1 Session on Process Change

Chair: Carlo Ghezzi, Politecnico di Milano.

1.1 Carlo Ghezzi: “Introduction on change in the software process”

Some initial questions are:
What kinds of change do we need to support, and how is change specific to software processes? What does change-support imply at the formalism, architecture, or user-support level? What mechanisms/policies do we need to support change? How can we reason about change in the process? How can change be managed/disciplined? And keep in mind: where does change occur, when/where does it start?

See also the forthcoming proceedings from the Montreal workshop on Software Process Evolution in January 1993 as a Springer LNCS, edited by Nazim Madhavji.

The session consists of introductions by Alfonso Fuggetta from CEFRIEL/Politecnico di Milano on a “Possible Reference Model of Change” and by Gregor Engels from University of Leiden on “Complementary Viewpoints on Language and Specification Issues”, followed by a discussion.

1.2 Alfonso Fuggetta, CEFRIEL/Politecnico di Milano: “Possible Reference Model of Change”

A general and very simple Reference Model for change is proposed, to cover many underlying formalisms and mechanisms. The emphasis is on how to support process model evolution, not why. The proposed reference model is independent of specific Process Modeling Languages (PMLs), and covers the mechanisms at least in EPOS, Process Weaver, and SPADE. Cf. draft paper by Conradi, Fernström, and Fuggetta on this. Dennis Heimbigner has proposed an alternative reference model at a workshop on Process-sensitive Software Engineering Environment Architecture Workshop (PSEEA) in Denver, Sept. 1992; see later.

Some definitions: External process = part of the world. Internal process model = computerised description of a process, enactable by a process engine.

The following (sequence of) three model variations are identified:

1. Definition of process model: template variation (e.g. symbolic program).
2. Instantiation and binding model to project: enactable variation (e.g. load-image).
3. Execution of model: enacting variation (e.g. runnable image, bound to process engine and with extra state).

The binding sequence of these variations is not fixed. Upon each of these definitions, the following two basic operations are identified: create and modify, i.e. six combinations totally. We can implement any rights, policies, impact analysis, methodologies etc. on top of these basic operations, as well as more complex operations.

The meta-process to improve and evolve the process can be modeled either as a separate process or as a part of the normal process. That is, “processes are software, too” (Leon Osterweil at ICSE’9 in 1987). In the latter case we need a reflective PML that offer these operations/mechanisms. Different change propagation policies are then possible: busy, lazy, ...

The process modeling field has analogies with that of Programming languages, which evolved from offering simple and weekly typed languages in the 1960s, to abstract data types in late 1970s. The goal has been to support more structured development and maintenance of computer programs.

Similarly, the field of Software Engineering has evolved from e.g. Entity-Relationship and Data Flow diagrams in the 1970s, to integrated 4th generation languages with high-level specification models in the 1980s.

This carries over to software process modeling and to process-centered software development environments: We need to model the tools and the humans on one side (dynamic aspect), and the documents and roles on the other (static aspect). And we should separate between how to change the process (process-oriented) and how to change the goal (goal-oriented).

Generally, we need a precise semantics of process models and of changes. The above static and dynamic aspects should be combined into a unifying process view. This combines the processes, products and teams, and their relationships.

The proposed Leiden Approach by Engels and Groenewegen is called MOCCA (Modeling of Coordinated Cooperative Activities). It is object-oriented, perspective-integrating, fine-grained, graphical, and has explicit models of dynamic behavior of objects and their cooperation.

1.4 Discussion

An alternative to “wild” changes is linguistic structuring mechanisms, such as modularisation, parameterized/partial evaluation, dynamic binding, and many other techniques from programming languages. We could also look at capability-based systems in the OS field, based on modules and access control. Both processes and their support environments should be designed for change from the beginning, but this requires proper access e.g. of history data.

So, what new “high-level” PMLs do we need? The present PMLs are still at the assembler level. An interesting observation is encapsulation at compile- vs. run-time: what is close together at compile-time, may be spread out at run-time. And can we go “backwards”, e.g. from enacting to template model?

And how to handle divergence of the internal model and external behavior? Either of these must be pragmatically “corrected”, but this assumes reflectivity. And how to handle uncertainty (incompleteness), not only exceptions? And should we distinguish between temporary or permanent changes? – the latter assuming databases.

From the discussion, it seemed that a (long) transaction mechanism, with Start, transaction and Stop, transaction operations, is needed to ensure atomicity among concurrent actors, and especially because of reflective features. We may also need Suspend, (re)Start, and possibly Interrupt operations? And we must be able to read meta-data, as well as create and modify?

Heimbigner has proposed an alternative reference model for changing process models (at PSEE). This has four steps: 1) Decide changes (policy), 2) Analyze impact of changes (policy), 3) Perform changes (mechanism), and 4) Propagate changes to users (policy or extra operation?). This seems compatible with the proposed model, although it misses the “enactable” state.

It was objected to a fully reflective process model: E.g. MARVEL has one separate, generic meta-process to perform all process model changes, but this is not expressed as a (reflective) part of the total process. Note analogy to the programming language area: Lisp had to “reify” (to explicitly represent) the execution state and the environment to get programmed control.

We can probably learn from software maintenance and reuse. Likewise, traditional Configuration Management (CM) techniques can partly be used to assist the meta-process, e.g. versioning, reviewing, coarse-grain traceability, regeneration etc. But beware of circularity here: CM needs PM, and inversely. However, there are fundamental problems by changing an executing model during its execution (“pulling the rug”), cf. work on dynamic reconfiguration in telecommunication and distributed systems. Both impact analysis and implementation techniques are hard here. There may also be problems with self-reference, since the active parts are used in the system itself, cf. CM of bootstrapping compilers. As an example: The programming environment EAST allows on-line reconfiguration of itself, using its own CM system.

In general, making a change assumes impact analysis to foresee and prepare planned changes, considering fault tolerance. That is, fine-grained (re)validation tools may be needed, such as the Cornell Synthesizer using attribute grammars. But how to “minimize” or optimize the changes needed?
On consistency enforcement vs. activity management: We should utilize both specific and local rules in a product database and dependencies between documents, as well as the rule/trigger and transaction mechanisms of the underlying DBMS.

There were some interesting experience reported from AT&T (see position papers by Votta and Culver-Lozo), which has introduced process engineers over the last 2.5 years. It seems that 15-20 such engineers are required per 100 developers, with 4 being the minimum for 100, and maybe more to get started. The AT&T processes are mostly manually supported, but have undergone hundreds of changes the last years. Process engineers both introduce and modify such production processes, but processes are often very “connected” and not easy to change.

AT&T tries to utilize data about real process performance among the developers, and such data is kept confidential from the managers. There are lots of internal meta-procedures on how to evaluate process change request (PMRs). There are two sources of process improvements: monitoring the actual process, and by scanning literature. But how to optimize processes wrt. the three major, macroscopic variables – quality, cost, and lead-time?

We should be able to simulate changes and perform role plays, then gradually implement the changes, and finally follow up revised process performance. It is important to have a “soft” introduction to process changes, both to get acceptance from the people involved and to reduce risks (but it is also a risk not to change!). However, a replaced process may soon be de-facto re-instated in its old version. Note also, that evolutionary process changes do not cover all cases, sometimes revolutionary changes are needed, which cannot be analyzed satisfactorily in advance.

The experience is, that it can be easy to implement a change once it has been agreed upon, but assessment of a proposed change is hard. And the involved technical people are notoriously not competent or objective to judge the effects of a proposed or implemented change. Cf. the AMADEUS system with an embedded measurement system, and tomorrow’s session on Metrics.

Lastly, there is also “panic” procedures in most organizations – 95% of time? (laughter!). Indeed, exceptions take a lot of time, cf. fire drills to maintain capability.