A Role-Based Empirical Process Modeling Environment

Brendan G. Cain
James O. Coplien

AT&T Bell Laboratories

Abstract

Much contemporary development process research is based on analyses of process steps, their duration, and the events they propagate. Our initial research in large, mature telecommunications development processes concluded that such models do not capture abstractions that remain stable over time. We turned our attention instead to empirical role-based models. The basic abstraction in the model is a ''role,'' a longstanding, stable locus of associated responsibilities in a process. A process model evaluation prototyping environment is used to visualize the process data in several ways, including community-of-interest clustering, communication network clustering, and hierarchical rendering. Analyses of these models have led to powerful insights both into individual projects, and into the properties of software development processes in general.

1: Introduction

There is abundant contemporary interest in improving the lot of software development by tuning, modernizing, or redoing a project's development process [1], [2]. Process improvements include both those that result from the infusion of new technology, as well as those owing to changes in organization, methodology, or life cycle. The work described here investigates process improvements that might arise from understanding the flow of information in a development culture. Most solutions suitable to this viewpoint seem to lie in the areas of organization, methodology, and life cycle. However, our current work is focusing on simply *understanding* what happens in software development, both for individual projects and, to the degree possible, for the industry as a whole. Our goal is that this understanding establish a basis for software development process improvement, both in the short term for existing projects, and in the long term for current projects and projects not yet chartered.

This paper describes our progress in formulating methodologies and modeling approaches, and in developing tools to support an overall process evaluation framework named *Pasteur*. We felt it important to build a framework in which processes could be understood, compared, modeled, and improved. Furthermore, we have striven to validate that framework against real development processes in organizations collaborating with us in this research. The work has gone through several iterations, culminating in a process modeling environment that is producing useful results. We still consider the work to be in its formulative stages; work in progress and considerations for the future are both discussed in this paper.

This paper starts with an overview of our initial models and early work, and explains why we decided to change modeling techniques. It then describes the role-based modeling approach and how we gathered data for our process models, followed by the process manipulation techniques we found useful. The paper concludes with some striking results and discusses their ramifications both for the individual projects we studied, and for software development in general.

2: Early work

From the outset, we took great pains to ensure that our models would track the activities of the real-world organizations and processes they represented. In our first modeling efforts, we felt that the best way to derive a model would be by instrumenting primitive process activities, and building higher order abstractions out of these low-level tasks. We evaluated using several ''hooks'' into the development processes, including the monitoring of electronic mail [3], phone calls, and detailed developer/machine interactions. We settled on the latter as a starting point because it was easy to gain access to them, and because most process activities for the organizations we were studying involved computer-resident artifacts.

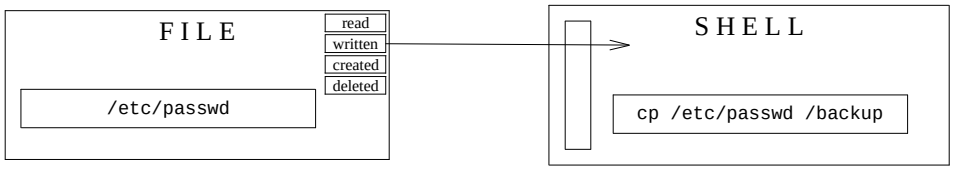

Figure 1: A Simple Process Circuit

We built a modeling environment that allowed us to draw a picture of a process in terms of primitive objects such as command invocations, files, programmers, and machines. Each object had ‘‘handles’’ representing its externally visible attributes or activities; for example, a file object had handles for ‘‘written,’’ ‘‘read,’’ ‘‘created,’’ and ‘‘deleted’’ (Figure 1). Objects could be created and handles wired together using a graphical programming interface, forming a process ‘‘circuit’’ of primitive connectors such as logical ANDs, ORs, event latches, and predicate applicators. Many aspects of the environment were weakly analogous to what one would expect to find if using Petri net formalisms [4], as in the Role Interaction Net (RIN) language [5]. The environment might also be viewed as a graphical design-by-constraint language comparable to GRAPPLE [6]. To help the environment scale up to handle large processes, aggregate abstractions could easily be created for common groups of objects (e.g., related collections of files) and entire circuits could be abstracted into new, user-defined objects to use in building a process model.

The environment was to have four uses: capture, data gathering, simulation, and static analysis. Process capture is just putting the process into machine-readable form. Each picture would be worth its proverbial thousand words of corresponding textual descriptions, an aid to understanding the process and communicating about it. With the model in place, we could ‘‘teach’’ each of the abstractions to gather data about its corresponding entity in the real-world process. That is, a file abstraction could keep track of how many times its real-world counterpart was checked out, read, or written, or even keep a more detailed log of all activities relating to a file. These data could be used to annotate the process ‘‘circuit’’ model with stochastic parameters of frequency and duration of interactions between the primitive objects, and this stochastic model could be used to run what-if simulations for a modified development process. Lastly, these same parameters could be used for simple, static critical path process analyses that could be used to do what-if experiments to reduce process interval.

The environment in place, we undertook the modeling of a software integration process that typically cycled once every two weeks, and that involved about 40 people full-time. The effort was an immediate disappointment for two reasons. First, people had a difficult time articulating what they did in concrete terms that mapped onto the abstractions we could express. Rather than interacting along well-defined data paths, the development organization viewed itself as a whole whose individuals moved together in quanta along major stages of progress. They viewed their individual activities at each of these states as ‘‘doing what was needed to get the job done.’’ This led to the second problem, that there was little to be found in the way of repeatable patterns. We held little hope of developing stochastic models from data which, at this level, seemed to point to a chaotic process.

We stepped back from the process modeling effort, conducted some additional interviews, and began to form a hypothesis that the processes we wanted to study *were not stochastically well-behaved at the level of individual activities*. Individual developers often became blocked for want of a resource, deliverable or event involving another process activity or external event, and these losses of continuity made it difficult for developers to establish overall cause-and-effect relationships within their process. That is, while an entire process may cycle with statistical regularity on long time scales, it seemed impossible to derive any of the large-scale interval data from data observable by individuals, or data that could be derived from our tools. We likened these processes to ant colonies, which can accomplish observable tasks and exhibit behaviors as a whole, behaviors that cannot be extrapolated from analysis of individuals’ activities [7]. We also called it a ‘‘Petri dish model’’ (with due apologies to Petri nets) because many of these processes behaved like cultures of organisms that moved en masse from one dish of agar to another, devouring each in turn.†

We set out to find a new approach. We needed to break not only with our original approach, but with much of the tradition of canonical process modeling. The definition of *process* in the software engineering community has converged on ‘‘a set of partially ordered steps intended to reach a goal,’’ as Curtis *et al.* cite

† This led to the current name for our modeling environment, Pasteur, which is also an obscure acronym of lesser importance.

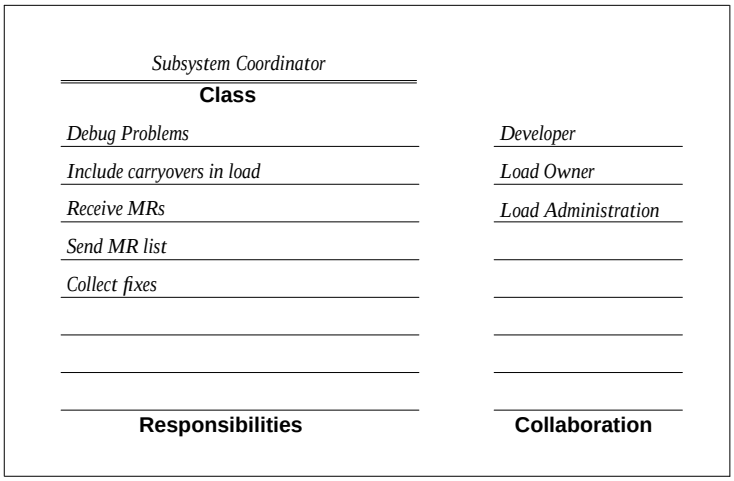

Figure 2: CRC Card

Humphrey. [2] The properties of a process lend themselves to formal analyses when expressed in this form, and are claimed to be difficult to analyze at all unless formally constrained. Few of the processes we study can be characterized as comprising any partial ordering of steps or events, but are *ad hoc* in their internal interaction patterns. Successive iterations of a given process provide new context, new structure, and new people that change the orchestration of events in ways that are difficult to avoid. Processes adapt to move with trends and to track ever-changing markets. Some new process designs based on iterative approaches deliberately maintain an *ad hoc* flavor, and do not seem well-suited to the established process modeling approaches in the first place. We nonetheless knew that many of these processes were competitive and effective, and that they demonstrated a consistency of external behavior that belied the chaos at the level of individuals and internal events. Many of these processes could adapt to new environments without major changes in structure. We felt a need to find stable abstractions on which we could build models to understand, analyze and evolve these processes.

3: A role-based approach

We decided to pursue a new modeling technique that would better suit our research goals of formulating insightful families of process models. We decided that what needed changing from the Pasteur prototype was not so much the *level* of abstraction, but the *units* of abstraction. The procedural, data flow abstractions were not robust in light of change; the interactions between them, which were even more central to analysis, were even more likely to change over time.

Instead of modeling *steps*, we decided to model *roles*. Roles are the building blocks of organizational structures, and are a longstanding abstraction for sociological and psychological research [8]. Manager, Integration Tester, and Subsystem Coordinator are examples of common roles. Interviews with the software integration team suggested that the roles in place today are much the same as they were more than a decade ago, even in light of orders of magnitude of project growth and introduction of several new generations of product and support technology. Data gathered on roles was more likely to have statistical significance than for the procedural model. Most other models we considered—state machines, activity-based models, and others—either addressed the problem at the wrong level of abstraction, or were overly reminiscent of the disappointing Petri-like model we had abandoned. From these conjectures, we decided to try the role-based approach.

The new model was based on *CRC Cards* [9], a modeling approach originally designed for system software architectures exemplifying object-oriented techniques. ‘‘CRC’’ stands for *C*lasses, *R*esponsibilities, and *C*ollaborators, which are the fields on the cards (Figure 2). Responsibilities define what a role offers to its community; collaborators enumerate stake-holding relationships between roles.

Though CRC cards were originally designed for system software architecture formulation, we thought they would be suitable for capturing the essential and interesting properties of organizational roles. Later consultation with Kent Beck confirmed that these cards have already been used for organizational modeling by Sam Adams and others [10]. We prefer the use of CRC cards to more ‘‘formal’’ role modeling tools, such as RIN

or GRAPPLE, because:

- The informality of CRC cards creates opportunities for better empirical models, allowing people within a process to develop process models at their own level and articulate the process on their own terms;
- Most existing formal tools bear vestiges of the Petri net and constraint-based models which were the subject of our earlier unfavorable experience.

The cards were created in an exercise using 3 x 5 index cards to capture information about a project. These cards are created from a brainstorming exercise that identifies stable abstractions (or ''classes,'' which we redesignated as ''roles'') in the development process, and then elaborates their responsibilities and collaborators using development scenarios. *Role*s are longstanding jobs within a process, usually intuitively recognized by its culture, for which relatively stable job descriptions may exist. *Responsibilities* are the tasks and activities for which role-players are accountable to their peers. *Collaborators* are roles to which a given role is tightly coupled: Roles are collaborators to the degree either becomes nervous if the other goes on vacation. (One type of data we collected was the strength of collaborations.) The focus is not on the scenarios, but rather on the roles themselves. We would later turn to studying pairwise interactions between roles, and are currently evaluating full animation of this model.

We revisited the software integration process and proceeded to collect CRC card data in a half-day session with project technical staff. Relevant roles were first identified using a group brainstorming process, after which the list of candidate roles was evaluated and refined. Cards representing specific roles were given to the individuals playing those roles in the software integration process. Role players were also assigned for the parts of the process external to software integration, but with which software integration interfaces in its work.

CRC card abstraction icons were then added to the Pasteur prototype. The Pasteur environment holds a transcription of the CRC cards wrapped in a hypertext database named Eggs [11], based on the HAM hypertext system [12]. The existing prototype was used as a base so that existing graphical interface primitives could be reused from the prototype. Pasteur allows the programmer to create a card abstraction on the screen, and type information into its fields using a modeless editor. Cards can be resized and moved about the screen. On a typical color work station monitor, we can easily fit about 40 full-size non-overlapping cards.

Now we had the ability to capture stable development process abstractions on-line, and the next step was introducing semantics to support analysis. We decided to focus on three kinds of manipulation of the CRC card interface: browsing, clustering, and animation. Browsing is simply the ability to locate easily a given card and its collaborators. Other kinds of browsing, such as finding all cards sharing a given responsibility, were also considered and are still under consideration for later implementation. Clustering is the ability to group cards based on their strength of mutual collaboration or the sharing of some properties in common. For animation, each card is represented as a finite state machine that models the behavior of its real-world role. By placing all cards in a known state, and introducing a stimulus to the system, we should be able to determine the next state of any given card. Most of our current work is focused on clustering and graphical visualization techniques, with process animation left to future work.

4: Gathering the data

Data in hand from the software integration process, we proceeded to some simple analyses. One obvious analysis was to illuminate all the roles having a strong coupling to a chosen role to determine the centrality of that role in the process. A role was considered central to the process to the degree it had strong coupling with remaining roles in the process. One curiosity we discovered early on was that the Developer role had strong coupling with most of the internals of the software integration process. On its face, this was a surprise, as the developer is supposedly isolated from the details of software integration.

Further data-gathering and analysis exercises confirmed our faith in the CRC modeling approach. In debriefings held with the development organizations to review our pictures of their processes, developers reported that our models matched their intuition of how their processes worked. Somewhat paradoxically, most developers felt they could not have drawn the pictures from *their* model of how their processes worked: that the data-gathering exercise itself had changed how they viewed the process.

We received some instructive subjective feedback on the state of development processes during data-gathering exercises with development organizations. To approach these organizations, we introduced ourselves as researchers who wanted to collect data about their process, and to gain a better understanding of how processes work in general. Most groups we interviewed had already developed specifications of their processes to support training and ongoing development. We were aware of these specifications, but most reflected the kind of event-and-task perspective we had found difficult to analyze in our early work. In introducing ourselves to these groups, we often needed to motivate people with our

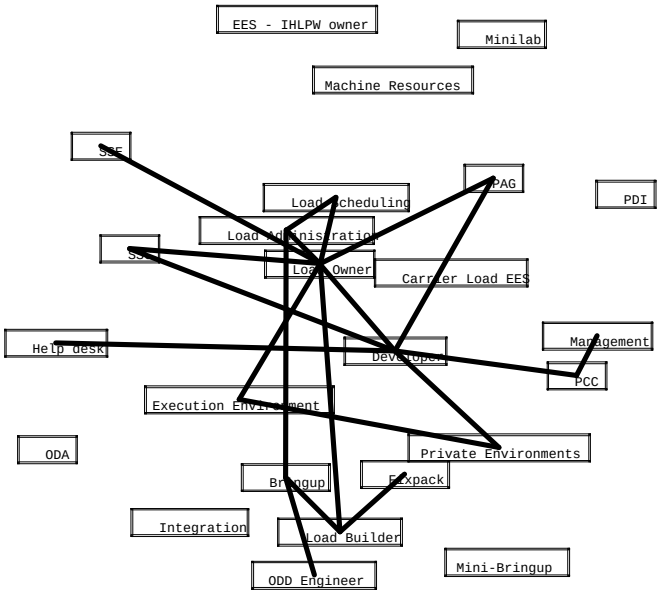

Figure 3: A Mature Development Process

need to collect process data from a ‘‘different perspective’’ than was present in their own pre-existing models. We were politely tolerated in anticipation of what, after all, would only be a one-time, three-hour data gathering exercise.

Then, in the middle of the CRC scenarios, a wonderful thing would happen. One or more of the participants in the exercise, unprompted, would interject something to the effect that: ‘‘We take back everything we told you about how well we understand our process!’’ The analytical role-playing in front of an external audience—us—caused the participants to see major aspects of their process interactions they had never perceived before. After this occurred several times, we started using the phenomenon as a selling point to gain entry to new organizations and convince them to spend three hours of their time to support our research—and to better understand their own process.

5: Manipulating processes

We soon were able to build a library of development processes for organizations both at our location and other AT&T locations, as well as a small number of organizations doing related work outside AT&T. The next set of challenges that arose was how best to manipulate the models we had gathered to further our insight into the issue of software development process in general, and to better support those giving us data (because good support for them meant continued high-quality data from them).

It was about this time that we added clustering facilities to our modeling tool: the ability to draw a picture of an organization where closely coupled roles were placed close to each other on the display. Several approaches were evaluated. The approach most commonly used, and which most often generates intuitive results, is a force-based approach. Each role is modeled as a charged particle, such that all roles repel each other with a certain strength. This repulsion is balanced by attraction proportional to the strength of collaboration between given pairs of roles (the strength of each collaboration is gathered in the interviewing process). A simple relaxation algorithm is used to converge on a stable picture of the process where cliques, strong patterns of coupling, and outlying roles are visually obvious.

The coupling between individual roles can be depicted by connecting them with lines (Figure 3). The width and color of each line is derived from the strength of the coupling between the roles it joins. Though our model has the flexibility to express semantics of directed interactions, for the purposes of this paper the interactions may be thought of as undirected mutual collaborations. Making these lines visible helps to highlight patterns of interaction in the process under study.

More advanced facilities were added as well. Many of our development colleagues wanted to depict what their processes would look like if a certain role or collaboration relationship were omitted. We added editing facilities to temporarily exclude selected roles and interactions from clustering analysis. One exotic facility we added was the ability to merge multiple processes into one. Using this

Figure 4: A ‘‘Messy’’ Process

capability, we could foresee how proposed process mergers might ‘‘look’’ if they followed the natural patterns of coupling present in the component processes. We were able to provide some of these merged models to organizations before their actual merger, giving them insight on what their core roles might be.

6: Process observations

In addition to supporting short-term development needs, our process work also proved satisfying in meeting our research goals of understanding the essence of aspects of software development. We observed common patterns in most of the processes we studied, patterns that were not anticipated and that still evidence the need of further study. We actively solicit the input of the reading audience to help develop theories for these observations, as well as to inform us whether they are ‘‘universal’’ or just peculiar to the mature, large system developments we studied.

6.1: Process typing

As anticipated, we first found that a development community has several different types of processes. ‘‘Type’’ in this context might be translated ‘‘rough shape,’’ the general appearance of the clustered process picture. Many were spoked-wheels with fat hubs; that is, they had a center of activity with more loosely coupled, surrounding support roles. Most other processes had a fairly even distribution of strong interactions between roles at all locations on the picture. These, we classified into two subcategories: ‘‘messes,’’ and ‘‘mature processes.’’

To distinguish a mess from a mature process might be possible from the picture alone to a discerning reader, but was equally supported by our own first-hand knowledge of the process characteristics. In a mature process, one that has evolved over a decade in a relatively constant domain, load balancing takes place. Information flow is broadly distributed across process roles (Figure 3). The network of communication between roles is not fully distributed, but the total distance between two roles is rarely more than two hops, and there is a good mapping between direct, strong coupling and ‘‘need-to-know’’ relationships. We also know that these processes have largely distributed decision-making loci and are not centrally orchestrated. In fact, the Manager role in these processes is usually at the outer edge, with tighter coupling to resource organizations (e.g., the computer center) than to the principals in the process! There is likewise no centralized decision making in a ‘‘mess,’’ and managers are likewise at the periphery of the process. However, a ‘‘mess’’ process tends to be a more fully connected graph than a ‘‘mature’’ process, indicating that the communication overhead on each role is unfocused and intense (Figure 4).

Another kind of process pattern looks like a collection of galaxies orbiting about each other. Some processes

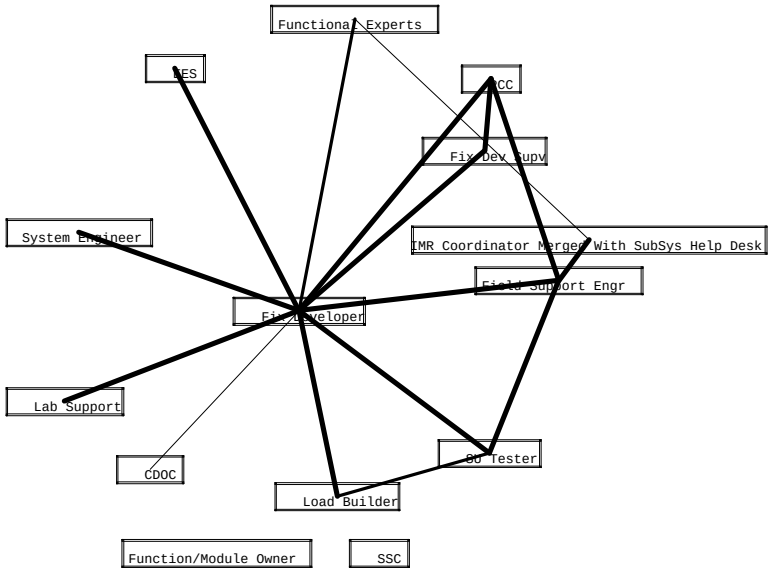

Figure 5: A Hub-Spoke-And-Wheel Process

form natural clusters, or *cliques*, that are self-organizing communities of interest (Figure 5). Such taxonomies can usually be explained in terms of particularly well-defined separations of concern within a process, and were not a surprise to us. They do make some of the most aesthetically pleasing pictures.

One thing we learned from this is that processes *need not be well-clustered to be good performers*, even though they may have head count far above the seven-plus-or-minus-two figure often cited for optimal communication. We find that there is a strong correlation between the maturity of these processes and their ability to find optimal distributed clusterings. An interesting exercise would be to determine whether such processes can be designed to work effectively without having to undergo a decade-long maturation period. Another exercise would be to determine whether these clusterings represent local optimizations that might be outperformed by radical reorganization of roles and the interactions between them, without upsetting the external interfaces to that process. These continue to be important goals of our ongoing research.

We are formulating further hypotheses at this writing to support models that explain why some processes become ‘‘messes,’’ while others apparently remain well-structured. One theory being discussed is that product-oriented processes (among which is the process of Figure 3) tend to have well-defined cycles. The duration of these cycles is from days to months. Service-oriented processes (such as that of Figure 4) do not have well-defined cycles, but are expected to respond within minutes, hours or at most a couple of days. Processes with low-latency response times may need higher coupling to be responsive, just as high-performance software systems often take advantage of tight coupling to gain performance. This suggests that so-called ‘‘messy’’ processes may not be bad—just optimized according to different criteria than for the processes used by product organizations. Managers should be admonished against tampering with such processes just because roles have close mutual coupling [13].

It is our hope to gather enough process data to be able to classify processes, such that processes within a class have a uniform correlation to interesting characteristics such as head count, cycle time, and product size.

6.2: Developer-centric processes

Every process we analyzed contained a role called Developer, or its analogue: the person who produces code or other products, and who is often also the person to do the design. Without exception, every process clustering we looked at found the developer at the center of the process! Developers—who live at the bottom of the food chain, so to speak—are the roles around which the entire process rotates. It doesn’t matter whether it is a testing process, a problem tracking process, a support organization process, or anything else: the developer is still at the center. The property appears to be independent of whether the process is a hub-and-spoke, cliqued, or ‘‘mess’’ process. The developer role is not customarily viewed as the focus of decision-making activity within a process: most developer interactions are informational or related to ‘‘problems’’ in execution of the process.

Being at the center, this role bears an unbelievable overhead. When we first saw this pattern, we whimsically joked that it was no wonder developers never got anything done. In fact, later analyses bore out that developers spend about two-thirds of their time blocked along a given development thread, waiting for information or input from other roles. (Because developers work several threads in parallel, however, they are rarely idle).

This analysis points to major opportunities for process improvement, and why one might want to focus on improving the lot of the developer. But it also points out an unforeseen value: the developer is the perfect person to pass information from one role in the process to another.

One thing we had expected to find was the ''gatekeeper'' role described by Allen [14], the one responsible for most ''scholarly'' contacts outside the organization. We expected the gatekeeper role to appear as a hub of sorts in our clustered process models, but this appears to be the case only in a single process (the problem tracking process). We doubt this role is vested with the developer role, which is usually at the center of the picture. We conjecture that the gatekeeper role will appear only in analyses of the interactions between multiple processes, an analysis which is currently in its fledgling stages.

One exercise we would like to do is gather and merge *all* the processes within a given development community and observe the role of the developer in that picture. It is likely that the developer serves as the major communication channel through which most important project data travels in the processes we analyzed.

6.3: Whither management?

In most processes, management showed up at the periphery of the model. One might speculate that management should be in the center, overseeing most activities and orchestrating the entire process. While this may be true in some domains somewhere, we are finding that it is definitely not the norm in software development.

Management roles tended to have high coupling to roles providing services or resources to their process. In a single testing process, we found a manager at the center. However, as mentioned above, the developer role was *also* at the center, with coupling patterns similar to those of the manager. We suspect that these manager-centric processes should best be modeled as hierarchical processes, using the hierarchical rendering facilities present in our environment.

6.4: Processes under stress

We like to view processes as organisms with a life of their own and, like all organisms, they exhibit defense mechanisms under stress. Using separate network analysis tools [15], an animation was performed that confirmed a tradeoff between domain experts' coupling to the process, and management coupling to the process. Management takes over when the process ''gets in trouble,'' but seems only to facilitate resource acquisition and coordination support under normal operation. In summary, the management team members might be thought of as the ''antibodies'' of the process organism.

7: Summary

We have realized satisfying results from a role-based, empirically rooted program in development process modeling. The work has proved useful to the development organizations under study, as well as the long-range programs of the research organization conducting the study.

Our tentative conclusions from the studies to date are:

- The developer role, or its analogue, is the primary channel of communication within (and potentially between) all processes.
- Service-oriented processes have more intricate internal structure than product-oriented processes.
- There appears to be a balance between the involvement of managers and domain experts in a process, with a high correlation between managers' involvement and the stress level of the process.

Acknowledgments go to Steve Eick for useful discussions and for use of his network analysis tools, to John Puttress for use of his hypertext database, to Larry Votta for his insights from related work, to Dave Weiss for his many useful suggestions, and to our many development collaborators and partners in this program. Many thanks, too, to the conference referees for their encouragement and suggestions.

References

1. Kitson, David H., Steve Masters. ''An Analysis of SEI Software Process Assessment Results: 1987-1991.'' Carnegie-Mellon University.

2. Curtis, Bill, Marc I. Kellner and Jim Over. ''Process Modeling.'' *Communications of the ACM 35*, 9, September 1992, 75-90.

3. Crowston, K., T. W. Malone and F. Lin. ''Cognitive Science and Organizational Design: A Case Study of Computer Conferencing,'' *Human Computer Interaction, 3*, 59-85.

4. Peterson, J. L. *Petri Net Theory and the Modeling of Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1981.

5. Rein, Gail L. *Organization Design Viewed as a Group Process Using Coordination Technology.* MCC Technical Report Number CT-039-92, February, 1992.

6. Huff, K. E., and V. R. Lessor. ‘‘A Plan-Based Intelligent Assistant that Supports the Software Development Process.’’ *Software Engineering Notes 13*,5, 1989.

7. Hofstadter, Douglas R. *Gödel Escher Bach: An Eternal Golden Braid*. 1979.

8. Sarbin, T. R. and V. L. Allen. ‘‘Role Theory.’’ *The Handbook of Social Psychology, 2nd Edition*, G. Lindzey and E. Aronson, eds. Reading, MA: Addison-Wesley, 1968, 488-567.

9. Beck, Kent. ‘‘Think Like An Object.’’ *UNIX Review*, September 1991.

10. Personal communications with Kent Beck.

11. Rizk, A., N Streitz, and J. Andre, eds. *Hypertext: Concepts, Systems and Applications* (Proceedings of ECHT ’90), Cambridge University Press, 1990. ‘‘The Toolkit Approach to Hypermedia,’’ by Puttress, J. and N. Guimaraes, 25-37.

12. Campbell, B., and J. M. Goodman. ‘‘HAM: A General Purpose Hypertext Abstract Machine.’’ *Communications of the ACM 31*,7, (July), 856-861.

13. Deming, W. Edwards. *Out Of The Crisis*. Cambridge, MA: MIT Press, ©1986, Chapters 7 and 11.

14. Allen, Thomas J. *Managing the Flow of Technology*, Boston: MIT Press, ©1977, 141-182.

15. Becker, R. A., S. G. Eick, and A. R. Wilks. ‘‘Basics of Network Visualization,’’ *IEEE Computer Graphics and Applications*, May, 1991.