POSITION PAPER
for the 4th International Software Process Workshop
Representing and Enacting the Software Process

Problems of Scale and Process Models

Dewayne E. Perry
AT&T Bell Laboratories, Murray Hill

Introduction

While software projects come in a wide variety of sizes and kinds, there are sufficient similarities among them so that we may abstract the general from the particular characteristics. These general characteristics then serve as basic issues in the creation of software processes. Some of these issues are moderately well-understood — such as various techniques of refining designs into implementation — while others are not. Scale is one of those important issues that have not been considered sufficiently and it is the issue of scale that I address in this paper.

First, I present a sociological metaphor, which I found useful in characterizing software development environments, and apply it to software process models. Next, I discuss the problems inherent in managing generic process models and their evolution in the context of this metaphor. Finally, I consider the requirements that the problems of scale and generic process models place on the supporting environment.

A Metaphor for Problems of Scale

In our paper, ‘‘Models of Software Development Environments (SDEs)’’ [1], Gail Kaiser and I present a sociological metaphor that is useful in considering the effects of scale in software projects. The metaphor suggests four kinds of social units: the individual, the family, the city, and the state. These social units are particularly suggestive of the kinds of interaction that arise within different kinds of software projects:

- the individual model characterizes the developer working in isolation;
- the family model characterizes developers working in a small group where each person works with a great deal of freedom and requires only a small degree of coordination to function well within that group;
- the city model characterizes a large group of developers where more formal and explicit rules are needed to ensure the sustained cooperation among individual members of the group (primarily because the freedom allowed the smaller unit leads to unacceptable anarchy); and finally
- the state model characterizes a collection of social units suggesting, perhaps, either a very large, dispersed project with widely different subprojects, or a company with a wide variety of projects.

While we apply this metaphor to characterize SDEs, it is equally applicable to characterizing software processes. In fact, it is doubly suggestive. First, as in the characterizations of SDEs, the metaphor emphasizes the differences inherent in considering problems of scale and its affect on process models. Second, it provides a useful separation of concerns and suggests a hierarchy of process models:

- the individual process model describes the developer working in isolation — for example, doing detailed design, coding, debugging, testing, etc.;
- the family process model describes how small collections of developers (individual processes) interact and coordinate their efforts in small projects or sub-projects – for example, doing requirements definition, system architecture, design, integration and testing, etc.;

- the city process model describes how collections of families (family processes) interact and cooperate in large projects — for example, project management issues, system integration and testing, etc.; and finally
- the state process model describes how collections of projects (city processes) are administered and coordinated.

The individual and family processes are relatively well-understood. While we do build large systems, our understanding of that 'city' process is still very shallow and the process used is an ad hoc one, somewhat like 'flying by the seat of our pants'. Of the state process, we can identify some of the goals, but have only a vague idea what it is. In the next section, I suggest some of the components that should be found in a state process model.

Process Models and their Evolution

One can distinguish, logically at least, between the providers of software process models and users of them. The actions of providers and users are distinct as well: providers create and evolve generic process models; users choose among several generic process models and instantiate or customize them for their particular needs. I argue that this separation should be a real one (with one exception) and that the state level is the appropriate provider (and, not coincidentally, a user as well) of process models.

Commonality is one of characteristics most desired at what I call the state level of SDEs. Analogously, it is in the context of the state process that we consider how commonality is to be achieved among its component parts. The primary reason for this emphasis on commonality is that of productivity: people can be moved from one place to another without loss of productivity; products can be shared and moved from one part to another, thereby reducing effort and cost. For this reason, the state process model should be the locus of the lower level processes' creation and evolution as well as its own.

The user operations of choice, instantiation and customization impose at least two constraints on the state process model. First, the various levels of process models should provide a set of different process models to chose from. There is a need for this that is different from that of customization: just as some paradigms and programming languages are more appropriate for particular problems, so some process models are going to be apt for some projects and not others; there is a qualitative difference, not merely a difference in degree that can be parameterized. Second, process models should be generic so that they can be instantiated within a reasonable level of customization for a particular project. This instantiation should be more at the level of Ada generics than at Emacs' open-ended customizability because of evolutionary considerations discuss below.

Just as evolution of software products is the dominant part of their life-cycle, so evolution will be a dominant part of the software process model life-cycle. This fact imposes a number of constraints and suggests a number of directions for the process model provider. In addition to the obvious need for formal representation of process models, there is the requirement for maintaining knowledge of the various choices and instantiations for each level of the process hierarchy in each process (for each project) managed by the state process. As various process models evolve, those instantiated and customized processes must be updated where they are being used. Successful propagation of changes can only be accomplished with extensive knowledge of where the changes must be made and how they interact with the various instantiations and customizations. This propagation process is complicated enormously if customizability is open-ended, but should be tractable if there are well-defined limits to customization.

Environmental Support

Process model support environments must have tools that are similar to those used in the modeled software process: tools for formulating process models, for managing their instantiation and customization, for managing their evolution, and for keeping track of various model versions and model configurations. While certain approaches are suggestive about ways to formulate process models, such as process programming [2], we know very little about how to support them. In the context of our model of SDEs (which consists of policies, mechanisms and structures), we have some good ideas about the *policies* that we want to have about process models, some general notion of the kinds of *mechanisms* to support process

formulation, but almost no understanding of the *structures* needed to support these policies or mechanisms. Instantiation and customization is, again, uncharted territory with some ideas as to policies, even vaguer ideas of mechanisms, and no notion of structures. For evolution management and version/configuration management, there is better support: Arcadia's object manager [3] provides a useful structure for basic objects and Inscape's semantic interconnection structure [4] provides a good basis for environmental-supported evolution and version/configuration management [5] both in terms of mechanisms and structures. On the whole, existing environments are at best suggestive of the environmental support needed for process models.

Further, there is an intimate relationship between the modeling process and the SDEs used in the actual process: first, changes in the SDE may affect the corresponding process model, for example by inducing changes in the model; second, the process model may dictate the composition of the SDE or direct its evolution. In both cases, the support environment must provide means for correlating the two entities and maintaining their consistency. Inscape's approach to describing the semantics of interfaces is suggestive for maintaining the consistency relationship in one direction — that of keeping the process model consistent with the SDE. By describing the SDE tool interfaces in semantic terms (as, for example, is done in Marvel [6]) and using the change propagation mechanisms of Inscape and Infuse [7], the environment can keep the process model consistent with the changes in the modeled SDEs. The other direction, changing the process model and reflecting those changes in the modeled SDE, devolves to the case of the process model providing the requirements for the SDE and that part of the software process currently remains informal.

References

[1] Dewayne E. Perry and Gail E. Kaiser. *Models of Software Development Environments*. Computing Systems Research Laboratory Technical Report, AT&T Bell Laboratories, August 1987.

[2] Leon Osterweil. ''Software Processes are Software Too'', *Proceedings of the 9th International Conference on Software Engineering*, Monterey, CA, March 1987. pp 2-13.

[3] Richard H. Taylor, *et al.*. *Next Generation Software Environments: Principles, Problems, and Research Directions*. COINS Technical Report 87-63, July 1987, University of Massachusetts at Amherst.

[4] Dewayne E. Perry. ''Software Interconnection Models'', *Proceedings of the 9th International Conference on Software Engineering*, Monterey, CA, March 1987. pp 61-69.

[5] Dewayne E. Perry. ''Version Control in the Inscape Environment'', *Proceedings of the 9th International Conference on Software Engineering*, Monterey, CA, March 1987. pp 142-149.

[6] Gail E. Kaiser and Peter H. Feiler. ''An Architecture for Intelligent Assistance in Software Development'' *Proceedings of the 9th International Conference on Software Engineering*, Monterey, CA, March 1987. pp 80-88.

[7] Dewayne E. Perry and Gail E. Kaiser. ''Infuse: A Tool for Automatically managing and Coordinating Source Changes in Large Systems'', *Fifteenth Annual Computer Science Conference*, St. Louis, MO, February 1987. pp 292-299.