◆ Patterns of Productive Software Organizations

Neil B. Harrison and James O. Coplien

Software development is a predominantly social activity. Individuals in a software project fill various roles and communicate with other roles, forming a social network of communication that embodies many important characteristics of the organization. These social networks lend themselves to both quantitative and visual analysis. In an attempt to isolate important factors that contribute to software productivity, we have used both visual and quantitative data to uncover patterns of organization and process that are characteristic of highly productive software projects. These patterns, in turn, have enabled us to establish guiding principles for roles and for communication among roles. Several projects have begun to apply these patterns in a generative manner to reshape their organizations.

Introduction

Although software development is generally viewed as a cerebral activity performed by individuals, it is really highly social. A typical software project consists of many software developers, system engineers, testers, managers, documentation specialists, and users, all communicating with one another. The patterns of interaction among the participants in a software project are important factors in the effectiveness of the team and may even have a significant impact on the ultimate success of the project.

Traditional analysis of software development has focused on *process*, or more precisely, the sequence of steps used by developers to create software, combined with various artifacts produced along the way. Although this type of analysis is useful, particularly for such things as achieving International Organization for Standardization (ISO) certification, it does not take into account interpersonal interactions and various social aspects of software development organizations.

Other views of development process complement and sometimes overshadow mainstream task-oriented process models. We used a role-based approach in studies of more than forty software development organizations. These analyses have provided us with insights into the types of practices and organizational structures typical of productive software organizations. We have codified these into sets of software development patterns that can be used by organizations to help develop software efficiently and effectively. A few organizations have already applied these patterns to establish the foundations of their projects or to transform troubled software organizations into healthy ones.

In this paper, we describe our approach to software process analysis and our specific visual analysis techniques. We also outline some characteristics of roles and communication among roles that shape software development organizations. After describing some key organizational patterns of highly productive software organizations, we conclude by explaining future directions of this work.

Role-Based Organizational Analysis

Early research suggested that roles be viewed as a basic unit of abstraction for organizational studies.[1] Roles are informal (or formal) positions with identifiable responsibilities. Typical examples of roles are those of designer, system tester, manager, and user. Besides having its own set of responsibilities, a role may communicate with another role. Within an organization, several people may play the same role, and one person may play different roles at different times.

In half-day sessions with project technical staff, we captured information about roles on CRC (class, responsibility, collaborator) cards, a common technique in object-oriented analysis.[2] Using one card for each class, or role, we identified the name of the role, its responsibilities, and other roles with which it communicates (its collaborators). 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 who normally performed those roles. The participants then acted out the process being studied. As a role became active, the person playing it explained the relevant tasks, or *responsibilities*, and identified dependencies on and communication with other roles, called *collaborations*. The role player also assigned a relative strength to each collaboration. All this information was recorded on the cards.

We have used CRC cards to gather data from more than forty organizations, almost all within the software industry. Our initial studies focused on large system development efforts in telecommunications, but we have since expanded our horizons to include projects of all sizes, except those with fewer than five people. We have also studied various applications, including aerospace organizations, medical software development, and consumer software. Most organizations that took part were directly involved in software development, but a small sample of them perform market management or other assorted functions.

Visual Analysis of Projects

To analyze the data we collected, we used two different techniques: force-based social networks and interaction grids, described in the sections that follow.

Natural Force-Based Social Networks

We collected the data from the CRC cards in a database and used a tool called Pasteur[1] to display the data. Creating a graphical representation of the cards, Pasteur provides various manipulation and visualization techniques that enable the user to analyze organizations.

Of these techniques, the one we use most frequently is force-based relaxation rendering. The cards, each of which represents a role, are placed on the screen randomly and caused to repel each other. A collaboration is represented as an attracting force between roles, based on the strength of the collaboration. Roles with collaborations move toward each other and away from those with which they have no collaborations. Eventually, they reach a stable state in which the roles with a strong affinity for each other are grouped together. Roles that exhibit collaborations with many others end up nearest the center. These properties give strong visual cues about the structure of an organization, making it easy to identify key roles and natural suborganizations.

To indicate the degree of collaboration with other roles, we color code the role cards. Shades of red, from dim to bright, are used to represent a role's relative degree of collaboration. As collaboration increases, roles glow brighter red. The roles may be overlaid with red, yellow, and green lines representing, respectively, strong, medium, or weak communication links with other roles.

In Figure 1 the software developer, shown at the center of the organization, has strong communication links to many other roles. Immediately to the right of the software developer, a cluster of roles—the system engineer and developers of firmware and hardware—interact closely with each other and with the software developer. This particular project appears to have a multi-role development core.

At the upper right is a cluster of four roles, revolving around the "core team" role. Together, these form a project management subgroup. On the far left another subgroup—product support, system test, and the modification request (MR) review board—represents quality assurance functions.

In this organization, the software developer and the customer can communicate only through the core team role. Separating the developer from the customer may prevent the developer from understanding the nuances of the customer's desires, potentially creating rework late in development or causing customer dissatisfaction. For this organization, we might recommend establishing a dialogue between customers and developers.

Interaction Grids

As an alternative to force-based relaxation rendering, we use interaction grids, a technique inspired by the work of Church and Helfman at Bell Laboratories[3] and DePauw et al.[4] An *interaction grid* is a square matrix whose columns are the roles that initiate collaborations, and whose rows are the roles that receive the collaborations. The roles in both rows and columns are sorted according to the amount of collaboration, with the busiest roles at the origin. As in the adjacency diagram described earlier, the roles are color coded in red. The squares of the grid, which represent collaborations between roles, vary from dim to bright red, with the brightest indicating the greatest degree of collaboration.

Figure 2, the interaction grid for the organization shown earlier, is one of the sparser grids we have seen. The most central role, the software developer, receives communication from most other roles, but also initiates communication to most other roles. Clearly, the software developer is the hub of the organization.

Figure 1.
Adjacency diagram for a design process.

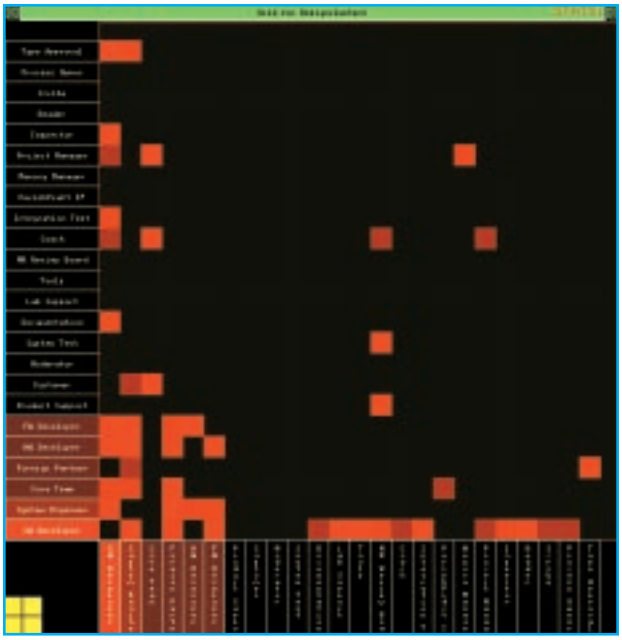

Figure 2.
An interaction grid for a design process of the organization shown in Figure 1.

The six busiest roles, with the exception of the core team role, have almost complete connectivity among themselves. In this project, these roles formed a small team that worked closely together to design software. Although using small subteams may be very effective, it introduces the danger that peripheral roles may not receive information critical to their jobs.

Visualizations also support further introspection by helping the organization face and understand its problems. The location of key roles in the diagram usually confirms the development team's expectations or helps explain exceptional or problematic behavior. For example, one organization immediately recognized the remote position of the architectural role in its social network diagram as one reason for its lack of product focus. Visualizations not only pinpoint organizational strengths or weaknesses, but also serve as a mirror in which team members can see and understand themselves better.

Analysis of Data

Besides analyzing individual projects, we examined all data across projects to uncover trends that centered on roles, collaborations, and organization size. Organization size refers to the number of distinct roles in an organization, not to the number of people on any given project. In some cases, the larger, more mature projects did have a greater number of roles, but there were many exceptions.

For an organization with n roles, the number of possible links is

$$\frac{n(n-1)}{2}.$$

In any organization, only a portion of the possible collaborations are exercised; everyone does not talk to everyone else. The ratio of the actual collaborations to the possible collaborations is called the *communication saturation*.

As the number of roles increases, it is reasonable to expect the number of communication paths to increase roughly as the square of the number of roles. However, Figure 3a shows that the number of communication paths increases as a *linear* function of the number of roles. This linear property is independent of the size of the organization. Adding a role increases the number of collaborations by a constant amount (on average).

As the number of roles increases, the number of possible communication links increases quadratically, but the number of actual links increases linearly. Therefore, the communication saturation decreases in a hyperbolic manner as the number of roles increases. Figure 3b shows a scatter plot of communication saturation as a function of roles.

For organizations with very few roles, satura-

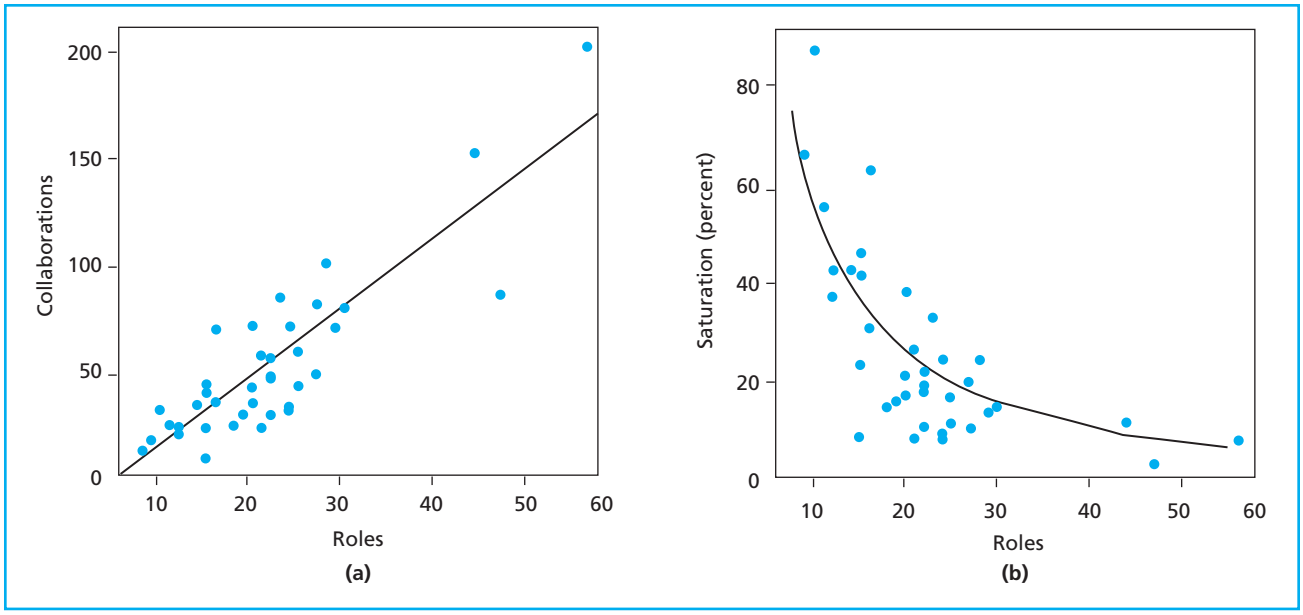

Figure 3.
(a) Roles and collaborations, with least squares fit. (b) Communication saturation, with line y = 6/x superimposed.

tion is generally complete; everybody talks to everybody else. However, the communication saturation rate drops very quickly, with the bulk of the organizations studied having saturation rates of ten to thirty percent. For large organizations, the impact of adding a role is proportionally smaller.

What is the impact of low communication saturation? A low saturation rate indicates that roles are not communicating with one another. In many cases, this is appropriate; certain roles may be more or less independent. In fact, larger organizations probably have evolved toward role specialization as a way to handle their complexities. However, such a condition may increase the risk that critical information does not reach the right person, or that the information is not received in a timely manner.

Communication appears to be critical to success; highly productive organizations have high communication saturation. The cost inherent in communication, however, makes it prohibitively expensive for organizations with many roles to achieve high communication saturation. None of the projects with a large number of roles had high communication saturation, and none were highly productive, leading us to conclude that complex organizations would benefit from a reduction in the number of roles they have.

Individual Roles and Communication

It is instructive to examine individual roles within projects. During our studies, we noticed that certain roles were directly involved, and others indirectly involved, in creating a product. Those directly involved are performing *producer roles*, such as a designer, coder, or tester. Those in support roles, on the other hand, somehow support producer roles in their tasks. As we mentioned earlier, the roles at the center, generally the busiest roles, are almost always producer roles. The difference between the amount of their communication and the average role in their organizations is often substantial.

We measure the difference in communication rates within an organization by using the *communication intensity ratio*, the ratio of the busiest role's link count to the average link count of that organization. It measures, in effect, how much the communication is concentrated on a single role. In general, the most productive projects have low communication intensity ratios, meaning that communication is distributed evenly among the roles. A low communication intensity ratio, however, does not guarantee high productivity. We have found that projects with less than stellar productivity may also have low communication intensity ratios.

As an organization becomes more complex, one might expect that communication would be spread more evenly, with the communication burden being shared more or less equally among the roles. However, Figure 4 shows that as an organi-

zation grows, the communication intensity ratio increases and the organization becomes more hierarchical, with the busiest role becoming even busier. Extremely busy roles spend much of their time in communication, allowing little time for directly productive activities. Because these roles are almost always producers, high organizational complexity may tend to harm productivity.

Characteristics of Highly Successful Projects

As we collected gross productivity measures from projects, we recognized that current productivity measures, such as lines of code per staff month, are limited in their precision. Therefore, we considered only large differences to be significant. The few projects in which lines of code per staff month were an order of magnitude larger than average were designated as "hyperprogramming" projects. In a few other projects, the development interval from beginning to end was about twice as fast as average. We noted these projects, although they were not necessarily conducted by hyperprogramming teams.

What made these teams especially productive? In analyzing the data, we searched for distinguishing characteristics of organizations that were highly productive. Certain themes recurred among all the projects, and certain others appeared chiefly in successful projects. We have codified these themes into patterns of software development as they relate to organizations and interpersonal interactions,[5] and we give a sample of some of the most significant patterns in the sections that follow.

Keep Organization Simple

Thoreau stated, "Our life is frittered away by detail... . Simplify, simplify." The most productive organizations we studied averaged fewer roles (about 16) than the average for all organizations (about 21). Often, they also had fewer people in the organization, but the projects themselves were not necessarily small. Perhaps fewer roles require fewer people to fill them, or small organizations do not allow themselves the luxury of many roles to fill.

Work Flows Inward

The roles at the center of a relaxation diagram, the busiest roles, indicate where the focus of the organization is. In healthy organizations, indeed in most organizations we studied, the roles at the center, such as designer or coder, were directly involved in the production of the desired product. Other roles should support these; when they instead become central, the focus of the project

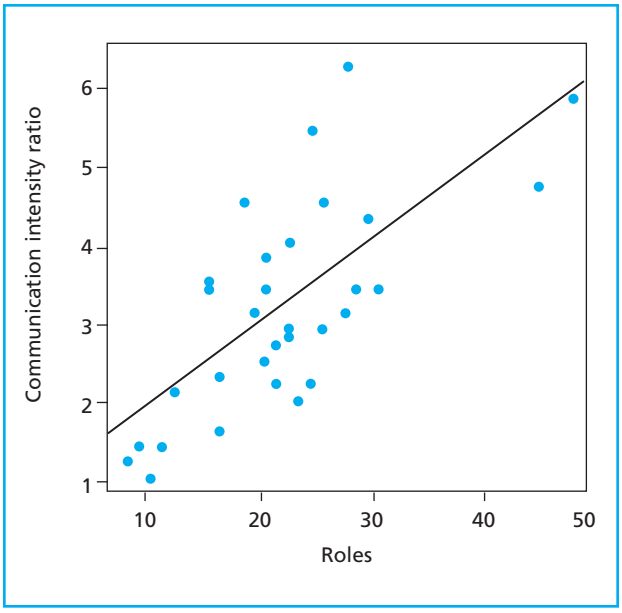

Figure 4.
Communication intensity ratio as a function of roles.

shifts away from the product itself. This situation is reminiscent of the lament by Philip Howard about the United States government: "How things are done has become far more important than what is done... . Process now has become an end in itself."[6]

Of course, if developer centeredness is too strong, the developer may be spending more time communicating than developing. Some developers report that they work nights and weekends because it is the only time they can get anything done.

The central role of the developer is necessary, but appears to be insufficient for high productivity. Communication, and by implication, work, must flow toward the central roles. When the central roles are generally consumers, rather than producers of communication, the organization tends to be more productive. Figure 5a illustrates such an inward-directed organization.[5] The implications are as follows: First, because peripheral roles generally have external connections, they are in a position to convey external needs to the producers. Second, the producers can get the information they need from just a few sources. Therefore, they spend less time getting the information they need and more time producing.

By contrast, Figure 5b shows an outward-directed organization.[5] In this case, the central roles are creating work for everyone else, rather than decreasing it. Because developers are often not at the center of this type of organization, pro-

ducers may need to talk to many people to get the information they need. This may indicate organizational pathology.

Distribute Work Evenly

Most organizations focus on the developer role. The highly productive organizations we have seen do not allow that focus to become extreme; instead, they spread communication around (which implies an even balance of work as well).

The Quattro Pro* for Windows* project at Borland International[7] was one of the most productive projects we have ever seen, with code production rates on the order of one thousand lines per person per week over the life of the project. Figure 6 shows the adjacency diagram and interaction grid for this project.[5] Both diagrams demonstrate the even distribution of communication among the roles. The communication intensity ratio is very low, indicating that the most central role does not shoulder a disproportionate amount of the communication burden.

Iterate, Iterate!

In nearly every organization we studied, the activities of software design and coding were inseparably intertwined. In fact, many organizations declined to separate the two roles, instead identifying a single role, "developer." For these organizations, the traditional waterfall model of software development exists only on paper. A number of particularly successful projects included tight prototyping loops of features such as user interfaces. In one project, a team of system engineer, software designer, and tester worked on the user interface portion of the product, iterating as often as three times a week. Successful iteration requires close coupling with customers or customer surrogates who can provide evaluation and feedback.

We found very few cases in which design and code are well-separated activities. The exceptions are in mature developments where the domain is well understood and defined, and instantiation of code from a design specification is largely automatic.

Compensate Success

Projects that are highly successful almost always provide meaningful rewards for their successful completion. Those with the highest productivity also tended to have a very lucrative reward structure. Celebrations can also be effective reward mechanisms. This is fully described in Coplien.[5]

We have seen some evidence of potential pitfalls in reward structures. If the proffered reward is not perceived as appropriate or desirable, it will not motivate. Rewards can also be cheapened if they are given when success is not achieved. In addition, if a reward is offered for the achievement of an apparently unachievable goal, such as an unreasonable schedule, the effect may tend to discourage rather than motivate.

Synthesis: What We Learned

Our studies have produced a large volume of patterns, many more than we can describe here. Other work[5] gives a complete list of the patterns we have discovered to date.

While studying organizations, we have been struck with the common sense nature of nearly every pattern we uncovered. It is perplexing, then, that organizations rarely follow all the patterns, and that they experience some difficulty in adopting them. An organization's practices are an integral part of its culture—how an organization does things is defined, in part, by who and what it is, and to change how it operates is to change its identity. For this reason alone, the participatory introspection phase of our CRC card exercise is critical; it helps participants reach a clearer understanding of who they are and what they want to become.

In some cases our organizational analyses and patterns have precipitated significant changes in the organizations we studied. An important sign of acceptance for our process was having participants adopt the vocabulary of the patterns in their everyday speech. In one case, managers began talking about becoming a "developer-centric organization." It appears that broad adoption of changes starts to occur when people begin to assimilate the patterns into their standard vocabulary.

Future Directions

The research into the nature of software development teams presented in this paper represents only a small part of an extensive effort that has spanned several years. Although we have learned much, this work is far from finished. We hope to analyze more projects to continue verifying our insights and patterns.

Beyond studying organizations, we hope to have more opportunities to apply these patterns to both new and existing organizations. Organizations that have used these patterns report encouraging results; we would like more data.

Our conclusions about the characteristics of highly productive organizations are still somewhat preliminary. Although we have not been able, nor may we ever be able, to quantify the benefit of applying individual organizational patterns to other

(a) (b)

Figure 5.
Work flows inward. (a) Because peripheral roles generally have external connections, they can quickly convey external needs to the producers. (b) In an outward-directed organization, the central roles create work for everyone else.

(a) (b)

Figure 6.
Borland's Quattro Pro for Windows (a) adjacency diagram and (b) interaction grid.

organizations, we should be able to provide guidelines for their systematic application. Such insights will become clearer as more projects are studied.

It may be instructive to study the extremes—the "sick" organizations as well as hyperprogrammers. They may teach us which practices and patterns we should avoid. It is perhaps understandable, however, that few dysfunctional teams have requested an organizational analysis.

Software has become profoundly important in the world today. Because software is developed in teams, we must understand how these teams work most effectively. Therefore, we expect to continue with this significant work. The very nature of organizational structures, however, defies attempts to study them using traditional controlled experimental methods. After all, we are studying the interaction of the most complex entities in the world—humans.

References

1. B. G. Cain and J. O. Coplien, "A Role-Based Process Modeling Environment," *Proc. of the Second International Conference on the Software Process*, Los Alamitos, California, February 1993, IEEE Computer Press, pp. 125-133.
2. Kent Beck, "Think Like an Object," *UNIX Review*, Vol. 9, No. 10, September 1991, pp. 39-43.
3. K. W. Church and J. I. Helfman, "Dotplot: A Program for Exploring Self-Similarity in Millions of Lines of Text and Code," *Journal of Computational and Graphical Statistics*, Vol. 2, No. 2, 1993, pp. 153-174.
4. W. DePauw, et al., "Visualizing the Behavior of Object Oriented Systems," *SIGPLAN Notices*, Vol. 28, No. 10, 1993, pp. 326-337.
5. James O. Coplien, "A Generative Development-Process Pattern Language," *Pattern Languages of Program Design*, Addison-Wesley, Reading, Massachusetts, 1995, pp. 183-237.
6. Philip K. Howard, *The Death of Common Sense: How Law is Suffocating America*, Random House, New York, 1994, p. 60.
7. James O. Coplien, "Evaluating the Software Development Process," *Dr. Dobb's Journal*, Vol. 19, No. 11, October 1994, pp. 88-97.

Further Reading

Thomas Allen, *Managing the Flow of Technology*, MIT Press, Boston, 1977.

Barry W. Boehm, "Improving Software Productivity," *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, N. J., 1981, pp. 641-689.

Daniel Katz and Robert L. Kahn, *The Social Psychology of Organizations*, 2d ed., John Wiley, New York, 1978.

Edward E. Lawler, *Pay and Organization Development*, Addison-Wesley, Reading, Massachusetts, 1981.

Stanley Wasserman and Katherine Faust, *Social Network Analysis: Methods and Applications*, Cambridge University Press, Cambridge, Massachusetts, 1994.

M. R. Zuckerman and Lewis J. Hatala, *Incredibly American*, ASQC Quality Press, Milwaukee, 1992.

*Trademarks

(Manuscript approved April 1996)

NEIL B. HARRISON is a member of technical staff on the Development Technologies Team of Business Communication Systems (BCS) at Lucent Technologies in Denver, Colorado. He works on software and organizational patterns, domain analysis, object-oriented development, and software reliability engineering. Mr. Harrison received a B.S. from Brigham Young University, Provo, Utah, and an M.S. from Purdue University, West Lafayette, Indiana, both in computer science.

JAMES O. COPLIEN is a principal investigator in the Software Production Research Department at Bell Labs in Naperville, Illinois. He carries out research programs in software design patterns, empirical organizational modeling, multiparadigm design, and the object paradigm; he is also a C++ expert. Mr. Coplien received a B.S. in electrical and computer engineering and an M.S. in computer science, both from the University of Wisconsin at Madison. ◆