measurement and for simulation may complicate the development of an executable process program.

6.3 Prospects for a Specification Method

The systematic treatment of purpose provides a principle around which methods for process-program requirements specification can be organized. Purpose is generally important in the specification of application requirements, even if it may be treated somewhat implicitly. The particular variety of purposes relevant to process programs, both as a group and individually, implies that purpose should be treated explicitly. As purpose motivates many requirements, it can serve as a starting point for requirements specification methods. Finally, the semi-systematic relationship of process-program purpose to diverse process-program requirements provides that it may serve as basis for organizing the specification and analysis of those requirements.

A method for specification of process-program requirements can be organized and elaborated according to the conceptual model shown in Figure 3. This models begins with the identification of purposes for the program. This should be based on the organizational context, the project or projects for which the program is intended, and the larger process-development life-cycle of which process-program development is a part. The determination of purposes that apply to a program also implies the determination of purposes that do not apply. The identification of applicable and inapplicable purposes allows the relevant properties of the process system to be delineated. Specific program requirements can then be imposed as needed to help support the desired properties. The ability to dismiss irrelevant properties reduces the number of concerns that may constrain program requirements and thus affords a corresponding degree of freedom in process-program development. This freedom is important because the ability to fulfill particular requirements on the program affects the ability to achieve desired properties in the system; in turn, that affects the ability of the program to fulfill its given purposes.

This conceptual model offers several potential methodological benefits. It motivates the identification and rationale of important aspects of process programs and of consequent specific requirements. Conversely, it allows efforts to be directed away from properties and requirements that are unimportant to program purpose (thus helping to maximize the degrees of freedom in process-program design and implementation). It also helps with the analysis and resolution of conflicts among requirements by providing a framework in which the sources of the requirements can be identified, the motivation for the requirements understood, and the priority of the requirements evaluated according to the priority of the purposes they serve. Finally, it offers the prospect of early detection of conflicts and also opportunities among potentially incompatible or synergistic types of requirements.

Figure 3 also indicates the influence of independent concerns that arise from organizations, projects, and processes. These may cause additional restrictions (or allow additional op-

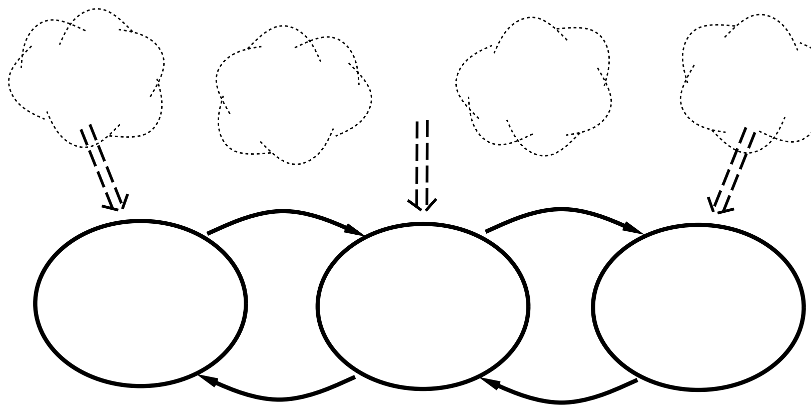

Figure 3: Conceptual Foundation for a Method of Process-Program Requirements Specification

portunities) at any stage of the method, and process-program requirements must ultimately reflect not only the central purpose of the program but also these additional qualifications.

7 Summary

Software process programs may be called on to serve multiple high-level purposes, including process specification, implementation, simulation, and proxy, among others. Multiple purposes apply both across the domain of process programs and also to individual process programs. This variety of purposes is unusual, although it may be characteristic of other process domains, and it extends beyond the usual implementation-oriented perspective assumed in typical treatments of requirements specification. In the specification of process-program requirements, the explicit treatment of alternative purposes is crucial because of this variety and its effect on program characteristics.

Process programs are used in a system that can be characterized in terms of properties that relate the static and dynamic aspects of process programs and the overall software processes they support. Potentially important properties that characterize process systems include consistency, fidelity, precision, verifiability, correspondence, coordination, determinability, and dependence. The properties that are actually important to a particular process program, however, vary significantly depending on the purpose of the program. Thus, in order to fulfill the intended purposes of a process program, the relevant properties must be obtained (while irrelevant properties need not constrain development). These properties, in turn, motivate the specification of particular requirements on the process program. Program purpose can thus be used to organize and justify specific process-program requirements.

The relationships between process-program purpose and process-system properties, and between those properties and specific kinds of process-program requirements, provides the basis for methods of process-program requirements specification. Those methods have several potential benefits, including support for the identification, rationale, organization, and analysis of process-program requirements and well as for the prevention, detection, and resolution of conflicts among process-program requirements and purposes. Such methods must be developed as part of larger meta-processes for software-process development; software-process development provides a context for process-program development including, in particular, the basis for determining process-program purpose.

Acknowledgements

This work was supported by the Advanced Research Projects Agency under Grant F30602-94-C-0137. Peri Tarr graciously reviewed a draft of this paper and offered many constructive comments.

REFERENCES

[1] Russell J. Abbott. *An Integrated Approach to Software Development.* John Wiley & Sons, New York, 1986.

[2] V. Ambriola, P. Ciancarini, and Montangero. Software process enactment in oikos. In *Proc. of the Fourth ACM SIGSOFT Symposium on Practical Software Development Environments*, pages 183–192, 1990. Irvine, California.

[3] Israel Z. Ben-Shaul and Gail Kaiser. A paradigm for decentralized process modeling and its realization in the oz environment. In *Proc. of the 16th International Conference on Software Engineering*, pages 179–188, 1994.

[4] Bruce I. Blum. *Software Engineering–A Holistic View.* Oxford University Press, New York, 1992.

[5] R. F. Bruynooghe, J. M. Parker, and J. S. Rowles. PSS: A system for process enactment. In *Proc. of the First International Conference on the Software Process*, pages 128 – 141, 1991. Redondo Beach, California, October, 1991.

[6] Jonathan E. Cook and Alexander L. Wolf. Towards metrics for process validation. In *Proc. of the Third International Conference on the Software Process*, pages 33–44, 1994.

[7] Wolfgang Deiters and Volker Gruhn. Managing software processes in the environment melmac. In *Proc. of the Fourth ACM SIGSOFT Symposium on Practical Software Development Environments*, pages 193–205, 1990. Irvine, California.

[8] P. H. Feiler and W. S. Humphrey. Software process development and enactment concepts and definitions. In *Proc. of the Second International Conference on the Software Process*, pages 28 – 40, 1993.

[9] Christer Fernström. PROCESS WEAVER: Adding process support to UNIX. In *Proc. of the Second International Conference on the Software Process*, pages 12 – 26, 1993.

[10] Christer Fernström and Lennart Ohlsson. Integration needs in process enacted environments. In *Proc. of the First International Conference on the Software Process*, pages 142 – 158, 1991. Redondo Beach, California, October, 1991.

[11] Karen E. Huff and Victor Lesser. A plan-based intelligent assistent that supports the software development process. In *ACM Symposium on Practical Software Development Environments*, pages 97 – 106, 1988.

[12] Marc I. Kellner. Software process modeling support for management planning and control. In *Proc. of the First International Conference on the Software Process*, pages 8 – 28, 1991. Redondo Beach, California, October, 1991.

[13] Jacques Lonchamp. A structured conceptual and terminological framework for software process engineering. In *Proc. of the Second International Conference on the Software Process*, pages 41–53, 1993.

[14] Peiwei Mi and Walt Scacchi. Modeling articulation work in software engineering processes. In *Proc. of the First International Conference on the Software Process*, pages 188 – 201, 1991. Redondo Beach, California, October, 1991.

[15] National Research Council. *Information Technology for Manufacturing–A Research Agenda.* National Academy Press, Washington, D. C., 1995.

[16] Leon J. Osterweil. Software processes are software, too. In *Proc. Ninth International Conference on Software Engineering*, 1987. Monterey, CA, March 30 – April 2, 1987.

[17] Roger S. Pressman. *Software Engineering–A Practitioner's Approach.* McGraw-Hill, Inc., third edition, 1992.

[18] Motoshi Saeki, Tsuyoshi Kaneko, and Masaki Sakamoto. A method for software process modeling and description using LOTOS. In *Proc. of the First International Conference on the Software Process*, pages 90 – 104, 1991. Redondo Beach, California, October, 1991.

[19] Xiping Song and Leon J. Osterweil. Experience with an approach to comparing software design methodologies. *IEEE Trans. on Software Engineering*, 20(5):145–165, May 1994.

[20] Donald V. Steward. *Software Engineering with Analysis and Design.* Brooks/Cole Publishing Company, Monterey, California, 1987.

[21] Stanley M. Sutton, Jr. Opportunities, limitations, and tradeoffs in process programming. In *Proc. of the Second International Conference on the Software Process*, pages 135–146, 1993.

[22] Stanley M. Sutton, Jr., Dennis Heimbigner, and Leon J. Osterweil. APPL/A: A language for software-process programming. *ACM Trans. on Software Engineering and Methodology*, 4(3), July 1995. to appear.

[23] Masato Suzuki and Takuya Katayama. Meta-operations in the process model HFSP for the dynamics and flexibility of software processes. In *Proc. of the First International Conference on the Software Process*, pages 202 – 217, 1991. Redondo Beach, California, October, 1991.

[24] Alex L. Wolf and David S. Rosenblum. A study in software process data capture and analysis. In *Proc. of the Second International Conference on the Software Process*, pages 115–124, 1983.