Software Process Support Through Software Configuration Management

Peter H. Feiler[1]
Software Engineering Institute

Software development embodies a range of software processes. Such processes can be captured in software process models. Two of the reasons for describing software processes through models are to document and communicate a particular process to others, and to encode knowledge of the process in a form processable by computer. Software process modeling has received attention by researchers in recent years - as this series of workshop indicates. Efforts are expended on determining what processes are to be modeled, what of power of the modeling language is, and how the model can be instantiated and executed.

This position paper examines a different trend to supporting software processes. Software development environments (SDE) support the evolution of software through teams of developers by providing software version and configuration management and system build functionality. This SDE functionality supports and embodies certain aspects of the software process. We have examined and experimented hands-on with a number of commercial software development environments. The environments include the Rational Environment, Apollo DSEE, Sun NSE, IST Istar, BiiN SMS, and Atherton Software Backplane. These environments have advanced the functionality provided to support software evolution over commonly used development support tools such as Unix Make/RCS, and DEC VMS MMS/CMS.

A number of observations can be made about these environments and the way they have attempted to - at least partially - capture (i.e., instantiate) software processes.

- Separation of mechanisms and policy: This notion has been in practice in operating systems for a number of years. The primitives (mechanisms) should be abstract enough to contain some of the semantics of the process model to be instantiated. For example the concept of managed workspace or transaction provides a higher-level abstraction than the check-out/check-in primitives and a working directory. A good set of primitives does not unnecessarily restrict the ability to build desirable executable process models. Such restrictions are detected when process model instantiations and executions are attempted. For example, the check-out operation usually performs two functions - making a program unit modifyable, and locking the unit to prevent concurrent modification. If optimistic development (i.e., permit concurrent development) is to be supported the user would have to resort to the branching function. Sun NSE directly supports optimistic development, but currently does not provide the primitives to provide serialized development (i.e., locking). Policies can generally be encoded by writing an envelope of functions on top of the available primitives. In summary, slow progress is being made in separation of mechanism and policy and in encoding process models.
- Source code control evolves to software configuration management: Development support in common practice provides source code control for individual files and a system build facility. Newer SDEs have expanded their functionality to support object code management, management of configurations, transparent access to source repositories, management of work areas, and primitives for reflecting development activities. In doing so, they relieve

[1]This work was sponsored by the U.S. Department of Defense

developers from managing versions of object code themselves, from constantly retrieving copies of files from the repository for viewing purposes, and from concerning themselves with the version history of individual files while being focused on evolving a system. The models embedded in the functionality of these environments are often not clearly described by the manufacturer, and sometimes best understood when related to the model in another environment.

- Configuration composition vs. evolution: These are two major approaches in maintaining configurations. Configuration composition refers to defining a configuration template, which lists the components. Together with a set of selection criteria, this template can be instantiated into a bound configuration. The binding may take several steps, such as source variant selection, version selection, tool version selection, and tool parameter selection. Appropriate selection criteria allow for a range of desirable configuration instantiations, such as experimental, conservative, or stable. Apollo DSEE is an good example of this approach. The evolution approach is reflected in environments such as Sun NSE. The user initially creates a system configuration, populates it with elements, and preserves it. Changes to the system are performed relative to a preserved configuration. Most configuration management operations are performed at the level of configurations, while versioning of individual files is largely transparent. The Rational Environment is an environment that combines the two approaches. Each of the two major approaches gives different perspective of a development process.

- Repository-oriented vs. transaction-oriented: Again, two major approaches in managing software evolution will create different perceptions of a development process. The repository-oriented approach (or product-oriented approach) centers its support for development management on the products to be managed. The history of product evolution is reflected in the repository and its organization. Environments, such as BiiN SMS, have applied the repository mechanisms to provide for management of the work area. The transaction-oriented approach (or process-oriented approach) is centered around the changes and the activities necessary for the changes. Istar is an example of a purely process-oriented support facility by directly modeling development steps and integrating it with project planning. Other transaction-oriented environments, such as Sun NSE, take a more modest approach by providing nested transactions as a key concept. A transaction plays the role of a change activity. A transaction log reflects product history. Non-terminating transactions act as repositories. As a matter of fact, one can find a spectrum of environment support ranging from repository-oriented to transaction-oriented: repository, work area management, nested transactions, activities/tasks, trigger-based task instantiation, and plan-based task instantiation.

- Concurrency and coordination: Some environment builders have recognized the need for different degrees of freedom in concurrency and coordination of development. A closely cooperating team of developers working on a particular subsystem want more freedom passing partially complete pieces among themselves than developers of different subsystems. Some environments are supporting both a cooperating team producing the next configuration releasable outside the team, and teams independently working on different development paths of the same subsystem (e.g., field release with bug fixes and further development) or different subsystems (i.e, partitioning of the system). This is new functionality and different environments provide provide different degrees of freedom.

- Scopes of visibility: Related to the previous point is the desire to provide scopes of visibility. Individual developers should be able to made snapshots of their work by freezing a configuration without making it visible to others. Developers should be able to make their work available to their team mates before it becomes visible to a test team or the general public. Various approaches are being tried. Use of access control mechanisms seems to be an obvious choice, but they tend to be one of the least developed areas in environments. The use of multiple repositories (one for each scope of visiblity) has the problem of potentially high cost of recompilation as elements are moved from repository to repository. Some repository mechanisms provide a viewing mechanism based on a status attribute.

Different users are limited to viewing elements with different attribute values. Evolution-oriented environments utilize nested transactions to reflect restrictions in visibility of changes.

The above reflects the state-of-the-art in commercial software development environments and their support for the process of software evolution. Many of these environments are reasonably robust that they can be employed in real projects and scaled up to handle management of large system development. On one hand, it is encouraging to see that there is progress being made in the functionality provided by these environments. On the other hand it can be a little discouraging to see the limitations that exist in capturing software processes, validating their model, instantiating them, and evolving and adapting them. This is, where researchers in the area of processing modeling can provide guidance to environment builders as to appropriate mechanisms to be made available, as well as facilities for capturing process models, executing them, and allowing for adaptations and changes during execution.