itivation is that form of abstraction that requires elaboration before execution: the actual implementation is left to the process executor and is bounded either by a formal grammar, a system of constraints, or by the existence of the available building blocks. This kind of abstraction enables one to provide guidance without prejudging the solution (important for generic processes) or straight-jacketing the process executer.

Stratification is that form of abstraction by which we layer a system. This means of abstraction works very well in conjunction with the previous form: stratification enables one to supply project specific solutions to the fragments which have their assumptions and goals specified, but not their implementations. Specifically, I recommend abstracting the methods and tools layers from the generic descriptions (which is consistent with the separation of project and environment structures from process structures discussed above).

3 Requirements for Best in Class Processes

Having detailed a number of ways in which one can achieve generic processes and thus gain a measure of reuse across multiple projects, there still are cases where that approach will not work. This is partly for technical reasons and partly for sociological ones. One of the technical reasons is the differences in the types of systems being produced: large—scale, embedded, fault—tolerant, real—time systems may well require an entirely different set of development processes than typical business information systems.

A sociological reason is that even if they do not really require different processes, they certainly *feel* that their processes are different. There is also bound to be a certain amount of reluctance to move towards common processes when there is this perception of difference. This was certainly my experience with various groups company wide.

An alternative approach is the one that we took in the process team to address the problems of company—wide processes [9]. Instead of common processes, we defined the requirements for best in class software development processes. The intent of this approach is to set the standards for the various parts of the company to aim for. Along with the requirements we provided examples of processes that have met these requirements and that were in use in some part of the company.

The advantage of this approach is that it allows autonomy while setting the goals for the individual projects processes. The pitfall to be avoided is defining the requirements at such a high level that virtually any development process can satisfy them. Conformance (or non-conformance) then becomes a matter for the individual projects.

References

[1] David C. Carr, Ashok Dandekar, and Dewayne E. Perry. "Experiments in Process Interface Descriptions, Visualizations and Analysis", *Software Process Technology — 4th European Workshop, EWSPT'95* , Wilhelm Schaefer, ed. Lecture Notes in Computer Science, 913, Springer-Verlag, 1995. pp 119 — 137.

[2] Ashok Dandekar and Dewayne E. Perry. "Experience Report: Barriers to an Effective Process Architecture", submitted for publication.

[3] Ashok Dandekar, Dewayne E. Perry and Lawrence G. Votta. "A Study in Process Simplification", submitted for publication.

[4] Dewayne E. Perry. Policy and Product-Directed Process Instantiation. *Proceedings of the 6th International Software Process Workshop*, 28-31 October 1990, Hakodate, Japan.

[5] Dewayne E. Perry. "Policy-Directed Coordination and Cooperation", *7th International Software Process Workshop* , Yountville CA, October 1991.

[6] Dewayne E. Perry. "Humans in the Process: Architectural Implications", *Proceedings of the 8th International Software Process Workshop*, March 1993, Schloss Dagstuhl, Germany.

[7] Dewayne E. Perry. "Enactment Control in Interact/Intermediate" *Software Process Technology — 3rd European Workshop, EWSPT'94* , Brian Warboys, ed. Lecture Notes in Computer Science, 772, Springer-Verlag, 1994. pp 114 — 118.

[8] Dewayne E. Perry. "Issues in Process Architecture", *Proceedings of the 9th International Software Process Workshop: The Role of Humans in the Process*, October 1994, Airlie VA. IEEE Computer Society Press. pp 138—140.

[9] SLG Process Subteam. "SLGProcess Subteam Best-In-Class Software Process Requirements; Release 2", AT&T, December 1995.

and process instantiation time to form a complete process system for a particular project and its environment.

2.1.2 Separation of Descriptions and Policies

I have argued that elsewhere ([4], [5]) that policies are one of the fundamental components of process descriptions and that combined with the product state become one of the primary drivers in the dynamic and concurrent aspects of software processes. To achieve the flexibility necessary for a process system to serve effectively the development of software systems, one must be able to vary the policies governing activity assumptions and results, cooperation and coordination among people, and the important product states.

Analogous to abstracting specific tools into generic technological activities is the abstracting of specific policies into a generic policy vocabulary. Not only then does this enable the process administrators to vary the policies over the cycle of product development, it enables us to abstract these policies so that they may be instantiated as appropriate for a specific projects and thus vary appropriately across projects with minimum effect on the process descriptions themselves.

2.1.3 Separation of Interfaces and Implementations

For the same reasons that we separate interfaces from their implementations in the components that are used to build the software products, we separate process interfaces from their implementations. As in software modules, this form of encapsulation enables us to vary the implementation while maintaining a uniform interface. In terms of generic processes, we can maintain an uniform interface across various uses, but vary the implementation. Moreover, it enables us to do a variety of analyses and visualizations [1].

Reuse of the interfaces may extend in either of two directions. First, it may be that the implementation is varied from instantiation to instantiation. Second, we may reuse just the interface and delay the binding to the actual implementation and use the interface as an effective means of controlling stratification (discussed below).

2.1.4 Separation of Fragment Concerns

A different kind of separation is that along functional lines and akin to the problem of module and subsystem organization in software products. It is the appropriate packaging of process fragments into domain related process components. For example, two typical cases are reviews and effort estimates. It is often the case that both design reviews and design effort estimates are packaged as part of the design process, partly because it is what designers do. However, in both cases these activities are more properly part of quality and project management processes respectively.

This incorrect alignment is even more obvious when you consider that both reviews and effort estimations are part of requirements, architecture, coding and testing as well. When looked at carefully, these activities are often identical, with only minor differences that could be parameterized for the particular domain. Thus, these activities should be considered to be generic process fragments and realigned with the appropriate process components.

It is a natural enough temptation to confuse what people do with an appropriate process component and process system architectural structure. The latter approach will yield a more generic process system, where the former will not.

2.2 Level of Detail

A pervasive problem that affects both use and reuse is that of process complexity [3]. One important contributor to that complexity is the level of detail and the way that detail must be describe because of the process formalism. Clearly complexity and an inappropriate level of detail will inhibit reuse as well as use.

We claim that a useful approach to this problem is that provided by Interact/Intermediate [5]: defining process fragment assumptions and goals (expressed via policies) is a way of combating that complexity and also a way of focusing the fragment descriptions on the important aspects that will contribute to more generic process descriptions. Another side-effect of this approach is the focus it puts on the human-centric nature of the environment: it is the person executing the process who decides when and how to satisfy the assumptions and when and how to satisfy the various goals.

2.3 Abstraction Mechanisms

Without the necessary abstraction mechanisms, it is impossible to define generic processes. One of the standard forms of abstraction is that of parameterization: abstracting values, types, objects, activities, methods, techniques and even processes and subprocesses.

There are several other forms of abstraction that are also useful: primitivation and stratification. Prim-

Practical Issues in Process Reuse

Dewayne E. Perry
Systems and Software Research Center
Bell Laboratories
Murray Hill NJ 07901

1 Introduction

There are two fundamental considerations that must be taken into account for in supporting software development processes: the necessity of a human—centric process environment and the extremely dynamic and concurrent nature of software development processes (especially in large—scale software development, but even in small—scale as well) [6]. This context is the basis for discussing various practical issues in process reuse.

In this paper I will discuss two rather different approaches to process reuse: the first is the use of generic processes [2], and the second is the use of requirements for best in class processes [9]. The first approach aims at providing a generalized approach that allows for as much commonality across projects while supporting the necessary customization and tailoring for each individual project. The second approach is one that is perhaps more realistic for company-wide process reuse where the differences between projects may be too great to be accommodated by generic processes.

2 Generic Processes

There are several dimensions to generic processes: the separation of concerns, the level of detail, and the abstraction mechanisms. Each of these dimensions affects the generality of process descriptions: some choices will enhance the generality, others will limit it. I will discuss each of these dimensions and indicate where generic descriptions are enhanced and where they are hindered. Coincidentally (perhaps), the appropriate choices are aligned with choices that are congruent with the contextual issues of human—centric and dynamic/concurrent approaches.

2.1 Separation of Concerns

Analogous to non-functional considerations that limit the potential reuse of product components (by embodying different tradeoffs from the ones desired and not being able to disentangle those tradeoffs from the functionality) are a number of process related aspects that cause similar limitations. Among these limiting factors are

- project and environmental information,
- lack of separable interfaces, and
- inappropriate organization or encapsulation of process fragments.

2.1.1 Separation of Process, Project and Environment

Foremost among these factors limiting reuse are project aspects that are embedded in the process descriptions. The supporting environmental details are almost as pervasive as project related details. In part this is a process architectural issue: where to draw the boundaries between the significant architectural components in a fully formed process system [8].

One sure way to make a process system unusable by other projects is to embed a specific project structure into the process descriptions. For example, three project aspects that are often incorporated into process descriptions are

- project milestones and schedules
- project roles, obligations and permissions, and
- the projects organizational structure.

Equally embedded in process descriptions are the underlying supporting tools and technology. It is relatively easy to abstract from a particular tool to the basic technological activities that represent these tools. Given this approach the particular projects can merge in their own environment descriptions and bind the basic technological activities to the appropriate tools and objects.

These project and environmental components should be removed from the process descriptions, be described separately, and then merged in at project