2000

A Social Analysis of Software Development Teams: Three Models and their Differences

Steve Sawyer
Pennsylvania State University, sawyer@ist.psu.edu

A Social Analysis of Software Development Teams: Three Models and their Differences

Steve Sawyer, School of Information Sciences and Technology, The Pennsylvania State University, 513 Rider I Building, University Park, PA 16801, sawyer@ist.psu.edu.

Abstract:

In this paper we analyze the socio-technical activity called software development by focusing on the social perspective. We do so to pursue two questions: What can we learn about software development by focusing on its social aspects and what insight does a social perspective give us regarding the production methods, techniques and tools uses in software development? From the social perspective, this analysis suggests three models of software teams. For each of these we outline, compare, and comment on issues with the way the task, methods and tools are conceptualized. We include a brief discussion of hybrid models such as those used at Microsoft and other packaged software vendors.

Introduction

What can we learn about software development by taking a social perspective? That is, what can we learn about software development by focusing on social structures, interpersonal interactions, and the rules and norms that define these structures and interactions? And, what insight on software development production methods, techniques and tools does a social perspective provide? By social perspective we mean explicitly acknowledges that a work group's members organize themselves and interact in ways that both respond to and shape the production tasks which they are charged to perform (Goodman, 1986). A social perspective on software development differs from the more traditional production perspective by highlighting the social milieu in which the work done to make code takes places (Sawyer and Guinan, 1998; Wynn and Novak, 1995). By taking a social perspective we focus on the social unit – an aggregate set of behaviors – and only indirectly address the individual perspective on software development.

This paper continues in five sections. We begin by highlighting the socio-technical nature of software development to highlight the interdependencies between the social and technical (or production) aspects of this effort. In the second section we outline three general models of software development that a social perspective helps to illuminate. In the third section we discuss some of the issues and insights that arise from comparing these three models of software development. Section five presents a short discussion of hybrid models. We conclude with a discussion of hybrid software models.

The Socio-Technical Nature of Software Development

In this paper software development is conceptualized as a set of activities comprised of people interacting with each other and with (primarily information) technologies they use to do their work. That is, software development is a socio-technical activity (Bijker, 1995). A socio-technical characterization of software development highlights the analytic distinction between people and the technologies they use. However, in practice it is very difficult to disentangle the way people do things from the technologies that they use. Nor is it easy to gain insight into the use of a technology if it is removed from the social context of its use. That is, while the social and technical aspects of a socio-technical system can be explicitly characterized, they are mutually interdependent. This mutual interdependence makes it difficult to understand either aspect independently.

Interdependence also suggests a context-dependency – that a specific socio-technical system is, in turn, a part of a larger socio-technical system. This further suggests that different contexts will shape the use of the same technical artifact in differing ways (Sproull and Goodman, 1989). One example of this context-dependency is the differential uses of Lotus Notes across departments of a large consulting firm (Orlikowski, 1993) and the same technology at Zeta Corp. (Orlikowski, 1996). Another example is the variations in use of similar computer-aided software engineering (CASE) tools among software development teams (Guinan, Cooprider and Sawyer, 1997).

Relative to software development, the technical aspects of this socio-technical system include production methods (such as the spiral model), production techniques (such as joint application development or JAD) and production tools (such as computer aided software engineering or CASE). The social aspects of software development include the range of social actions and efforts between and among members and stakeholders, the structure of these interactions and the relationship between the task and group member's structures and processes. Issues of skill, domain knowledge, interpersonal conflict and project management thus reflect the social aspects of software development's socio-technical nature (Sawyer, Farber and Spillers, 1997).

Traditionally the most common perspective on software development has been technical, focusing on the means of production. That is, most of the software development literature is focused on production methods, techniques or tools research. Because of the socio-technical nature of software development, implicitly or explicitly this body of research embeds certain views of the social aspects of this effort into the production view. For example the Software Engineering Institute's (SEI) Capability Maturity Model (CMM) focuses primarily on the production aspects of software development. The People-Capability Maturity Model (P-CMM) goes on to explain how people can best fit the CMM approach (Humphrey, 1989; 1995). Together, the CMM and P-CMM reflect the traditional production first, people second, approach to the socio-technical nature of software development. In contrast, in this paper we ask: what can we learn about software development by privileging the social perspective?

Three Models of Software Development Behavior

From a social perspective we can articulate three perspectives that highlight the social processes of software development: the sequential, the group and the network model. We define each of these in Table 1. In the rest of this paper we address two questions that arise from this characterization. Firstly, in what ways do these perspectives differ? Secondly, what insight can these differences provide? Responding to the first question, in Table 2 we present a summary of the contrasts among these three social process models of software development. We discuss each in the following sub-sections.

Table 1: Three Models of the Social Processes of Software Development

Model	Definition
Sequence	Software development is seen as a production effort. It is a linear set of discrete tasks. People work in specialized functions with a minimal need to interact across these functions. All information needed for subsequent steps/tasks are embedded in the work product or in supporting documentation. This implies a functional organization of people with a hierarchical management approach. The traditional waterfall model is an example of the sequential model of software development.
Group	Software development is seen as combination of development and production. A set of discrete tasks may need to be repeated until the product is complete. Group members are interdependent and are valued for their particular skills and for heir ability to work with others. This implies a team-based model and a collaborative management approach. The spiral or evolutionary method is an example of the group model of software development.
Network	Software development is seen as a process of constant development with a specific focus on the outcome/ product. Tasks are not seen as sequential. Tasks are also tied to individuals (or small groups) who often compete for inclusion. Group members are valued for these abilities. This implies a complex network of ties between people and a hub-and-spoke management approach. The open source effort exemplifies the network model of software development.

Table 2: Aspects of the Three Models of Software Development Team/Behavior

Model Aspect	Sequence	Group	Network
Conceptual Basis	Industrial Engineering	Social Psychology	Sociology
Perspective	Process 1st	Process 1st	Product 1st
Orientation	Control	Conflict	Interaction
Use	Prescriptive	Normative	Descriptive
Task view	Production	Production / Development	Development
Implied Method	Linear & Sequential	Iterative & Sequential	Emergent & Non-linear
People's Actions	Prescribed	Role & goal-driven	Individual but linked
Examples	SDLC, SEI/CMM	Spiral, JAD, RAD	Open source

680

The Sequential Model

The conceptual basis for the sequential model of social processes in software development comes from the work design literature in industrial engineering (Nadler, 1963). This perspective on software developer interactions is driven by the work first laid out by Frederick Taylor and often known as scientific management. The social process embedded in a sequential model of software development implies a linear and pre-specified ordering

of the requisite tasks. This pre-specified ordering further implies a prescriptive view of the production model. Each person's task is discretized and specialized. Specialization allows for greater expertise within the task and makes it easier to train for the task. The sequence model prioritizes the process over the product and the underlying belief is that a good process leads to a good product.

The social interactions in a sequence model of software development are based on the concepts of control. That is, people's interactions are seen solely as a function of the work they do. This work can be measured and compared, the roles are stable, so the needed interactions can be formalized (and perhaps even automated). This also suggests that any particular member can be replaced as needed by another person with the same functional level of training. The sequence model of social process also highlights the importance of automation: for discrete tasks (such as coding or testing) capital can replace (or at least augment) labor. The social structures of a sequence approach are hierarchical, with little intra-functional discussion needed (save through formal channels). The emphasis is on embedding the required information in the work product and/or associated documents to pass on to each downstream task. As such there is a control orientation to the interactions among team members and very little need to create strong social bonds. Examples of this include the traditional waterfall models (such as the systems development life cycle or SDLC), the CMM and the SPICE project.

The Group Model

The group model draws its intellectual roots from social psychology, primarily that of "work redesign." (Hackman and Oldham, 1980). Work redesign arose in response to the issues (such as motivation, retention and productivity) that typically arise in response to a work design approach. In a group model of software development social process the work effort is seen as interactive and collaborative. The tasks, while often sequential, are seen as iterative and there is explicit attention to process improvement (or tuning) by the members of the group. Thus, a group model is normative. From this perspective, software development processes are based on a set of predefined tasks but this must also take into account (and build on) the individual skills and weaknesses of the group's members.

A group model of software development makes explicit that there is a role for social interaction among the group's members. This means that there is the potential for *semi_* automation of tasks and that the tools and methods should explicitly provide for collaboration among the group's members. The social structures and interaction modes are oriented to resolving the inevitable conflicts that arise from people collaborating. The group model also explicitly recognizes, in its iterative nature, that software production and development are (often intimately) linked. From this perspective, the production process is more flexible than it is from a sequence perspective. Still, the implicit belief reflects the process-first approach also present in the sequential model. Examples of group models of the social process include the spiral or evolutionary approach to software development, rapid application development (RAD) and joint application development (JAD).

The Network Model

The network model draws on social network theory (Grannovetter, 1973; Wellman, et al, 1996). In this model a group of people are linked by the relative "strength" of the social ties between them. These ties reflect the frequency and value derived from these ties. Stronger ties reflect common interests, significant information sharing and significant interaction. Weaker ties reflect less frequent interactions and different forms of information transfer.

The chief programmer team model of software development (see Baker, 1972) is one well known example of a network model. In this form, the network appears to be a hub-and-spoke: strong ties between members and the chief programmer and weak ties between individual team members. The recent growth of open source development efforts (see Raymond, 1999) reflects a second form of the network model. This network has multiple hubs (of varying relative ties strengths) and multiple ties between members of the network. In both, however, there is some form of central hub. The chief programmer is the clear center in Baker (1972) while most open source efforts rely on small number of hub/people to server as a center.

In the network model, the production process is secondary to the product. Moreover, the process typically emerges and reflects the network ties developed by the participants. This implies that a network model tends to be descriptive – it reflects a "what is happening" perspective. However, this emergent process is constrained by tasks. That is, even if the process is not central, there is some form of version control, of testing and of documentation. One belief underlying the network model of the social processes of software development is that a good product comes from having good people. This people-first approach recognizes that it is very difficult (if not impossible) to replace key members (or a member) of a network as they are the hubs of connection. Simply, what would have happened to Mosaic (and Netscape) if Mark Andreesen had left the project?

The network perspective implies that the individual members, and the social ties that connect them, define the software development effort. To support the network it becomes critical that the software development tools provide for interconnection. Simply, any software development tool is valued for how it helps the individual member and/or for how well it enables network

interconnectivity. A second implication is that a network model, being both emergent and descriptive, reflects a developmental view (as opposed to production) of software. Further, given the centrality of the social structures and individual members interaction to a network model, the group's effort are often contentious (Zachary, 1998). From this perspective, the interactions between the members are focused on product features, functions or actions. There are few procedural details and the expectation among the network's members is, essentially, "show and tell." That is, the resolution of disagreements is often rooted in delivering code that delivers on the concepts discussed. This also re-affirms the product (or outcome) oriented belief mode that typically permeates a network model of the social processes of software development.

Hybrid Models

A more typical scenario is that an actual development team may adopt a hybrid model of social process. For example, both Baker (1972) and Brooks (1975) write about the IBM System 360 development effort. Brooks highlights the importance of a project development structure (essentially the SDLC) while Baker explains how the chief programmer team (a star network model) works. In this set of cases, the sequential model advocated by Brooks provides a stylized view of development while Baker's view provides insight into how the chief programmer can create a hybrid social process underlying the sequence of the SDLC.

One proposition that arises from this is: *for the SDLC to work, a hybrid network of relations is needed between the developers.* Circumstantial evidence suggests that this may be true. For instance, Weinberg (1971) noted how developer productivity fell when vending machines were removed from programmer's break area and they, subsequently, spent less time interacting informally with each other. Simply, the developers lessened their reliance on the informal network that had emerged around the vending machines.

Carmel and Sawyer (1998) and Sawyer (2000) document how packaged software (software made for sale/license to others) development approaches differ from more traditional custom/internal software development. They note that in packaged development – relative to custom development, developer interaction is greater, there is less reliance on formal processes, and more reliance on product construction. Many of these issues are also documented by Cusamano and Selby (1997). They highlight this product-focused approach in their analysis of Microsoft's development practices. Zachary (1998), in his research on Microsoft's development practices, highlight that such a product focus allows for internal competition among developers. Further, he characterizes the Microsoft development approach as a hybrid between a group and network model. For each development effort, a small number of critical people (called the core team by Carmel and Becker, 1995) serve as the hubs in the evolving network of programmers. However, these networks are then formally structured and this lead to issues (of conflict and interdependence) of the group model to software development.

Finally, the issues with hybrid models can also show up in the use of tools to support software development. For example, Vessey and Sravanupudi (1995) show how CASE tools are typically designed to be used by individuals (and implicitly or explicitly support a sequence perspective). However, CASE tools are often used by groups. The implicit (or explicit) sequence-based model of CASE tools makes it difficult for groups to use these effectively (Guinan, Cooprider and Sawyer, 1997).

Privileging the Social Perspective

Our point is that a social perspective on software development helps us identify three general models of the social organization. Comparing these three models, in turn, leads us to insights on the socio-technical act of software development. And, given the predominance of the production-focused perspective of software development, a social focus provides a means to explore the current literature's findings from a different perspective.

A social perspective on the sequence model of software development leads to highlighting why there are problems with adhering to the production process (methods). Simply, such a process demands people's jobs become over specialized and thus have little autonomy, task feedback, task variability and limited skill sets (Hackman and Oldham, 1980). That is, a sequence model implies the intellectual equivalent of factory work without the attendant product safeguards.

From a social perspective the group model of software development provides for a more enriching and rewarding experience for team members. The group model explicitly ties to the production process and better reflects the variations of its human performers (McGrath and Hollingshead, 1994. Because this approach is premised on consensus, problems with conflict and its management tend to dominate the effort (Curtis, Krasner and Iscoe, 1988; Robey, Farrow and Franz, 1989; Zachary, 1998).

The network model fully embraces the social perspective as the dominant organizing principal. This is why this approach often has problems with process stability and risk management (Zachary, 1994; Raymond, 1999). Because the process (and process controls) are tied to people (or nodes) the loss of that person removes that aspect of the process. Further, and coincidentally, such person-specific process ownership often leads to the same sort of over-specialization (and limited personal value) that arises in sequence-model approaches to development. That is, in the network model, a person could become so tied to a few parts of what they do that it dominates their role in the network.

Such differences between these three models of software development leads to asking: which is the best approach? This is both an empirical and philosophical question that we do not address in this paper. However, we do provide evidence that hybrid social process models exist and that these models can be decomposed into some combination of the three base social process models. Further, this analysis of the social processes of software development suggests that the socio-technical nature of this effort is often under-represented by focusing on the technical (or production) side. A social process view of software development helps to see past the tools and to gain a better sense of the relation between the tools and the people's use of these tools.

References

Baker, F. Chief Programmer Team Management of Production Programming. *IBM Systems Journal*, 11(1), 1972, 56-73.

Bijker, W., "Of Bicycles, Bakelites and Bulbs: Toward a Theory of Sociotechnical Change," MIT Press, Cambridge, MA, 1995.

Brooks, F. The Mythical Man-Month, *Datamation*, 1974. 44-52.

Carmel E and Becker S. A process model for packaged software development. *IEEE Transactions on Engineering Management, 41***(**5), 1995. 50-61

Carmel E and Sawyer S. Packaged software development teams: What makes them different? *Information Technology & People, 11*(1), 1998. 7-19

Curtis, W., Krasner, H. and Iscoe, N. A field study of the software design process for large systems. *Communications of the ACM*, 31(11), 1988, 1268-1287.

Cusumano M and Selby R How Microsoft builds software. *Communications of the ACM, 40*(6), 1997 53-61.

Goodman, P. *Impact of Task and Technology on Group Performance*. San Francisco, Jossey-Bass, 1986.
Granovetter, M. The Strength of Weak Ties. American Journal of Sociology, 78(6), 1973, 1361-1381.

Guinan, P. Cooprider, J. and Sawyer, S. The Effective Use of Automated Application Development Tools: A Four-Year Longitudinal Study of CASE. *IBM Systems Journal* 38, 1997,124-141.

Hackman, J., and Oldham, J. *Work Redesign.* Addison-Wesley, Boston, 1980.
Humphrey, W. *Managing the Software Process*. Reading, MA: Addison-Wesley, 1989.

Humphrey, W. A Discipline for Software Engineering. Reading, MA: Addison_Wesley, 1995.

McGrath, J., and Hollingshead, A. *Groups Interacting With Technology*. Sage, Thousand Oaks, CA, 1994.

Nadler, G. *Work Design.* Richard D. Irwin, Homewood, IL, 1963.

Orlikowski, W. Learning from Notes: Organizational Issues in Groupware Implementation. *The Information Society* 9, 237-250, 1993.

Orlikowski, W. Improvising Organizational Transformation Over Time: A Situated Change Perspective. *Information Systems Research* 7(1), 1996, 63-92.

Raymond, E., *The Cathedral and the Bazaar*, Available online at http//www.tuxedo.org/~esr/writings/, 1999.

Robey, D., Farrow, D. and Franz, C. Group Process and Conflict in Systems Development. *Management Science 15*(10), 1989 1172_1191.

Sawyer, S. Packaged Software: Implications of the Differences from Custom Approaches to Software Development. *European Journal of Information Systems, 9*(1), 2000, 1-18.

Sawyer, S. and Guinan, P., "Software Development: Processes and Performance," *IBM Systems Journal, 37*(4), 1998, 552-569.

Sawyer, S., Farber, J. and Spillers, R.. Supporting the social processes of software development teams. *Information Technology & People, 10*(1), 1997, 46-62.

Sproull, L. and Goodman, P. Technology and Organizations: Integration and Opportunities. in *Technology and Organizations*, P. Goodman& L. Sproull, (Eds.)**.** San Francisco: Jossey-Bass, 1989, 254-266.

Vessey, I., and Sravanapudi, P. CASE Tools as Collaborative Support Technologies. *Communications of the ACM, 38*(1), 1995, 83-95.

Wellman, B., Salaff, J., Dimitrova, D., Garton, L., Gulia, M., and Haythornthwaite, C. *Computer Networks as Social Networks: Collaborative Work, Telework and Virtual Community*. Univ. of Toronto, Toronto, Canada, 1996.
Wynn, E. and Novick, D. Relevance Conventions and Problem Boundaries in Work Redesign Teams. *Information Technology & People, 9*(2),1995, 61-80.

Zachary, G. Armed Truce: Software in the Age of Teams. *Information Technology & People*, 11(1), 1998, 62-65.

Zachary, G. *Showstopper: The breakneck race to create Windows-NT and the next generation at Microsoft.* New York: The Free Press, 1994.