Considering three software development team archetypes and their implications.

Software Development Teams

By Steve Sawyer

Software development is an impressively complex socio-technical activity. It requires people to interact with each other and with both the technical methods and computing technologies they use to perform their work [3]. Analytically, the social aspects of software development include how people interact, behave, and organize. Technical aspects of software development include the use of production methods, development techniques, and computing technologies. In practice, it is difficult to disentangle the way people do things from the methods, techniques, and computing technologies they use [6].

In this article, I focus on what we can learn about software development by focusing on the ways software developers organize. By taking this social perspective, I highlight how software production methods, techniques, and tools are enmeshed in and enacted through the structures and interactions of the professionals who work together to build software. A social perspective considers the team as the unit of analysis, seeing it as more than the aggregate of individual software developer's attributes and actions. A social perspective contrasts with production-focused views of software development; social action becomes the focal activity, not the by-product of a method's prescription.

More common perspectives on software development, such as the Software Engineering Institute's (SEI) Capability Maturity Model (CMM), focus on the means of production (methods, techniques, and/or tools): a techno-social approach. The People-Capability Maturity Model (P-CMM) complements this by explaining how people can best change their behavior to fit the CMM approach. Together, the CMM and P-CMM reflect the traditional production first, people second, approach. The fundamental question I pose here is: What can we learn about software development by privileging the social perspective?

To characterize this social perspective, I outline three archetypes of software development. Archetype here means an idealized form premised on an internally consistent set of assumptions. A software team archetype represents the often implicit beliefs about human behavior and work organization that structure the explicit activities of development. Here, I review and build on these archetypes as a means of providing guidance for practicing professionals relative to the organization of software development tasks and the selection of methods and tools.

Three Social Structure Archetypes of Software Development Teams

My premise is that there exist three generic archetypes of software development teams: sequential, group, and network (see Table 1); the sidebar describes the conceptual bases for these archetypes. Table 2 outlines the contrasts among these three social archetypes of software development, which are discussed in greater detail later. These archetypes help us to provide guidance and to understand the more practical issues of hybrid models—the approaches to software development that combine elements of the three social archetypes (see Table 3).

Archetype	Definition
Sequence	Software development is a production effort based on a linear set of discrete tasks. People work in specialized functions with formalized interactions across functions. People are valued for their particular specialized skills.
Group	Software development is seen as a combination of development and production where a set of discrete tasks may need to be repeated until the product is complete. Developers are organized into interdependent groups and are valued for both their particular skills and for their ability to work with others.
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, and tasks are tied to individuals (or small groups) whose participation is based on interaction. Group members are valued for what they can produce. This implies a complex network of ties between people and a hub-and-spoke management approach.

Table 1. Three archetypes of the social processes of software development.

Sequential Archetype

The sequence archetype enacts the belief that a good process leads to a good product. Software development is seen as a linear, task-driven, structured effort driven by a known and prespecified ordering of the requisite tasks. The social structure is set within the host organization's administrative scheme, hierarchical, role-based, and formalized. The task and role specialization help to reduce intra-functional discussion which, when needed, is done via formal channels. People's roles are task-specific, discrete, specialized, and identifiable. A prespecified task ordering further implies a prescriptive view of the production process.

Conceptual Bases of the Three Archetypes

The sequential team archetype of software development team social structure draws on the work design tradition in industrial engineering [3]. Work is seen as a set of discrete tasks that can be measured.

The group archetype draws its intellectual roots from theories of social psychology, such as "work redesign" [2]. Work redesign arose in response to such issues as personnel motivation, retention, and productivity that typically occur in a work design approach.

The network group archetype draws on concepts of social network theory [1, 4]. In this archetype a group of people is linked by the relative "strength" of the social ties among them. Work is seen as the use of these links to deliver and receive information; these uses both span and define tasks.

The social interactions in the sequence archetype of software development are based on concepts of control. That is, people's interactions are seen primarily as driven by the work they do. If this work can be measured and differences in performance analyzed, the roles can be transferred through training. This task/specialization orientation also suggests it is possible for one member to be replaced as needed by another person if they have the same functional level of skill.

Archetype Aspect	Sequence	Group	Network
Perspective	Process first	Process first	Product first
Belief mode	Control	Conflict	Interaction
Orientation	Prescriptive	Normative	Descriptive
View of task	Production	Production and Development	Development
Implied method	Linear and sequential	Iterative and sequential	Emergent and nonlinear
Tie to context	Prescribed boundary	Permeable boundary	Embedded
People's actions	Prescribed	Role and goal driven	Individual and linked
Examples	SDLC, SEI/CMM	Spiral, RAD, JAD	Open source, Chief programmer

Table 2. Aspects of the three archetypes of software development team structure.

The work emphasis is on embedding the required information in the work product and/or associated documents to pass on to each follow-on task. This means that if required inter-team and extra-team interactions can be defined and formalized they may be automated (a capital/labor substitution). The control orientation, formalized interactions among team members, and automation emphasis suggest there is little need for strong social bonds. Examples of the sequence archetype include the traditional waterfall model (such as the systems development life cycle, or SDLC), the CMM, and the SPICE approach.

Insight Archetype	Sequence	Group	Network
Team/Task Issues	Cohesion	Consensus	Contribution
Opportunities	Cross-training, Process management	Teamwork skills Conflict management	Evaluation Project management
Method Issues	Repeatability	Regulation	Reliability
Opportunities	Components Feedback mechanisms	Connections Iteration controls	Tasking Interdependencies
Tool Issues	Comprehension	Collaboration	Connection
Opportunities	Automation Process control	Shared tools Process support	Interoperability Interaction support

Table 3. Insights and opportunities.

Group Archetype

Group archetypes also focus on process-to-product orientation. In this archetype, software development is based on a set of predefined tasks that build on the collective skills and weaknesses of the group's members. Social structures in the group archetype are based on collaboration. The tasks are sequential but iterative, and there is explicit attention to process improvement by the members of the group. The group archetype also explicitly recognizes, in its iterative nature, that software development and production are often intimately linked. Thus, a group archetype is normative. Further, a group archetype implies the boundary between the team and the social context is permeable and that formal and informal boundary-spanning (for example, to key stakeholders) is important.

The group archetype makes explicit the need for social interaction. The group's rules and behaviors are designed to help resolve the inevitable conflicts that arise when people collaborate. There is the potential for some automation of production tasks, particularly for tools and methods that explicitly support and/or enable collaboration among the group members. Examples of group archetypes are the spiral/evolutionary approach to software development, rapid application development (RAD), and joint application development (JAD).

Network Archetype

The product is the central focus in the network archetype; production processes are secondary. The development effort takes shape through the network ties developed by the participants. The strength of these ties reflects the frequency and value derived from interaction. Further, this network is fully embedded in a larger social context that may not easily map to any organizational or geographic boundary. In the network archetype, the people's connections and the tasks they perform define the process. However, these constraints are a function of the social ties that create the social network. So, even if a process focus is not central, there is some form of version control of testing and of documentation. One belief underlying the network archetype is that a good product comes from having good people. This people-first approach recognizes that it is difficult (if not impossible) to replace key members of a network because they represent important hubs. These key people serve as the nodes that define the network.

To support the network, software development tools must provide for interconnection. Simply stated, any software development tool is valued for how it helps the individual member and/or for how

well it enables sharing among the network. A second implication is that a network archetype, being emergent, reflects a product development (as opposed to production process) view of software. Further, given the centrality of the social structures and individual members interaction in a network archetype, the effort is often contentious [11]. From this perspective, the interactions between the members are focused on product features, functions, or actions. There are few procedural details and the general expectation among the members is "show and tell." That is, the resolution of disagreements is often rooted in providing code that delivers on the concepts discussed.

The chief programmer team model of software development [1] is one example. The network structure is hub-and-spoke: strong ties between members and the chief programmer and weak ties between individual team members. The recent growth of open source software development efforts [7] reflects a second form of the networked group archetype. This network has multiple nodes (of varying importance) and multiple ties among many members of the network.

The empirical and philosophical question of which archetype or what hybrid blend is best is likely to have many viable responses.

Guidance for Practicing Professionals

The value of this archetypal frame is measured by the insights and guidance it provides to both scholars and practicing professionals. Here, I focus on the latter, and highlight issues with team/task arrangements, methods, and tools (see Table 3).

Sequence Guidance. Given the degree of task specialization and decoupling, it is often difficult for participants to see the value of their individual contribution to the whole. Limited interaction with other members often reduces the likelihood that project team cohesion will develop. Symptoms of this lack of cohesion are analysts not speaking with developers and testers remaining independent of the rest of the team. This suggests that cross-training personnel (for multiple roles) and more interaction through formal channels (such as cross-functional meetings and product walkthroughs) are important to developing stronger sequence teams. The sequence archetype is predicated on repeatable methods and a principle means of doing this is to break the project into many components. This puts pressure on maintaining standards and generating feedback, suggesting it is important to ascertain between-task adherence to standards. Mechanisms to do this include walkthroughs, checklists, and sign-offs. Walkthroughs can serve two purposes: to improve the team's clarity of purpose and a means for cross-assigning members of one task to be part of another. Since the sequence archetype is premised on routinization, automation is often a goal. This suggests that tool development should focus on embedding process control into an integrated development environment to reinforce method adherence.

Group Guidance. The group archetype focuses on team-member interaction and developing consensus among team members. Combined with the cyclic and integrated nature of production, this suggests that the task/team issue is to improve a member's team-working skills. Further, regulating the iterative nature of the project is important. That is, how do teams know when to stop iterating? One clear form of iteration control is money (or some other resource constraint), though others may be more valued such as user feedback or functional compliance assessments. Guidance for method development in the group archetype includes increasing cross-iteration task linkages such as change and version tracking, release control, and release planning. Group-based approaches demand tools that support team-member collaboration, which suggests developing shared tools that allow for group access. The proliferation of Lotus Notes databases to support software development teams exemplifies this attention.

Network Guidance. Given the interdependent nature of the work combined with the individualized nature of the way the work is done, evaluating contributions is often outcome or deliverable-based. Such an approach demands strong product management. Often this product management seems centralized in one person or a small number of people who act as hubs. Thus, using the network archetype, dispersing work seems to concentrate management. Control of the product is with a person (hub). And, while this control may shift over time, it is rarely shared. However, members of a network can often choose to leave the effort if their contributions are not being rewarded.

In the network archetype one method issue is repeatability: the effort to ensure a common process for repetitive activities (such as task assignment,

progress reporting, and issue tracking) is often problematic. While these are issues with all forms of software development, they are particularly central to the network archetype (and its reliance on social interaction as the dominant form of structure). One method opportunity for network archetypes is automated tasking mechanisms—in essence, a public project tracking board. Such a public tracking mechanism would also help to maintain and track the interdependencies among the members of the network. Another issue is the move to outcome measures such as "does your module run and does it interact with the larger product?" Outcome-oriented approaches are common aspects of the Microsoft models of development [5, 10]. The interdependent nature of work in a network archetype means the tools to support this approach must provide for interoperability and interaction. That is, from a network perspective, it must be easy (if not seamless) to share files and even to pass useful utilities and tools. This helps to explain why stable platforms (such as Unix/Linux) are the base for many open source development efforts [7].

Hybrid Models

A more typical scenario is that practicing software development teams will adopt a hybrid social structure, drawing elements from several archetypes. For example, both Baker [1] and Brooks [3] write about their IBM System/360 operating systems development effort. The sequential archetype 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 structure underlying the sequence of the SDLC. As a second example, Microsoft's development approach is a hybrid between a group and network archetype [4, 5, 10]. Guinan et al. [4] and Vessey and Sravanapudi [9] highlight how CASE tools are often designed to support sequence models but in their use they support group interactions.

The guidance to take from these examples of hybrid approaches to software development comes in three parts. First, it suggests the importance of aligning a software development team's tasks, production methods, and computing technologies. Second, it appears from this limited but illustrative sample of empirical work that focusing on alignment among tasks, methods, and technologies is likely to lead to increased attention to social and behavioral aspects of software development. Third, the importance of the social network that develops among developers suggests the need to develop network-sensitive methods and tools, and an equally focused effort to reduce the reliance on sequence-focused methods and tools.

Privileging the Social Perspective

The empirical and philosophical question of which archetype or what hybrid blend is best is likely to have many viable responses, demands more attention, and is certainly beyond the limited space of this article. I have provided evidence that a range of hybrid approaches exist and that these can be decomposed into some combination of the three base social structure archetypes. Further, I have demonstrated that focusing on the social structures of software development suggests that inattention to the social processes and structures often leads to mismatched selections of method and computing technologies. As I noted, for example, the social perspective makes clear the valued roles of a developer's social networks, something a production process focus often neglects. Moreover, the iterative nature and complexity of social interaction implied in hybrid models also suggest that software development tools and methods that enable collaboration and support production would be highly valued. C

References

1. Baker, F. Chief programmer team management of production programming. *IBM Systems Journal 11*, 1 (Jan. 1972), 56–73.
2. Bijker, W. *Of Bicycles, Bakelites and Bulbs: Toward a Theory of Sociotechnical Change.* MIT Press, Cambridge, MA, 1995.
3. Brooks, F. The mythical man-month. *Datamation* (1974), 44–52.
4. Carmel, E. and Sawyer, S. Packaged software development teams: What makes them different? *Information Technology and People 11*, 1 (Jan. 1998), 7–19.
5. Cusumano, M. and Selby, R. How Microsoft builds software. *Commun. ACM 40*, 6 (June 1997), 53–61.
6. Guinan, P., Cooprider, J., and Sawyer, S. The effective use of automated application development tools. *IBM Systems Journal 36*, 1 (Jan. 1997), 124–139.
7. Madey, G., Freeh, V., and Tynan, R. The open source software development phenomena: An analysis based on social network theory. In *Proceedings of the 8th Americas Conference on Information Systems*, 2002, 1806–1812.
8. Sawyer, S., Farber, J., and Spillers, R. Supporting the social processes of software development teams. *Information Technology and People 10*, 1 (Jan. 1997), 46–62.
9. Vessey, I. and Sravanapudi, P. CASE tools as collaborative support technologies. *Commun. ACM 38*, 1 (Jan. 1995), 83–95.
10. Zachary, G. *Showstopper: The Breakneck Race to Create Windows-NT and the Next Generation at Microsoft*. The Free Press, New York, 1994.

Steve Sawyer (sawyer@ist.psu.edu) is a founding member and an associate professor at Pennsylvania State University's School of Information Sciences and Technology in University Park, PA.