POSITION PAPER
for the 6th International Software Process Workshop
Support for the Software Process

Policy and Product-Directed Process Instantiation

Dewayne E. Perry
AT&T Bell Laboratories,
Murray Hill NJ 07901

Introduction

There are three essential problems in building software systems [1]: complexity, evolution, and scale. The evolution of large-scale systems is compounded both by the problems of complexity and by the problems of scale. Thus, process models for the evolution of large-scale systems must define approaches to manage the compounding effect of complexity as well as to manage the multiplicative effect of multi-people interaction while dealing with the realities of changing and extending existing systems.. Ideally, we should be able to use the same policies, mechanisms and structures to bound the effects of complexity that we use to support and enforce multi-people coordination and cooperation. Further, these policies, mechanisms and structures should gives us the basis for successfully managing the evolution of these large-scale systems.

Policies, Mechanisms, and Structures

Perry and Kaiser [2] introduced a model of software development environments in which an environment is modeled as a set of policies, a set of mechanisms, and a set of structures. I claim that this model is useful for process models as well. To be effective, policies need supporting structures and mechanisms as means of implementation. Further, there must exist various mechanisms and structures to support the needed flavors of policy enforcement and support.

While one would ideally separate all policies from mechanisms and structures, in reality mechanisms and structures embody within themselves, or at least provide a bias towards, some lower level policies. Even in the traditional separation of policy and mechanism (as for example in operating systems) one does not get a policy-free mechanism but an abstraction of certain policy considerations usually by means of parameterization of critical choices or constraints. It is this same level of separation that I envision in the definition of policies in a process model: there must be available mechanisms and structures that embody appropriate low-level policies with which to implement the high-level polices of the model.

The II-Model — and Example

Let's suppose that we have the following very general, high level policy:

> *Evolution of a system shall proceed by incrementally integrating well-tested components in a coherent fashion.*

A general, high-level process model that implements this policy might look like the following:

> *negotiate the changes*
> *partition the set of components to changed*
> *install the changes and test the components*
> *integrate the changed components and do integration testing*
> *integrate the changed components with the base components and do system testing*

While at a high enough level this captures what we want to do, it fails to provide any insight as to how one might instantiate, implement, or use this model in the context of a large-scale system with its numerous small-grain concurrencies and complicated interactions.

What is needed to support the highly dynamic, interactive and incremental evolution and integration process is a model that is somewhat similar to the very general model, but with several subprocess that are dynamically instantiatable and modifiable for each of the particular instance or class of instances. A primitive version of this model is encoded in Integrate (Infuse with Integration Testing [3]). Integrate marries the ‘‘crowd control’’ facilities of Infuse with a unit test harness system to provide incremental change and integration test management.

In order to support the process described below, the process modeling and instantiation environment must understand aspects of the project management structure, the system architecture and design structure, the implementation structure of the system being evolved, and the various mechanisms and structures being used to implement the process model. The application of the various activities, the instantiation of the various instances of the model, and the determination of process state depends on these various aspects of the system and model support environment.

The following are three subprocesses that are necessary for the II-model (Incremental Integration Model) along with the general activities appropriate to each subprocess:

General Change Determination Process

* *Determine Problems*
* *Negotiate Changes*
* *Partition Components*

Change Unit Process

* *Install Changes*
* *Analyse Unit*
* *Apply Test Harness*
* *Apply Unit Tests*
* *Deposit Unit into Parent*

Integrate Components Process

* *Analyse Integrated Components*
* *Apply Integration Test Harness*
* *Apply Regression Tests*
* *Apply Integration Tests*
* *Deposit Integrated Component into Parent*

Each activity is defined by a set of preconditions and a set of results each of which consists of a set of postconditions and obligations. This is the model I use in the Inscape Environment [4] to define the behavior of operations. There is, however, an important difference: the preconditions, postconditions and obligations are policies that are required to be satisfied, are guaranteed to have been satisfied, or are entailed to be satisfied eventually, respectively.

In the II-Model, activity ‘‘Partition Components’’ plays a key role: it creates the process model structure within which the three subprocesses are instantiated and enacted in various ways. ‘‘Partition Components’’ is applied recursively to the set of components until the various policies have been satisfied in the context of those components. The critical policy choice here is whether the partitioning is done according to the management structure, the design structure, the dependency structure, or by hand. This provides one aspect of instantiation of the general change process. Indeed, the policy might change within the structure itself, reflecting the management structure at one level, the design level at another, and the dependency structure at another.

Instantiation of the ‘‘Change Unit’’ and ‘‘Integrate Components’’ subprocesses depends upon the hierarchy of partitions that has been established by the structure of the product and the chosen partitioning policy. At the leaves of the hierarchy, a ‘‘Change Unit’’ subprocess is instantiated for each component; at the interior partitions in the hierarchy, an ‘‘Integrate Components’’ subprocess is instantiated for each integrated component. The ‘‘General Change’’ subprocess may be invoked at various points in either process according to the dynamic behavior of the process and current state of the product.

An example of this latter kind of instantiation can be seen in the activity description for ‘‘Analyse Integrated Components’’.

```
Activity Analyse Integrated Components
    input     experimental database EDB
    output    error set ES
    preconditions
        every component C in EDB has been deposited into EDB
    successful:   ES == {}
        postconditions:
            all the components in EDB have been analysed
        obligations:
            <none>
    unsuccessful:     ES != {}
        postconditions:
            EDB has semantic errors defined in ES
        obligations:
            some problem set PS is determined from
                the error set ES such that
                some change set CS is negotiated
                    from the problem set PS
                    and CS is partitioned
```

The obligations of the unsuccessful analysis induce the general change subprocess activities: ‘‘Determine Problems’’ results in the problem set that has been determined from the change set; etc.

The policy ‘‘change set CS is negotiated from problem PS’’ may depend on knowledge of the management structure. For example, we might define this policy as ‘‘a change set CS is negotiated from a problems set PS when the all the managers of those responsible for the components in CS up to and including the lowest common manager have agreed to the changes proposed in CS’’. The process by which resolution is reached may be outside the view of the process support environment; however, the agreement of the participating managers should be within its view.

The desire to enforce particular policies depends on the flavor of policy that one wishes to specify. One can imagine cases where strict enforcement is desired in all cases; this is an *enforced policy*. One can also imagine policies that tend to produce very good results, but which are supported rather than enforced; one might divide these into *best practice* policies, *useful* policies, etc. The level of enforcement may well depend on the current state of the process, the project, or the product.

Evolution of the process model and various instantiations 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. Appropriate process model/instantiation support is needed for each of these kinds of change.

Bibliography

[1] Dewayne E. Perry. ‘‘Industrial Strength Software Development Environments’’. *Proceedings of IFIP ’89 - 11th World Computer Congress*, August 1989, San Francisco, CA. Invited Keynote Paper.

[2] Dewayne E. Perry and Gail E. Kaiser. ‘‘Models of Software Development Environments’’. *The Proceedings of the Tenth International Conference on Software Engineering*, April 1988, Raffles City, Singapore.

[3] Gail Kaiser, Dewayne E. Perry, and William M Schell. ‘‘Infuse: Fusing Integration Test Management with Change Management’’. *Proceedings of COMSAC 89,* Kissimmee FL, September 1989.

[4] Dewayne E. Perry. ‘‘The Inscape Environment’’ *The Proceedings of the Eleventh International Conference on Software Engineering*, May 1989, Pittsburgh, PA.