Annals of Software Engineering 2 (1996) 259–286

Social patterns in productive software development organizations

Brendan G. Cain and James O. Coplien

Bell Laboratories, Naperville, IL, USA

Neil B. Harrison

Bell Laboratories, Westminster, CO, USA

Software development is a predominantly social activity. It is important to view software development groups, departments, and corporations as social bodies. We study software organizations using a novel data-gathering approach that combines several techniques commonly used in social network analysis. Our techniques differ from those of ordinary social anthropology in that we help the organization introspect about itself; the technique is a "mirror" for the subject organization. We catalogued social network diagrams using a variety of visualization techniques. We have found visual patterns that correlate well to subjective measures of a good organization. We built analytical models to capture properties of the social networks, employing techniques similar to those used in established social network science. The emerging design technique based on architectural patterns provides a good vehicle for communicating organizational patterns. We have captured practices from outstanding organizations in a group of patterns that form a "pattern language" for productive software development.

1. Introduction

Software development is a predominantly social activity. It is important to view software development groups, departments, and corporations as social bodies. Software engineering too often pushes social concerns aside, perhaps dismissing them as unscientific and therefore as being ill-suited to a so-called engineering discipline. The essentially human nature of customer interactions, programmer creativity, and programming team dynamics demand that we deal with the social side of software production enterprises.

Organizational behavior is an area that most computer scientists view through the eyes of local management policy or process standards, rather than through the conventions of organizational psychology and social anthropology. Computer scientists distance themselves from the organizational concerns they disdain as non-technical as organizational analysts perhaps distance themselves from the computer scientists they view as asocial. Computer science aspires to be a science, and does so by distancing itself from imprecision and subjectivity. The work we present here does not pretend

to be an experimental science (true of much of what passes as computer science): it is based on case studies that explore frontiers for which good science has not yet been developed, laying the groundwork for science that may follow later (see [Schneiderman 1980], sections 2.2 and 6.5). It is also important to remember that the subjects of this work aren't MIPs and megabytes, but living, feeling human beings with multidimensional lives. The number of experimental variables is large for any study of human endeavor, so care must be taken to match experimental results with the mores of subject organizations.

Our research takes a sociological view of software process. We are not the first to adopt this perspective; sociologists have certainly studied technological organizations before (e.g., [Brajkovich 1994]). However, we are software engineers who have learned to use the tools of sociologists, not sociologists who have learned software. We feel that our perspective helps us cast our results in terms of the processes, artifacts, and methods that are acceptable standards of software culture.

One goal of our research is to understand the broad underlying structures in software development organizations, and thereby better understand the activities of software development and how to shape organizations according to market needs. This is particularly important as the industry emerges from top-down design and embraces techniques like "round-trip gestalt design" [Booch 1994], drifting from waterfall development models to spiral models and iterative development. While object-oriented design techniques are often credited with the rise of iterative techniques, it is not only object-oriented projects that are breaking with waterfall orthodoxy in contemporary software development.

Because software was a natural outgrowth of computer hardware, it was natural for software development process models to grow from the industrial processes that supported hardware development. Many software organizations are tied to hardware vendors, and continue to feel the influence of the mature process cultures supporting their associated disciplines, such as hardware development or general business management [Gabriel 1994]. That legacy has left its mark on the contemporary prevailing practices in software development process. Waterfall development is reminiscent of mechanical assembly line processes, which are very unlike the creative processes that dominate contemporary software. However, waterfall has been unfashionable in progressive software development circles for some time now. We set out to provide a model that accommodated the changes in technology, the market, and the work force that have unfolded over the past decade. To the perspectives afforded by industrial techniques, we add a sociological perspective.

Section 2 describes how increased interest in software development process in the late 1980s motivated us to study organization structure. Section 2.1 tells why current process study methods must be supplemented with additional techniques, and in particular why empirical studies are important. Section 2.2 describes the research technique we used to collect empirical process data. In section 3, we describe the visualization techniques we used to explore patterns of organizational structure, and introduce the tie between our work and sociometric modeling. Section 4.1 presents

the patterns we found to be pervasive in most software development cultures, and section 4.2 focuses on the patterns for ultra-high productivity organizations that we call *hyperproductive organizations*. We deepened our organizational models with sociometric data and other analytical studies; we present some of the more interesting results in section 5. Our research culminated in a collection of development principles that seem necessary to build a hyperproductive organization; these are introduced in section 6. Section 7 proposes further sociometric experiments to refine these case study results.

2. Study and self-study of organizations

We launched our work on process and organization as ISO 9000, STD 2167a and other standards broadened their influence on software development in the late 1980s and early 1990s. These standards focused on process reproducibility, striving to reduce organizational performance variation more than on raising the mean. Such baselining is important to the quality techniques popularized by W. Edwards Deming [1986] based on statistical process control. A process can be improved if it can be understood; it can be understood only if it has a consistent structure [Sengé 1990]; its structure can be consistent only after the first steps of process improvement have reduced process variability. We found that the process culture in most contemporary organizations has a strong focus on process documentation, but what is documented is often distant from day-to-day practice. The process cultures often ignored important variations in organizational behavior that are key to dealing with market uncertainties, or the uncertainties that arise in any process rooted in human intellect.

2.1. *Shortcomings of the state of the art*

Most process-intensive organizations look to a process specification document as the final word on development activities. We noted three problems with this approach in practice: lack of empirical conformance between practice and process specifications; incompleteness of process models; and inability to capture long-term stable process abstractions. Many processes exhibited such broad variation in behavior that it was difficult for process specifiers to agree on a process that represented the "typical" scenario. Many organizations informally built process specifications from anecdotal process experience instead of driving the baseline process model with empirical models and data. Many organizations we studied captured an "ideal" specification instead of capturing empirical practices [Coplien *et al.* 1993]; organizations used those specifications as a baseline for improvement despite this mismatch. Because many process specification models were divorced from empirical practice, nothing forced development practice into statistical control.

Second, process models are often incomplete and inconsistent. Most process models focus on the task and event perspective, leaving artifacts, roles, actors and

agents as secondary abstractions. Much of this task perspective is driven by a preoccupation with interval prediction and reduction on one hand (one manages overall interval by focusing on individual intervals) and quality on the other (methodical reviews form obvious task/event benchmarks). Task models fit well the waterfall-based development model that is predominate in most development cultures. Furthermore, task models hold up the promise of process automation (e.g., [Krishnamurthy and Rosenblum 1991]). We note that the disconnect between process tasks and the artifacts they produce continues to plague most of the organizations we work with today.

Third, many organizations built their process improvement programs around the task or event dimension of process. Well-understood processes (like bug report flow) often can be regularized, but the core processes of architecture, design, implementation and validation are poorly understood from a task perspective. We have found that task ordering changes rapidly in a high-technology development organization, so it can't be counted on as a stable component of process structure. One large organization we studied surveyed its developers and found that 80% of them were working under officially granted process waivers instead of the official common process, largely because the project's process standard didn't capture the essential, stable structure of the process. One reason that task chain models don't capture the stable structure is because of the high degree of concurrency present in modern software development. It is interesting to note that iterative and incremental design cultures were demonstrating success at about the same time that process consciousness was growing. Project managers still find it difficult to reconcile iterative and incremental techniques with process standards that prescribed process steps [Coplien *et al.* 1993]. Many organizations we studied exhibited concurrent engineering practices, where requirements, design, and implementation activities proceeded in parallel [Hartley 1992]. Few organizations *intentionally* apply concurrent engineering. In fact, many organizations using concurrent engineering (as we discovered empirically) remained stalwart about the accuracy of their waterfall design methods (as stipulated in project process documents). We felt that a role-based model would be a better match for these concurrent engineering organizations than would models based on tasks and events.

We wanted to adopt process formalisms that would allow us to compare iterative processes with traditional waterfall models; this meant going beyond task and event models. While many aspects of process might be automatable, we found that productive processes emphasized the creative value added by the people in the process. In general, this suggested that we should study many dimensions of process: artifacts, organizational roles and structure, personal skill sets, and many other factors. Together, these diverse properties define a *process architecture*. Architecture is a partitioning of a system that results from applying a set of partitioning principles, together with the relationship between the parts resulting from that partitioning. Our resources didn't allow us to study all of these at once, we decided to focus on organizational structure, to balance the investment most organizations had made in task models. The industry has a fascination with the relationship between development organizations and the software they create [Fraser *et al.* 1994a and b], so we felt there would be interest in such research.

The organizations we studied didn't necessarily correspond to formal organizational structures, but arose as communities of interest develop within a project. The "real" organizations in any culture can be defined in terms of coupling between actors or roles, brought together by a common interest or objective. Such organizations are called *instrumental organizations*, and should be distinguished from the formal organization structure. An instrumental organization is the "instrument which regulates organizational behaviour" [Swieringa and Wierdsma 1992, p. 10]. These two structures line up in some organizations; see the studies by Swieringa and Wierdsma [1992].

2.2. *The CRC card methodology*

We set out to build instrumental organizational models from first-hand accounts [Cain and Coplien 1993]. Since process works at the level of the engineers doing the day-to-day design, coding, and fire-fighting, why not build the models from their perspective? We chose CRC cards as the tool we would use to analyze organizations. CRC cards (which stands for classes, responsibilities, and collaborators) had been developed as a software design tool by Beck and Cunningham to support their work on software architecture and implementation in the mid-1980s [Beck 1991]. They fit our needs in several respects. First, they support a highly participatory information gathering technique – something that would help us get at empirical behavior. Second, they made it possible to gather data in a group setting. Role-play is a powerful technique to help people recall past events, particularly in the company of the original players. Sociometric research by Bernard, Killworth *et al.* has shown that informant accuracy is less than 50% when people report individually on their interactions [Wasserman and Faust 1994, p. 57]. Bringing group members together helps recall corporate memory. It is also an opportunity to plant the seeds of *group learning* [Sengé 1990], as we will discuss later.

Third, CRC cards were a good "fit" for the domain we were studying. Each card could be used to model an organizational role; the "responsibilities" captured the responsibilities of each role to the organization; the "collaborations" captured the dependency relationships. At this early juncture, we were naive about social network theory and sociometric diagrams, but we would find that the CRC model would serve us well to support social network formalisms.

Fourth, CRC cards balanced important aspects of several techniques commonly used in social network data gathering [Wasserman and Faust 1994, p. 19, p. 44]. There are many different flavors of modeling units. Examples of modeling units include actors, dyads, triads, and others. Our primary modeling unit was roles, a generalization of related actor responsibilities. In dyadic data gathering, each actor is asked about their interactions with other actors. In triadic data gathering, one actor offers an opinion on how a second actor interacts with a third. The CRC modeling technique focuses on dyadic data by helping role actors focus on their interactions with others. However, group discussions led to the collection of triadic data, particularly

for controversial or problematic interactions. It was possible to gather data from an entire organization in one or two sessions of a few hours each.

Sociometric modeling admits several varieties of network relationship data. During the CRC interview, we collected only *dichotomous* network data: in other words, a relation either exists between two roles, or it doesn't. We take care to capture *directed* lines to support studies of information flow. A directed line is called an arc, and a graph of arcs is called a *digraph*. Participants annotate the arcs at the end of the interview, giving them strengths so they become *valued* arcs.

CRC cards had some unanticipated benefits as well. They can be used as a therapy session that helps an organization introspect about itself in real time. Our CRC depositions usually served as mirrors in which organizations could see themselves in a new light. As such, the data gathering technique itself played out the sociodrama and laid the seeds of group therapy.

CRC cards have drawbacks related to groupthink, consistency, and granularity. Perhaps the most serious problem is the opportunity for *groupthink*: the tendency for a group to fall into modes of social conformity [Janis 1971]. Most of the organizations we visited had a diverse collection of strong personalities that avoided many of the problems of self-censorship found in organizations dominated by groupthink. We avoided the problems of "mindguards" (those who protect the power holders in the organization from painful truths) by asking that the subject organizations not send process professionals from their organization, since they usually perform a policing function and frown on departures from stipulated practice. Most groups viewed the exercise as an opportunity to help their self-improvement efforts, and didn't seem to be blind-sided by illusions of invulnerability. We observed rationalization-based groupthink in some groups that participated in these studies because they perceived an allied group – such as the organization that created or managed their development process – as an "enemy." While we feel these factors would have affected our results if we were explicitly looking for process compliance, we don't feel that it affected our models of role relationships within these organizations. For more on groupthink, see Janis [1971].

The second problem is consistency. Each group has its own culture that colors the meaning of common role names: Is it fair to compare the "Developer" role in a startup company to the role of the same name in a legacy organization? Is there a commonly understood meaning for "role" itself? Different organizations produce different products that solve problems of widely varying difficulty: Is it fair to compare otherwise similar teams if one produces aerospace software and the other produces biomedical engineering control software? Such problems plague most software studies; the number of control variables is large.

Granularity is another issue. A complete understanding of process incorporates roles, actors, artifacts, and other dimensions; we are focusing on roles. It is difficult to define a role formally (we "define" them in terms of their responsibilities); roles may overlap, or may exhibit complex mapping to actors. The Pasteur tools (section 3) allow us to combine roles, which helps us evaluate some actor-to-role mappings. The general problems of granularity and mapping remain research issues.

We have used CRC cards to gather data from about forty organizations, almost all of which serve the software industry. We focused on large system development efforts, including development organizations in AT&T and other telecommunications companies, companies producing software development environment products, aerospace organizations, and medical software development. We also have data points from areas as diverse as government administration projects and consumer software. Most of these have been "software development organizations:" the folks who design, implement, and test software. We have a smaller sampling of organizations that interface the market to the development organization, and which perform other assorted functions.

3. Visual analysis techniques

We collected the CRC data in a database, and created an environment called Pasteur [Cain and Coplien 1993] to analyze the data. The data were stored as a digraph representing a social network. Each node in the graph corresponds to an organizational role as characterized by a CRC card. Each arc in the graph corresponds to a collaboration between roles, starting from the role that initiates a collaboration and terminating on the "helping" role of the collaboration. Subjects in the organizational studies assign a weighting value to each arc to express how dependent one role is on the other with respect to the corresponding interaction.

Pasteur supports a variety of network data visualization techniques. The visualization techniques rely on graphical placement algorithms, each of which accentuates different organizational characteristics. The technique we used most often is a natural force-based placement technique. The technique employs a simple relaxation algorithm:

1. All nodes are assigned random coordinates on a plane.
2. A repelling force is set up between all pairs of nodes, following an inverse square law.
3. Arcs exert an attracting force between the nodes they connect; the stronger the interaction between a pair of nodes, the stronger the force.
4. The graph reaches a stable state when all the nodes migrate to positions where their forces balance.

There are other fine points of the algorithm that avoid anomalous "cornering" of nodes that suffer an unfortunate initial placement. This algorithm creates a geographic representation of an organization's interaction graph in two-dimensional space (so far, we have not resorted to multi-dimensional scaling). Pasteur supports other placement algorithms as well, such as two-dimensional hierarchies (created by a topological sort that employs heuristic cycle-breaking techniques) and automatic graph partitioning around selected "seed" roles. The framework accommodates customized rendering

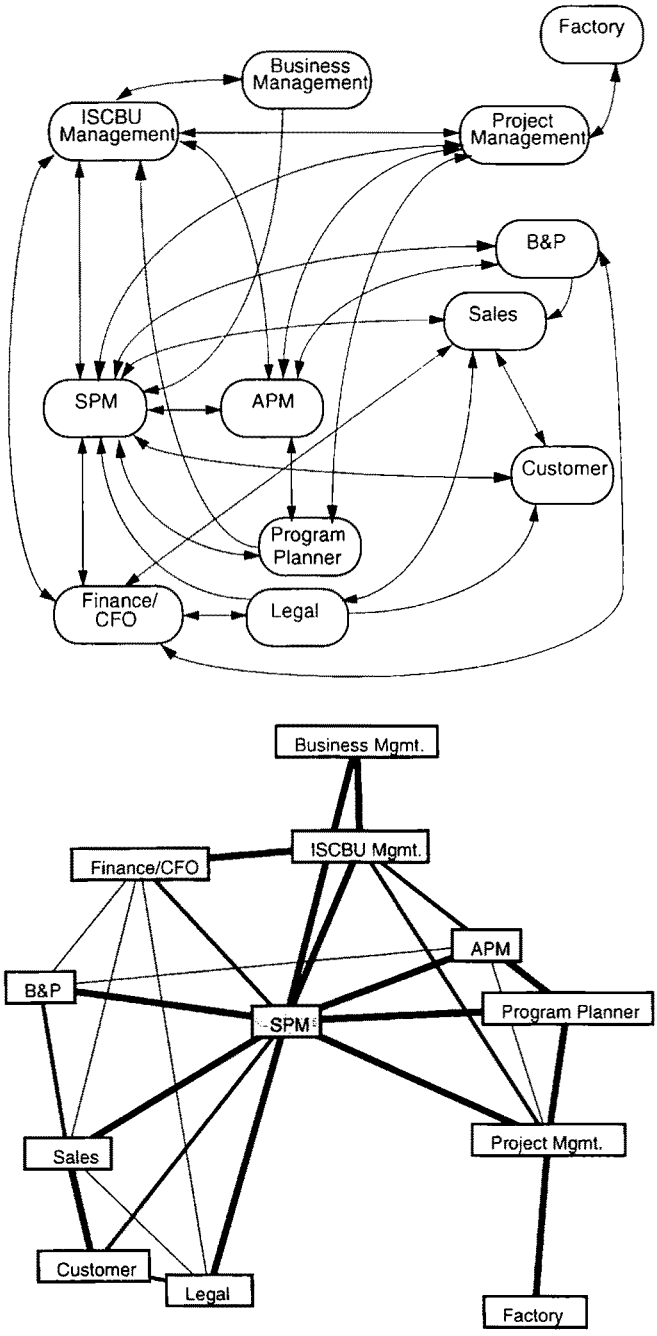


Figure 1(a: top; b: bottom). A traditional social network, and a Pasteur natural force-based rendering of the same organization. Figure 1a is a sociometric network of a product management process. Figure 1b is generated by Pasteur's natural force-based relaxation placement algorithm. (The Pasteur diagram is from [Coplien and Schmidt 1995, p. 227].)

techniques for individual experiments, using a rich programming environment based on the experimental languages GIL and Romana-I [Brown *et al.* 1986].

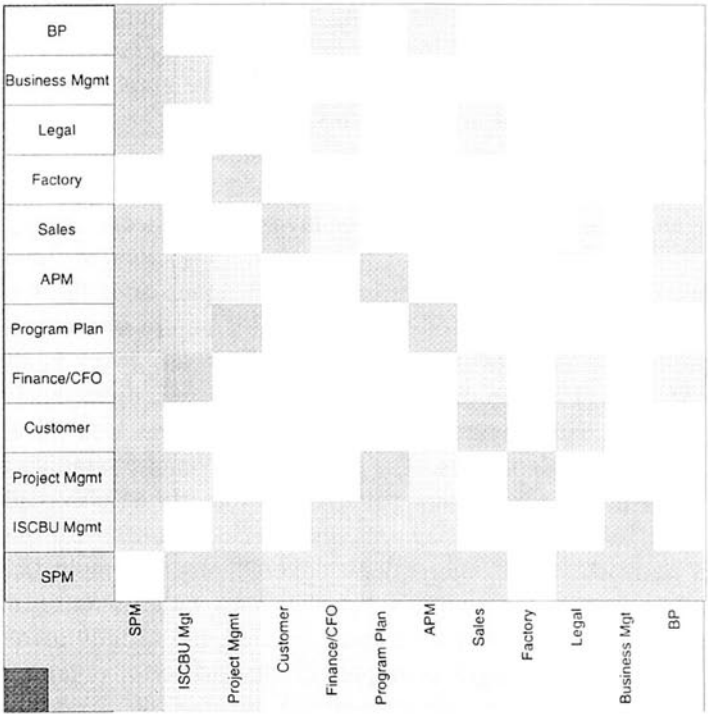
Pasteur displays the graph on either an interactive graphical display or a color printer. Nodes are color-coded according to their intensity of interaction with neighboring nodes, relative to the organization as a whole. The graphical interface allows researchers to directly interact with the model. A user can interactively remove nodes or arcs, create annotations, merge graphs, or invoke any placement algorithm.

While analytical techniques can be applied to sociometric data to discover cliques, cutsets, cutpoints, and the like, visual techniques offer the researcher quick intuitive insights into many facets of organizational structure at once. Social psychologists use a pictoral social network called a *sociogram*, a network analysis technique developed by Moreno [1934] in the 1930s. Like our visualizations, sociograms graphically depict network data (Figure 1). However, sociograms lack the spatial cues of the visualized placement algorithms. The placement techniques amplify the sociogram data, presenting it in a format where patterns can directly be observed by the organizational analyst. We call these diagrams *amplified sociograms* for that reason. The Pasteur social network visualizations depict interactions as simple lines rather than directed arcs, focusing on the coupling between roles rather than on the flow of information. Few human interactions are truly directed, but usually involve "dialogue" or "meetings:" the Pasteur diagrams emphasize that aspect of organization structure. Interaction grids (below) capture the structure of directed interactions.

We also employ interaction grids, a technique inspired by the work of Church and Helfman at AT&T Bell Laboratories [Church and Helfman 1993]. These diagrams are reminiscent of the structure of sociomatrices: a square matrix whose columns are the roles that initiate collaborations, and whose rows are the roles receiving the collaborations. The sociomatrix – and hence the interaction grid – has information that is isomorphic to that in the social network diagram (Figure 2). The sociomatrix and interaction grid communicate patterns of *directed* interactions, something that is present but difficult to read in sociograms, and which is missing entirely in the force-based network visualizations. Shading makes it easier to recognize patterns in the interaction grids than can be found in the numbers of the sociomatrix. The ordinate axis of the interaction grid enumerates roles that initiate interactions; the coordinate axis enumerates (the same) roles as they are the targets of interactions.

Visualizations are an intuitive presentation of more formal underlying concepts. We can analytically measure the centrality of an organization using several formal definitions. By standard sociometric measures, the centrality of the organization in Figure 1 is 2.2. Which do you find more compelling: the number or the picture? Instead of explaining sociometric vocabulary to team members (particularly to managers), we appeal to their intuition and imagination with these organizational portraits.

These visualizations support the second phase of introspection by the subject organizations: they are data that help the organization face and understand its problems. The location of key roles in the diagram usually confirms the development team's expectations or helps team members explain exceptional or problematic behavior. For



	SPM	ISCBU Mgmt	Project Mgmt	Customer	Finance/CFO	Program Plan	APM	Sales	Factory	Legal	Business Mgmt	B&P
B&P	3				1		1					
Business Mgmt	3	2										
Legal	3				1			1				
Factory			3									
Sales	3			3	1					1		2
APM	3	2	1			3						1
Program Plan	3	2	3				3					
Finance/CFO	2	3						1		1		1
Customer	2							3		3		
Project Mgmt	3	2				3	2		3			
ISCBU Mgmt	2		2		2	2	2				3	
SPM		3	3	2	2	3	3	3		3	3	3

Figure 2(a: top; b: bottom). An interaction grid (Figure 2a) and the corresponding sociomatrix (Figure 2b).

example, one organization immediately noticed the remoteness of its architectural role in the social network diagram, and explained that was one of the reasons for the lack of product focus in the organization. A crucial point here is that an individual sociogram or interaction grid alone doesn't pinpoint organizational problems; it is a mirror in which team members can see themselves better, and thereby better understand their problems.

We collected pictures into a catalogue and categorized them. Following Gamma's [1992] studies of recurring, re-used patterns of code in software systems, we wanted to find the recurring patterns of communication in software organizations. One goal of the study was to collect and catalog typical recurring patterns from a wide spectrum of organizations: a social anthropology of software development. Such studies would form an empirical basis for models of contemporary software development as it really happens, as opposed to ideal models built from first principles.

We were particularly interested in finding the patterns peculiar to successful, productive organizations, to investigate whether any organizational "shapes" correlated to productivity or success. We quantified "successful" or "productive" only informally or through very coarse-grained metrics. For example: like everyone else, we used thousands of non-commentary lines of code (KNCSL) per staff-month as raw productivity data, but we thought of these data in terms of metrics like $\log_{10}$(KNCSL)/staff-month. We also took note of remarkably short development intervals. Patterns did emerge over time; we discuss those in the next section.

At about the same time, we started extracting sociometric parameters from the sociograms. These parameters include standard sociometric data such as graph density and graph centrality. Some of these data correlated well to productive organizations, and some of the data are interesting in their own right. These data analyses are discussed in section 5.

4. Recurring themes in organizational patterns

We learned much from the recurring patterns in the process visualizations. These pictures gave us insight into the self-organizing structures that recur in software development organizations in a wide variety of development organizations. In section 4.1, we explore the patterns that recurred in most organizations we studied. Section 4.2 presents the patterns we found in highly productive organizations.

4.1. Patterns common to most organizations

There is no such thing as a self-contained design process. Most organizations we studied claimed to have a design process. They could point to a process description, complete with inputs, outputs, internal review benchmarks, and activities. These documents were designed to support the expectations that organizations held for task-based process formulations, as discussed in section 2.1. However, the CRC interviews showed that most developers run design and implementation processes in parallel. The

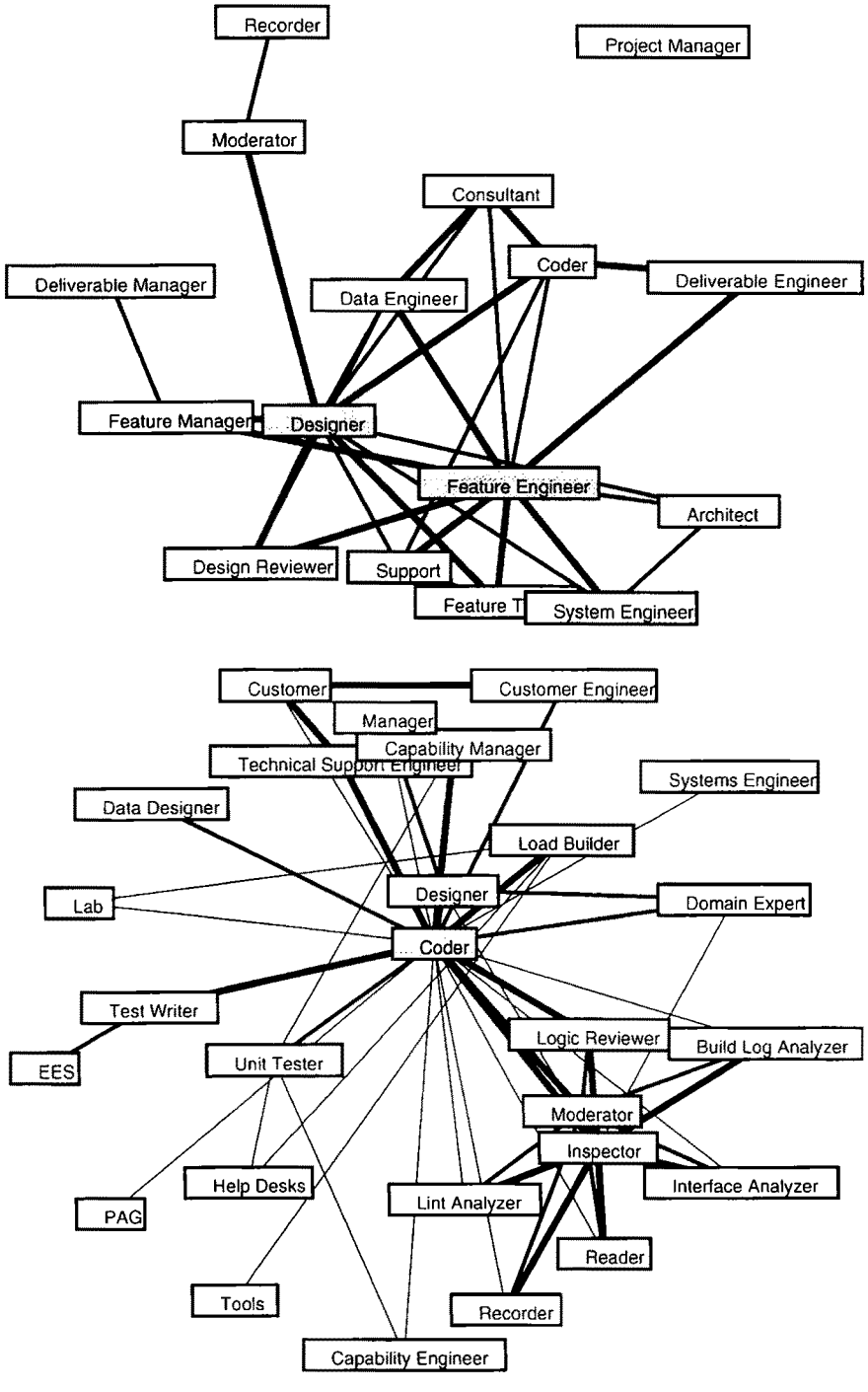


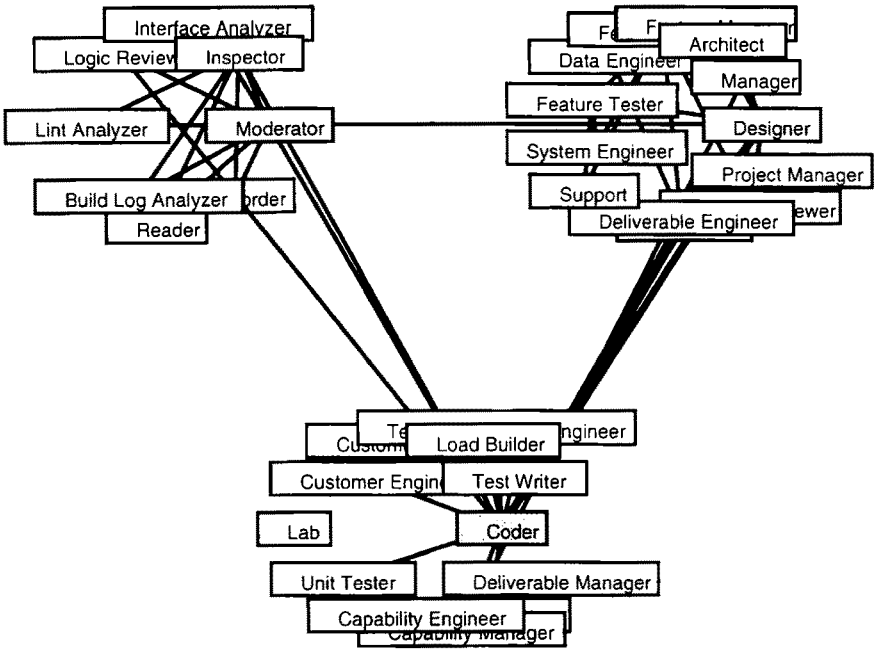
Figure 3(a: top; b: bottom). Design process (Figure 3a) and coding process (Figure 3b) from a typica development organization.

designer and coder roles are often filled by the same individual. A typical development scenario would find this person sitting at a multi-window terminal, editing the design document in one window, and the corresponding code in another. (Multi-window terminals are perhaps one of the unheralded enablers of iterative development!)

This parallelism suggests that the instrumental organizations for design and coding (Figure 3) co-exist in time, and that they can be combined into a single graph. The graphs are combined after establishing isomorphic mappings between roles or groups of roles in one graph to the corresponding roles or groups of roles in the other. We can analyze that graph for the cliques – the instrumental organizations – that exist inside it. The clique model is created using a semi-interactive seed-based partitioning algorithm similar to that used to partition integrated circuits onto circuit boards. The analyst selects seed roles; Pasteur factors the graph into sub-graphs built around these roles. The algorithm attempts to optimize cohesion within cliques while minimizing coupling between them. The result is the graph of Figure 4a.

Three cliques can be factored from the combined organizations. One clique contains roles that participate in reviews (both design and code reviews). The clique on the upper right contains most of the design roles, and the clique at the bottom center contains most of the coding roles. The most strongly coupled pair of roles in the graph are the Designer in the design clique, and the Coder in the coding clique. In fact, the coupling between these cliques is strong. We can measure the coupling between these two clusters using coupling and cohesion metrics. If we sum the weighted interactions *internal* to the coding clique, we get a value of 25. The sum of all *external* interactions from the coding clique is 23; of this total, 16 owe to interactions with the design clique. Two processes with this degree of coupling don't seem like independent processes – particularly if their activities proceed in parallel, as our interviews suggest. Furthermore, the mutual interference between these processes suggests that neither can be baselined independently: any process improvement effort must consider both processes together.

Some projects exhibit a good separation between their design and implementation processes. These projects tend to represent development in well-understood domains, such as database design. The amplified sociograms for these organizations tended to exhibit low graph density, and usually had a centralized structure we call "hub-spoke and rim." Rim (perimeter) roles represent steps in a rigorously orchestrated time-serial process, as in the organization of Figure 1. "Design" in these processes consists largely of parameterization or mechanical structuring. As such, these processes might be able to automate some tasks. Sociologically, we can categorize such organizations as *high-context cultures*. The organization has built up shared models, shared knowledge, and a shared vocabulary over time, a shared world-view that obviates the need for close coordination between roles. Most projects address problem domains where design remains a creative activity. These are *low-context cultures*, where most new development initiatives foray into unchartered territory to develop features not easily extrapolated from current behavior. The process must be flexible to accommodate unforeseen configurations and needs in these domains; it is these domains that are well served by iterative development.



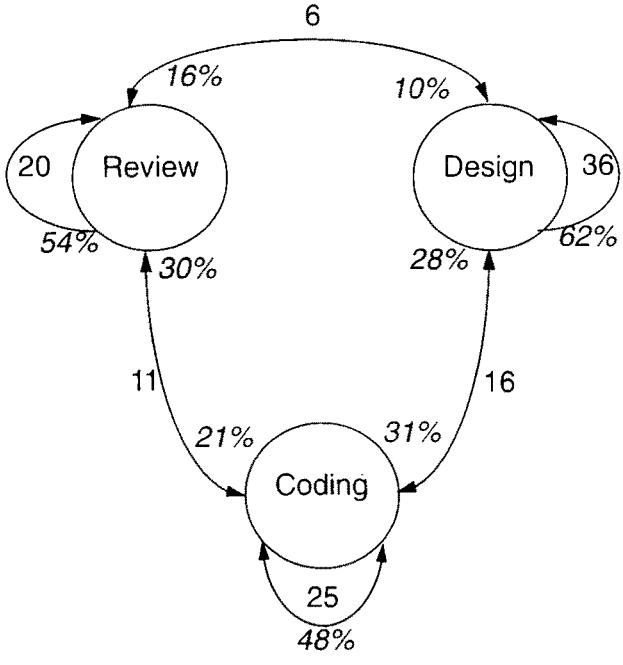


Figure 4(a: top; b: bottom). The graphs of Figure 3 combined, showing the clique partitioning and coupling between cliques. Figure 4a is from [Coplien and Schmidt 1995, p. 218].

The coder is a cutpoint of most design and implementation processes. We noted above that the design and coding processes are tightly coupled to each other. Most process models admit coupling between processes through well-defined artifacts (like documents) or events (like meetings). We found that in practice, most processes are tied together because they share one or more roles. In particular, the Coder or Developer role coupled the activities of most processes. (Coder and Developer are two different titles that often designate the same activities, though the organizations using the Developer designation were more conscious that design and coding are interleaved tasks.) We found the Developer role to have the highest *popularity* rank in almost all the processes we studied; popularity is a measure of the coupling from a single role to remaining roles in the social network. In most of these organizations, the Developer also had the highest *prestige* rating; prestige is a measure of how often other roles initiate interactions with a given role. In both these patterns, the Developer role tends toward the middle of the natural force-based social network diagrams. Such characteristics are to be expected of design and coding processes, but we found similar patterns in some organizations' software integration process and in a test lab support process.

Let's return to the combined design and coding processes treated above. We find that the Coder role is almost a cutpoint between the coding and design cliques; it is a complete cutpoint between the coding and review cliques.

Other patterns. A host of other patterns emerged as our process catalog grew, but they are too numerous to enumerate or elaborate here. These patterns include:

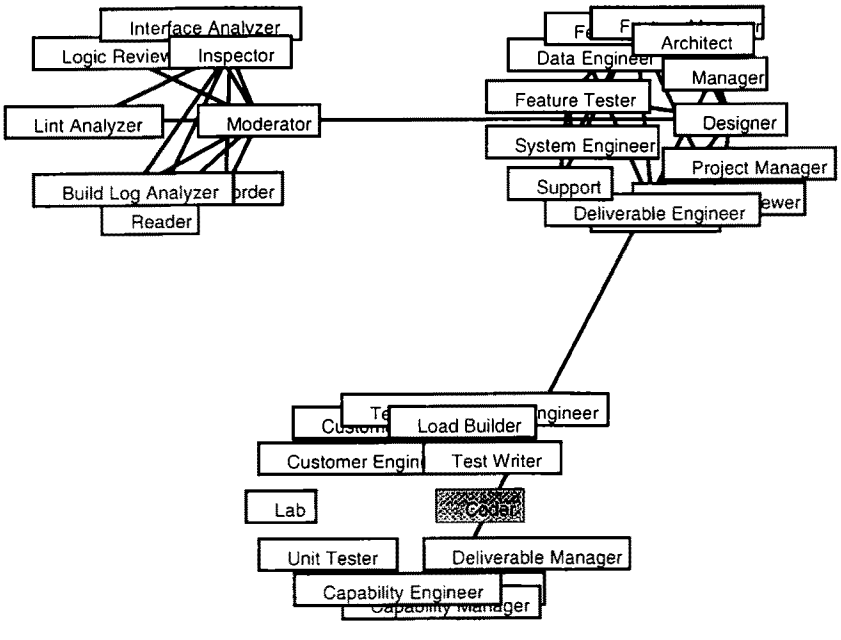


Figure 5. We remove the Coder from Figure 4.

- *Managers intervene in emergencies*: managers move toward the center of the graph in high-stress or urgent situations, but otherwise rarely exhibit a high prestige rating;
- *Coupling decreases latency*: optimal service processes maintain short inter-node walks with a minimum of arcs;
- *Hub-spoke-and-rim*: Mature, well-understood processes (like bug report management or high-level product management) usually appeared as spoked wheels, which is consistent with a regular, time-ordered process (see Figure 1).

4.2. *The patterns of hyperproductive organizations*

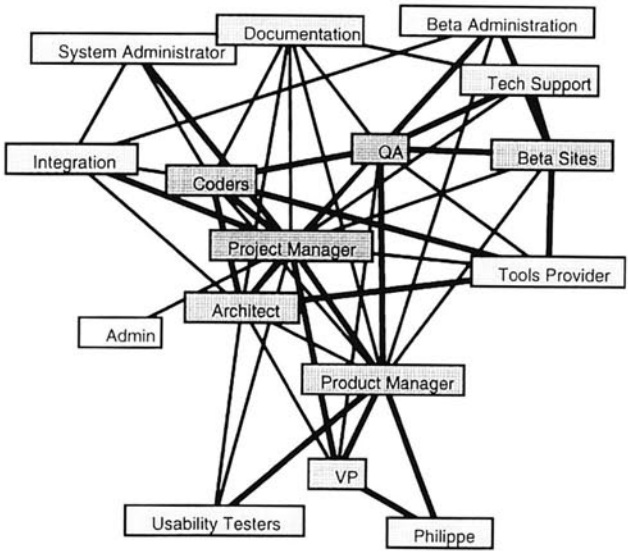
We found a few organizations that stood far above the rest in productivity, quality, and interval. We often found all three of these characteristics together in the same organizations. All of these are necessarily subjective; there are few widely accepted metrics that would prevent the problems of comparing apples with oranges. We tried to overcome the subjectivity by looking at extreme values. For example, we don't distinguish between organizations that produce the same code volume per person per unit time within a factor of two or three. What interests us is orders of magnitude. Most of the hyperproductive organizations we studied produced on the order of 10^3 lines of code per programmer per month, which is easy to distinguish from typical organizations that produce on the order of 10^2 lines [Gabriel 1994]. By focusing on exponential differences, we felt safe in ignoring the noise of programming language differences and other control variables that are difficult to hold constant in such comparisons.

We were pleased to find recurring patterns in highly productive organizations – patterns that distinguished them from average or dysfunctional organizations.

Distribute work evenly. The members of a highly productive team work as a unit, dividing the work load evenly among themselves. The combined outdegree and indegree of each node is one measure of its work load. We find that highly productive organizations have a small standard deviation in the distribution of work load across roles. Such organizations are said to have *low centrality*.

Figure 6 depicts the Pasteur natural force-based picture and interaction grid for one of the most highly productive organizations we've studied, the Quattro Pro for Windows development at Borland International [Coplien 1994]. Both these views of the organization show that communication was evenly distributed among roles. (The effect is particularly pronounced when one compares these graphs to most of the other graphs in our catalog.) We explore more analytic models for this characteristic in section 5 of this paper.

Work flows inward. In section 4.1, we noted that the developer was at the core of most organizations. We find the developer near the core of the hyperproductive organizations, but a high prestige value for the developer isn't sufficient for success.



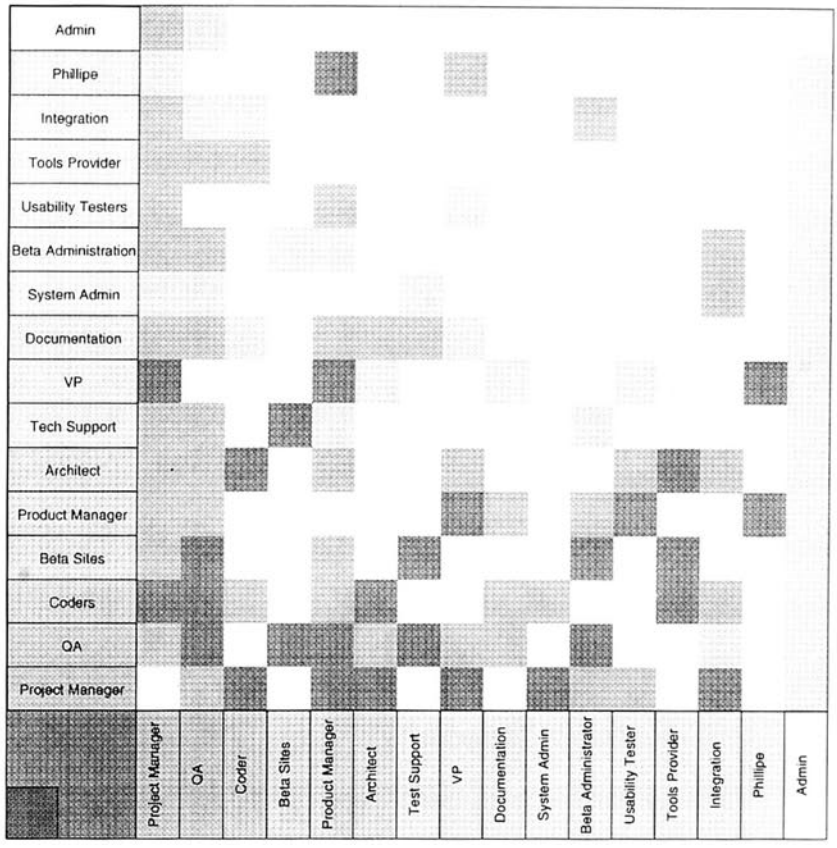
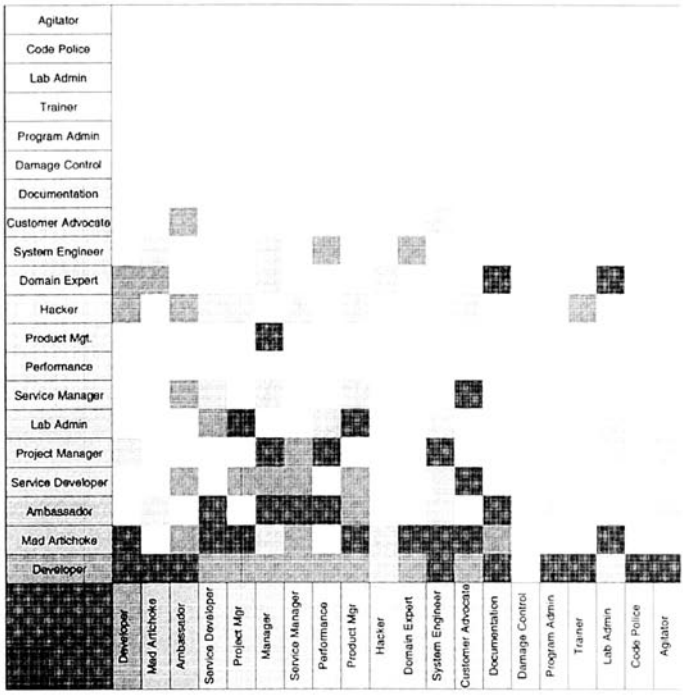


Figure 6(a: top; b: bottom). Distribute work evenly. From [Coplien and Schmidt 1995, p. 209, p. 219].



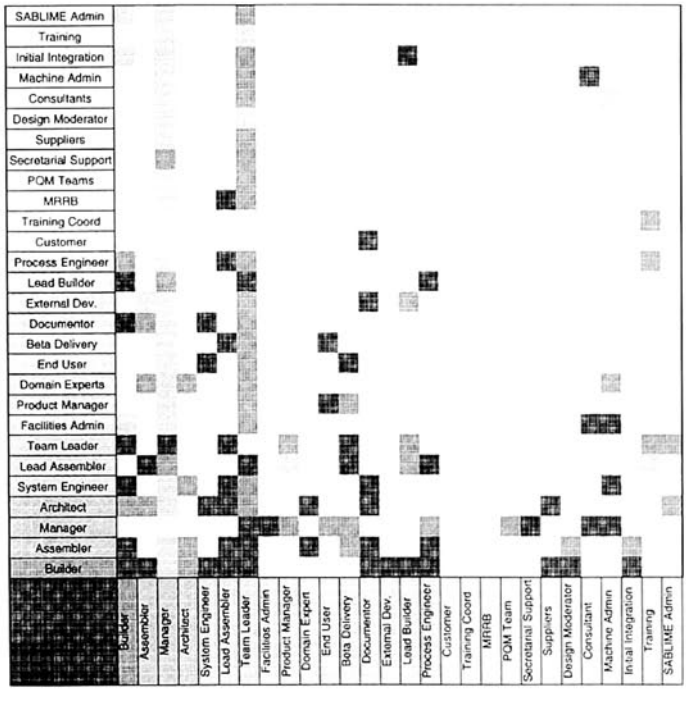


Figure 7(a: top; b: bottom). Work flows inward. From [Coplien and Schmidt 1995, pp. 222–223].

There is a strong correlation between productive organizations and a strong indegree for the developer role. Organizations with interaction grids like Figure 7a tend to be more productive than those that are similar to Figure 7b. These groups tend to focus on the roles that directly add the most value to the organization's product. (The pattern isn't relevant to service organizations.)

Some organizations exhibit much higher prestige values for management functions than for those who directly add value to the product. Collaborations in these organizations almost always flows *from* the manager in the center *to* other roles in the organization. Such organizations usually exhibit average or poor performance. We find these map closely onto the *A1-style organizations* of Ouchi [1981].

A chief surgical team would be another organization in this family of patterns, with a strong architectural presence at its core, directing work to an implementation support team. This pattern is distinct from either of the above two patterns, because it has a strong technical presence at the center directing technical support nearer the edges. In organizations where work flows inward, the technical roles are at the center, surrounded by support roles that drive them. A1-style organizations are like chief surgical teams, except that the central role is more managerial than technical.

Other patterns. Other important patterns of productive organizations include:

- *Architect controls product*: the architect role has a high popularity rating in the social network;
- *Engage QA*: Quality assurance is well-coupled to the organization as a whole;
- *Engage customer*: Models for most successful organizations exhibited close customer engagement, while less productive organizations often omitted the customer role altogether.

5. Analytical properties

We derived graph data from the information gathered in our interviews, and analyzed it using statistical and sociometric techniques. In addition to standard sociometric data, such as graph density, we explored other parameters as well. This section summarizes two interesting results from these analyses.

5.1. Trends in graph density

A social network of n nodes has $n \times (n-1)/2$ possible links. One might expect the number of communication paths to grow roughly as the square of the number of roles in the corresponding organization. However, our data indicate a *linear* growth in communication paths with the number of roles (Figure 8). What's more, this is true independent of the size of the organization: the addition of any role to any organization has a remarkably predictable and constant effect on the number of collaborations in a process. This supports some remarkable conclusions. It's as if each role brings a need

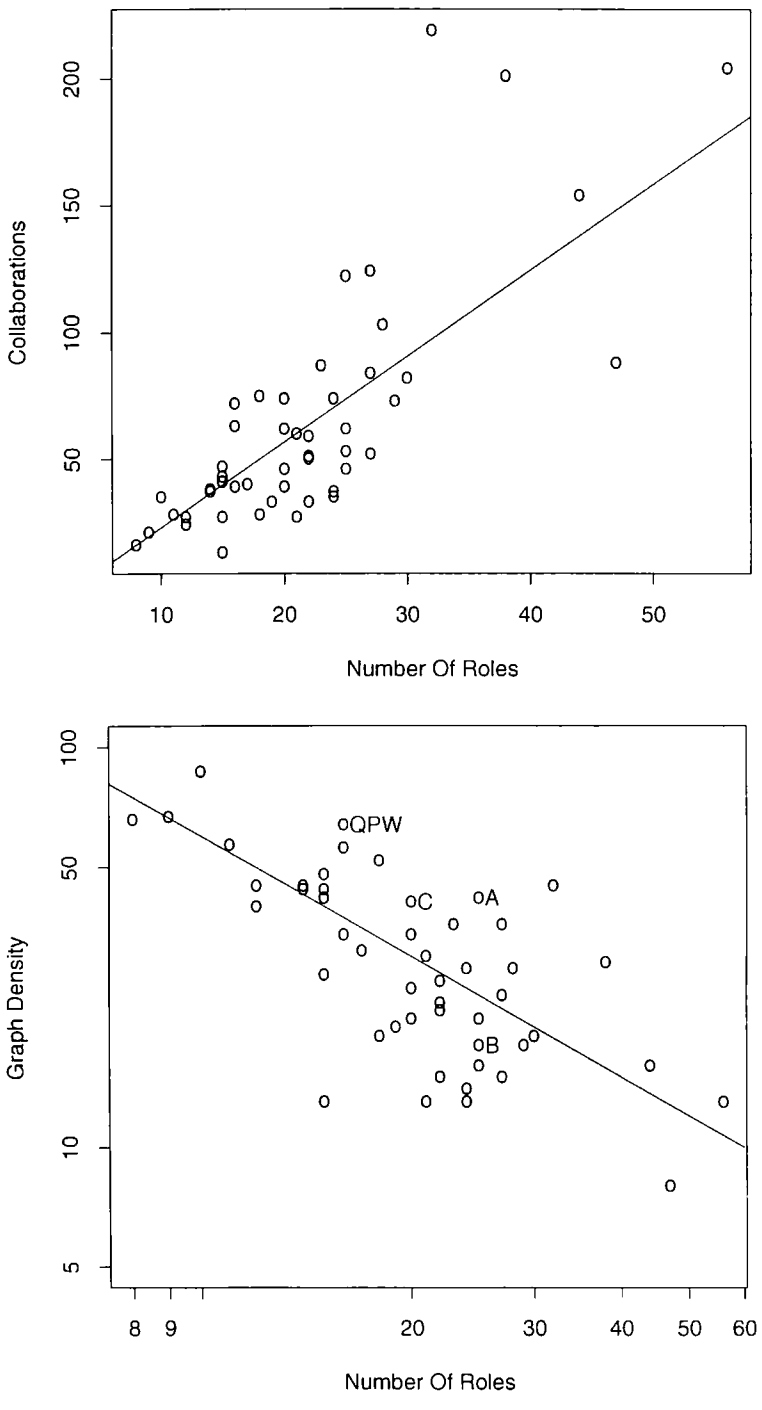


Figure 8(a: top; b: bottom). Collaborations and graph density as functions of the number of roles. Both from [Harrison and Coplien 1996].

for a critical mass of social contacts. Even if the domain knowledge is spread across a larger number of contacts, most newcomers must expend extra energy to expand their contacts above some fixed number. The slope of the line is about 3 : 1, consistent with the number of collaborations per role we find across organizations, most of which had a coupling/role ratio of between 3 and 4 (though the distribution has a tail to the right that raises the mean to 5.62, with a variance of 2.43). Coupling per role is almost completely independent of the number of roles (the correlation coefficient is about 0.07).

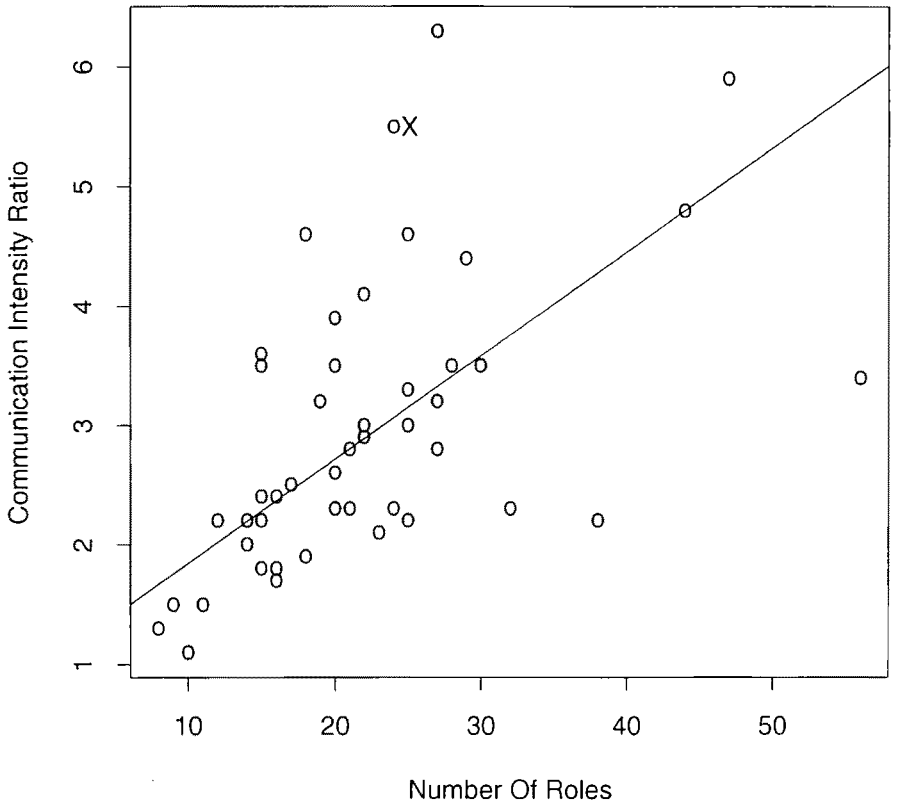
The graph of Figure 8b is a log-log plot of network density as a function of the number of roles. The relation *density* = *roles/6* is superimposed on the graph, a good fit for the data. To interpolate this relation naively, we note that for values of *roles* less than 6, the graph is fully connected. This is our own contribution to the magic number 7 plus or minus 2. In any case, the moral of the story is: keep an organization small (minimize the number of actors) and simple (minimize the number of roles). The conclusion should come as no surprise to experienced project members, but these models help explain why small projects work better than large ones, and offer some parameters that help quantify the effects of size.

Some organizations can beat the game and maintain strong collaborations in spite of their size. Those organizations lie above the line in the collaboration graph. Borland's QPW (Quattro Pro for Windows) is one such organization, as indicated on the graph. For example, the point labelled "A" is a productive support product development organization in the aerospace industry; "C" is the second most productive organization we found; "QPW", the Borland Quattro Pro for Windows development [Coplien 1994], is the most productive organization we've found. By contrast the point labelled "B" is a high-tech development shop that was experiencing serious organizational problems at the time the data were collected.

5.2. *Communication intensity ratio*

We developed an organizational metric called the *communication intensity ratio*. This metric measures the ratio of the most popular role's link count to the graph density; it is a rough measure of organizational cohesion. We plot this ratio in Figure 9 as a function of the number of roles in the organization. Most high-productivity software production organizations exhibit an evenly distributed work load, that is, they have a low communication intensity ratio. However, we have also found troubled organizations with low communication intensity ratios, though they tend to not exhibit the other patterns common to high productivity organizations.

The communication intensity ratio increases as the organization size increases. This is due in part to intuitive mathematics: the distance from the mean to the extremes increases as populations grow. One expects the graph density to decrease as the graph grows, as discussed in the previous section, but it's curious that the intensity of core roles isn't diluted by organizational growth. These core roles are often producer roles – the ones that add value directly to the end product. Since these core roles can't



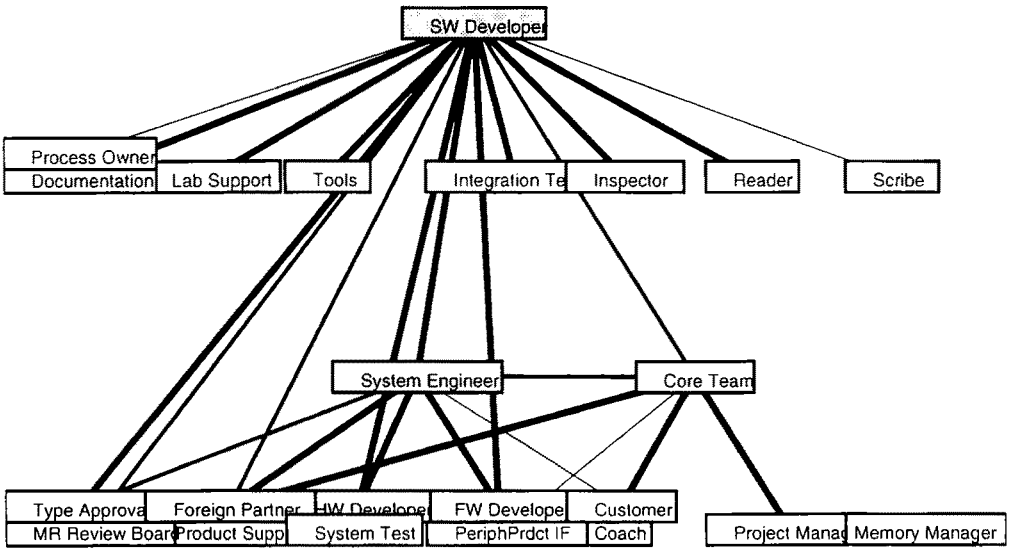


Figure 9(a: top; b: bottom). Communication intensity ratio. Figure 9a is from [Harrison and Coplien 1996].

easily extend their direct influence to all members of the organization, organizations with large communication intensity ratios tend to have a hierarchical structure. We can use the Pasteur tool to extract the hierarchical structure from a social network. The hierarchy of Figure 9b depicts the organization marked by the "X" in Figure 9a. Large organizations exhibit hierarchical structure, but even small organizations may resort to hierarchies when cultural, technical, or geographical forces dictate. Development organizations with hierarchical instrumental structures rarely enjoy the dynamics of a productive, tight-knit team. In fact, Sengé quotes Bohm in noting that "[h]ierarchy is antithetical to dialogue" [Sengé 1990, p. 245].

6. Generative patterns

After working with more than forty organizations, we became good at predicting sociometric values for new organizations we studied, as well as the shapes of their amplified sociograms. This supported our feeling that the deep structures of an organization generated observable emergent behavior in day-to-day practice. Not only had we found recurring patterns of organizational structure in software development organizations – as Gamma did for software – but we could correlate patterns of organizational structure to organizational health and prospects for success. At this point in our research, we had established only that there is a correlation between the structural patterns and organizational success. We had not experimentally verified a cause and effect relationship between them. However, we could explain most of the patterns in terms of widely acknowledged software engineering principles (as the importance of having a knowledgable architect) or common sense (low graph density in a low-context culture bodes poorly for success). We could start talking about these patterns not only in the sense that our eyes perceive recurring regularity in nature (or in a sociogram), but in the sense of a dress pattern: a construct we can use to build and reform organizations to suit our needs.

Sengé notes that organizational change comes about from changing the deep structure of an organization [Sengé 1990, p. 53], but only in an indirect way. Structure (in which process architecture plays a part) affects patterns of behavior, which in turn affect events. Most software development process cultures attempt to address events without regard to the structure of the organization (structure in the sense of affecting patterns of behavior, not in the sense of corporate organizational charts). We felt that by helping organizations build on the deep structures that bode well for success, we could make broad, lasting changes in organizational behavior. This is called *double-loop learning* by Swieringa and Wierdsma [1992]. Double-loop learning requires insights and understanding that support organizational renewal. We can contrast this with the (more common) single-loop learning, which is based on rules that support a system of obligation and permission. Single-loop learning rarely achieves structural improvement, but changes only surface behavior. Double-loop learning is indirect: it is learning how to learn. We watched organizations attain some level of double-loop

learning as they came to grips with their deep structural problems through the diagrams we generated for them (see section 3 above).

Sengé uses the term *generative* to describe the indirect effect of structure on patterns of behavior. The software design community is exploring generative patterns as a way to document problem/solution pairs in software architecture [Gamma *et al.* 1995, Coplien and Schmidt 1995]. That work draws in turn on a school of building architecture based on Alexander's generative patterns [Alexander 1979]. Alexander's patterns are a literary form used to capture key architectural (structural) insights within a system. These patterns help readers interpret system structure through their intuition, common sense, and sense of aesthetics. The form includes a declaration of the context in which the pattern applies, a discussion of the forces at work in the pattern, a description of the structure, and an explanation of how the structure works. We felt that the pattern form used to capture good practices in building architecture and software architecture might be a good vehicle to capture the patterns we observed in outstanding organizations.

Generative patterns not only capture the structures we observed in the visualizations for outstanding organizations in sections 3–5, but expand on these structures to incorporate insights about how the patterns work. We collect the insights in a pattern system that forms a network of recommendations for organizational construction, repair, and maintenance. Most of these recommendations derive from intuitive management practices, social network theory, or organizational psychology. The recommendations don't stand alone, but are borne out in the structures of the successful organizations we studied. We can illustrate with an example, drawn from organizations such as the one behind Figure 6. Like other successful software projects, this one has an architect role with a high prestige value.

Name: Architect Also Implements (or "αρχιτεκτον", to emphasize that the etymology of "architect" as "chief builder" reflects the intent of this pattern).

Problem: Preserving the architectural vision through to implementation.

Context: An organization of Developers that needs strategic technical direction.

Forces: Totalitarian control is viewed by most development teams as a draconian measure. The right information must flow through the right roles.

Solution: Beyond advising and communicating with Developers, Architects should also participate in implementation.

Resulting Context: A development organization that perceives buy-in from the guiding architects, and that can directly avail itself of architectural expertise.

Rationale: The importance of making this pattern explicit arose recently in a project I work with. The architecture team was being assembled across wide geographic boundaries with narrow communication bandwidth between them. Though general architectural responsibilities were identified and the roles were staffed, one group had expectations that architects would also implement code; the other did not. One manager suggests that, on some projects, architects should focus only on the implementation of a common infrastructure, and that the implementation of non-core code should be left solely to the Developer role.

(The original pattern contains additional rationales based on analogies to building architecture; see [Coplien and Schmidt 1995].)

This pattern reflects a well-known principle of good software organizations; our research uncovered this pattern, but did not discover it. We discovered other patterns that are less well-known, at least in software engineering literature. "Work Flows Inward" is one such pattern:

Name: Work Flows Inward.

Problem: Work that adds value directly to the product should be done by authoritarian roles.

Context: An existing organization where the flow of information can be analyzed. The organization exhibits a management pecking order. The developer is already sensitized to market needs through the application of other patterns like "Gatekeeper" [Allen 1977] and "Fire Walls" (a pattern that describes how managers insulate developers from outsiders who feel a need to offer help and suggestions).

Forces: Some centralized control and direction are necessary. During software production, the work bottleneck of a system should be at its center (in the social diagram sense). If the center of the organization generates work more than it does work, organization performance can become unpredictable and sporadic. There must be one place where technical details come together right before implementation, not just in the front-end analysis and requirements processes.

Solution: Work should be generated by customers, filter through supporting roles, and be carried out by implementation experts at the center. You should not put managers at the center of the communication grid: they will become overloaded, and make decisions that don't take day-to-day dynamics into account.

Resulting Context: An organization whose communication grid has more points below the diagonal than above. Managers are freed to make day-to-day business process decisions, and to fill their role to "keep the pests away" from developers (pattern "Fire Walls").

Rationale: Katz and Kahn note that the exercise of control is not a zero-sum game [Katz and Kahn 1978, p. 314]. Organizations run by professional managers tend to have repeatable business processes, but don't seem to reach the same productivity plateaus of organizations run by engineers [Gabriel 1994]. Mackenzie characterizes this pattern using *M-curves* that model the percentage of task processes for each task process law level (planning, directing, and execution) as a function of the classification [Mackenzie 1986].

[Coplien and Schmidt 1995] contains several dozen organization patterns of this form in three pattern systems (those by Coplien, Kerth and Whitenack). Generative patterns like this are to be used as social anthropology tools in reverse. Instead of capturing the patterns of an organization, they attempt to establish patterns in an organization. These deep patterns of structure drive patterns of human behavior, which shape the events of successful development processes.

The mapping from the Gamma patterns of our empirical research to the generative patterns of [Coplien and Schmidt 1995] is not always one-to-one. For example, several generative patterns work together to generate an organization where work is evenly distributed among roles (which is empirically desirable; see Figure 6 above and its associated text). These patterns employ work role titles, shuffling of responsibility ("Move Responsibilities"), geographic collocation ("Organization Follows Location"), and other techniques to achieve an organization like that of Figure 6.

We have seen organizations use these patterns to attack dysfunctional organizational structures and business practices, particularly when the patterns they are used in an indirect, generative way. When we first worked with the organization labelled "B" in Figure 8, group members were able to develop deeper insight into their problems by comparing their own structures to those in the pattern language. There was no systematic program to use the patterns for organizational improvement. However, the patterns became widely read in the group, and pattern names started to appear in the group vocabulary as they discussed organizational restructuring. The patterns provided a shared vision of what was possible, and of what practices were relevant to their specific needs. Today, the organization is a viable and healthy organization; its new culture has precipitated changes in other organizations in the company.

7. Future research

Our work to date has avoided employing controlled experiments pending case study experience that helps us understand the experimental variables better. We now have a set of hypotheses – the generative patterns – that we can evaluate against successful and dysfunctional organizations. The experiment postulates the presence of these patterns in good organizations, and their absence in dysfunctional organizations. We also hope to establish a list of *anti-patterns* [Koenig 1995] that characterize bad practices, and test for their presence in dysfunctional organizations and their absence in outstanding organizations.

We see two major challenges in this research. The first is how to objectively characterize success. Our current success metrics are largely subjective, drawing on many factors such as productivity (KNCSL derivatives), employee satisfaction, and market success (including long-term maintenance flexibility). To capture these measures individually is a challenge. Even for "analytical" notions such as productivity, the debate continues between advocates of KNCSL, function points, and other measures. It is an even greater challenge to combine these elements into a single framework for comparing one organization to another. The second problem is finding suitable subjects for the experiment. We need data from dysfunctional organizations, not many of whom are eager to air their dirty laundry. As one colleague put it, it's like casting a suitable actor for a movie role that calls for an ugly person.

We also need to study how well these patterns might be used to turn around a dysfunctional organization. Do these patterns cause problems of their own? Is it possible to instill these patterns into an organization? What kind of method supports pattern-based process improvement?

8. Conclusion

A sociological perspective on software development organization has proven insightful. If there is one consistent measure of successful organization, it is how well its members maintain relationships through communication. Highly productive organizations are cohesive and usually are de-coupled from other organizations. Communication paths in a good organization are like a woven fabric with no thin spots and no "bunching": communication is even and strong. Good organizations nurture roles that work directly with product artifacts. Other research results can be found in [Coplien and Schmidt 1995]. These patterns are broader and more realistic (because they are empirical) than most institutional process descriptions. Furthermore, they can be used in a generative way to build and improve software production organizations. Our analyses, both as a "mirror" and a benchmarking technique, have helped several organizations improve their development processes.

References

Alexander, C. (1979), *The Timeless Way of Building*, Oxford University Press, New York.

Allen, T. (1977), *Managing the Flow of Technology*, MIT Press, Boston, MA.

Beck, K. (1991), "Think Like an Object," *UNIX Review 9*, 10, 39–43.

Booch, G. (1994), *Object-Oriented Analysis and Design with Applications*, 2nd Edition, Benjamin-Cummings, Redwood City, CA.

Brajkovich, L.F. (1994), *Sources of Social Structure in a Start-up Organization: Work Networks, Work Activities, and Job Status*, North-Holland Elsevier Press, Netherlands.

Brown, D.W., T.A. Burrows and P.M. Zislis (1986), "GOS: A Tool Providing Support for Graphical Human-Machine Interfaces," In *Proc. of the 1986 International Zurich Seminar on Digital Communications*, Zurich.

Cain, B.G. and J.O. Coplien (1993), "A Role-Based Process Modeling Environment," In *Proc. of the Second International Conference on the Software Process*, IEEE Computer Press, Los Alamitos, California, pp. 125–133.

Church, K.W. and J.I. Helfman (1993), "Dotplot: A Program for Exploring Self-Similarity in Millions of Lines of Text and Code," *Journal of Computational and Graphical Statistics* 2, 2, 153–174.

Coplien, J.O., S. Hutz and B. Marykuca (1993), "Iterative Development/OO: The Bottom Line," *OOPS Messenger 4*, 2, 101–108.

Coplien, J.O. (1994), "Examining the Software Development Process," *Dr. Dobb's Journal 19*, 11, 88–95.

Coplien, J. and D. Schmidt, Eds. (1995), *Pattern Languages of Program Design*, Addison-Wesley, Reading, MA.

Deming, W.E. (1986), *Out of the Crisis*, MIT Center for Advanced Engineering Study, Cambridge, MA.

Fraser, S., K. Beck, G. Booch, D. Coleman, J. Coplien, R. Helm and K. Rubin (1994), "How Do Teams Shape Objects? – How Do Objects Shape Teams?" *OOPSLA 1994 Proceedings 29*, 10, 468–473.

Fraser, S.D., K. Beck, G. Booch, D. Coleman, J. Coplien, R. Helm and K. Rubin (1995), "How Do Teams Shape Objects? – How Do Objects Shape Teams?" (Addendum to the *Proc. of OOPSLA/94*) *OOPS Messenger 5*, 20, 63–67.

Gabriel, R.P. (1994), "Productivity: Is There a Silver Bullet?" *Journal of Object-Oriented Programming 7*, 1, 89–92.

Gamma, E. (1992), *Object-Oriented Software Development Based on ET++: Design Patterns, Class Library, Tools*, Springer-Verlag, Berlin.

Gamma, E., R. Helm, R. Johnson and J. Vlissides (1995), *Design Patteners: Elements of Re-Usable Object-Oriented Software*, Addison-Wesley, Reading, MA.

Hartley, J. (1992), *Concurrent Engineering: Shortening Lead Times, Raising Quality, and Lowering Costs*, Productivity Press, Cambridge, MA.

Harrison, N. and J.O. Coplien (1996), "Patterns of Productive Software Organizations," *Bell Labs Technical Journal 1*, 1, Summer 1996.

Janis, I.L. (1971), "Groupthink," *Psychology Today*, November 1971, 43–46, 74–76.

Katz, D. and R.L. Kahn (1978), *The Social Psychology of Organizations*, 2nd Edition, John Wiley and Sons, New York.

Koenig, A. (1995), "Antipatterns," *Journal of Object-Oriented Programming 8*, 1, 46–48.

Krishnamurthy, B. and D.S. Rosenblum (1991), "An Event-Action Model of Computer-Supported Cooperative Work: Design and Implementation," In *Proc. of the International Workshop on Computer Supported Cooperative Work*, IFIP TC 6/WG C.5, pp. 132–145.

Mackenzie, K.D. (1986), "Organizing High Technology Operations for Success." In *Managing High Technology Decisions for Success*, J.R. Callahan and G.H. Haines, Eds.

Moreno, J.L. (1934), *Who Shall Survive?: Foundations of Sociometry, Group Psychotherapy, and Sociodrama*, Nervous and Mental Disease Publishing Co., Washington, DC.

Ouchi, W.G. (1981), *Theory Z: How American Business Can Meet the Japanese Challenge*, Addison-Wesley, Reading, MA.

Schneiderman, B. (1980), *Software Psychology*, Winthrop Publishers, Cambridge, MA.

Sengé, P. (1990), *The Fifth Discipline*, Doubleday, New York.

Swieringa, J. and A. Wierdsma (1992), *Becoming a Learning Organization*, Addison-Wesley, Reading, MA.

Wasserman, S. and K. Faust (1994), *Social Network Analysis*, Cambridge University Press, New York.