from the Proceedings of the 8th Inter. Software Process Workshop (ISPW8), Wadern, Germany, March 2–5, 1993

Process-Centered Environments (Only) Support Environment-Centered Processes

Alexander L. Wolf

Department of Computer Science
University of Colorado
Boulder, CO 80309 USA
(alw@cs.colorado.edu)

David S. Rosenblum

Software Engineering Research Department
AT&T Bell Laboratories
Murray Hill, NJ 07974 USA
(dsr@research.att.com)

The software process research community has concentrated a large measure of its effort on the problem of how to "computerize" the software process. In particular, the notion of a *process-centered environment* has emerged, dominating software environment research for the past several years. Among the many benefits touted for process-centered environments are the ability to automate various aspects of a process and the ability to monitor the progress of a process in order to guide, enforce, or measure that process. This approach has shown great promise and indeed has even shown some early successes. Unfortunately, this emphasis on computerization in general, and on process-centered environments in particular, tends to focus attention on exactly those aspects of process that *can* be computerized, while giving short shrift to those aspects not amenable to computerization.

This issue became clear during the past year as we studied a large, mature software process in use at AT&T. We performed the study as part of an effort to develop process data capture and analysis techniques that could support the critical task of process improvement [4]. Our approach was to focus on the dynamic aspects of the process, such as the order of, and time taken by, each step in the process, as opposed to, say, the static roles and responsibilities assigned to project personnel or the static relationships among tools and product components. We took this approach in part because the process's dynamic aspects were the least well understood. In addition, we hypothesized that process problems ultimately lead to wasted intervals of time and that those problems can best be revealed by retrospective analysis of characteristic time intervals. For instance, a period of inactivity between the time a meeting is scheduled and the time the meeting takes place may reveal poor planning for activities that require a long preparation time. Bradac, Perry, and Votta have also begun to explore this hypothesis [1].

In order to analyze a process's characteristic time intervals, it is first necessary to capture the relevant data about the significant events of the process, including the times at which those events occur. To help define and structure this task, we developed a simple event-based model and taxonomy of process activities. An event in this model is an instantaneous happenstance within a process activity. Certain events, such as the beginning and ending of an activity, are useful for defining characteristic event intervals. The event taxonomy comprises six categories of events:

1. communication;
2. automation (i.e., tool);
3. analysis;
4. work;
5. workday; and
6. decision.

The taxonomy is instantiated for a particular process by identifying appropriate process-specific event kinds in each of the six categories. An individual enactment of the process is then characterized by the set of all events of the defined event kinds that occurred in the enactment.

The particular process we studied at AT&T is a software build process that is regularly repeated with little change in its basic, day-to-day activities. The group responsible for official builds enacts the several-day process once every few weeks. The software, which consists of several million lines of source code, is managed with the aid of a database-oriented change management system and is built using three major tools. The build process involves several roles. The *build owner* coordinates the process, tracks down build problems, and communicates with *developers* dispersed around the world. The *build administrator*, at the direction of the build owner, sets up the actual builds for execution according to a written guidebook. The build owner also typically has several *build assistants* to whom problem-tracking chores can be delegated.

One would think that this sort of process would be perfectly suited to enactment in a process-centered environment; in fact, (imaginary) build processes are often used to exemplify such environments. The reason one might have this impression is that idealized software building is an inherently *environment-centered process*; humans interact with tools to automatically transform environment-managed objects from one form to another.

It turns out, however, that the very real build process that we studied exhibits characteristics that are as much centered on non-environment factors, such as project organization, project finances, and human relations, as they are centered on the computing environment. In particular, after examining the critical events of the process, we concluded that activities occurring completely outside the purview of the computer had at least as much influence on the process as those occurring on the computer. We found, for instance, that inter-human communication is a significant pacing factor in process performance. In other words, the process is at least as (human) communication bound as it is compute bound. But the important point here is that communication, for the most part, is not conducted via the computer and, therefore, is not immediately "visible" to the environment. Other examples of critical process events not centered on the computer are the decision events that mark major turning points in the process; while certain factors contributing to a decision are related to computer-based activities, other factors, such as the approaching end of a work day in a foreign country seven time zones away where developers do not work overtime, are not as obviously accessible. We suspect that the situation we found in this build process characterizes many other kinds of software processes as well, such as processes for requirements and design specification.

Some process researchers have, of course, recognized this problem (see, for example, [3]). The current approach typically employed to address it is one that involves incorporating "pseudo-tools" or other such surrogates for humans into a process. That approach is probably adequate for most kinds of deductive analyses performed on abstract models of software processes, but can be awkward for something such as "real-time" process measurement and feedback as envisioned for many process-centered environments (e.g., in Arcadia [2]). In particular, it seems to suggest that the human must perform some sort of immediate data collection and entry in the guise of an "automated" process agent. More generally, the surrogate approach takes too narrow a view of the factors contributing to the dynamics of a software process.

There are some very hard problems to be solved here. In our study of the software build process, for example, we were compelled to develop a costly, labor-intensive technique for process capture that relies upon independent, direct observation to record events not occurring on the computer. This then led us to the problem of how to relate those manually recorded data with other, automatically recorded data. Beyond the problem of capturing data about a process, there is a critical modeling problem arising from a need to coordinate computer-based and non-computer-based process activities. In particular, how can a model of the "outside world" embedded within an (executable) environment-centered process be coordinated with what is actually going on in that outside world, and how does that process become aware of, and adjusted to, changes in the outside world?

In summary, the issue we see is that for essentially pragmatic reasons, recent work on process-centered environments has focused attention on the automation aspects of software processes, with little regard for non-computer-based activities. Moreover, the examples used to validate research ideas, namely environment-centered processes, do not adequately take into account the effect that those activities have on the software process as a whole. We witnessed first hand the importance of non-computer-based activities and hope to see the current scope of research on software process broadened to encompass these critical elements.

References

[1] M.G. Bradac, D.E. Perry, and L.G. Votta. Prototyping a Process Monitoring Experiment. In *Proceedings of the 15th International Conference on Software Engineering.* IEEE Computer Society, May 1993. To appear.

[2] R.W. Selby, A.A. Porter, D.C. Schmidt, and J. Berney. Metric-Driven Analysis and Feedback Systems for Enabling Empirically Guided Software Development. In *Proceedings of the 13th International Conference on Software Engineering*, pages 288–298. IEEE Computer Society, May 1991.

[3] S.M. Sutton, Jr. Accommodating Manual Activities in Automated Process Programs. In *Proceedings of the 7th International Software Process Workshop*, October 1991.

[4] A.L. Wolf and D.S. Rosenblum. A Study in Software Process Data Capture and Analysis. In *Proceedings of the Second International Conference on the Software Process*, pages 115–124. IEEE Computer Society, February 1993.