Anticipating Team Roles and Interactions When Planning a Software Development Project

Jeffrey H. Schweriner, Project Manager, Qwest Communications International, Inc.

Introduction

Most projects, no matter what the industry, require a team. Every team member has specified tasks designed to carry out the development and implementation of the project. But for *successful* projects, every team member not only performs their specified tasks, but members play a role as well. By definition, a role is defined as "a part or character to be played by an actor" (Random House 1979). Indeed, project roles are defined not by what a person does, but rather by what *part* they play or whom they act as.

Project roles become extremely important in highly technical projects such as software development. In software development, the nature of the work becomes more complex, and time constraints are frequently imposed. Although powerful programming languages and advanced engineering concepts are the available tools for producing good software, multiple developers are required to produce sophisticated software systems. How the human resources of programming are organized and managed becomes critical in the success or failure of a software development project. The goal of this paper is to identify a model by which a manager can anticipate and build a software development team, and how the roles and interactions of chosen team members can contribute to the success (or failure) of a software project.

Building the Software Development Team

Complex software development projects require critical decision-making in the upstream portion of the software development process, particularly during the specification and design phases. One of the most important upstream decisions is in building the development team. From a managerial perspective, many factors must be considered when building the team. For more complex projects, knowledge from multiple technical and functional domains is a necessity (Walz et al. 1993). Walz et al. claim that in an ideal situation, a software design team is staffed so that both the levels and distribution of knowledge within the team match those required for the successful completion of the project. Unfortunately, only under ideal circumstances can the above statement be applied. In the real world, a manager is often faced with the technical and knowledge shortfalls of team members. In most situations, individual team members must acquire additional information before accomplishing their set tasks. To bridge the knowledge gap, formal-training sessions, group meetings, and discussions may be frequently required. The most productive activities of the team come from the integration of knowledge. Walz et al. (1993) summarize as follows:

> A software design team seldom starts its life with shared models of the system to be built. Instead, these models develop over time as team members learn from one another about the expected behavior of the application and the computational structures required to produce this behavior. This means that team members need to be speaking the same *language* (or, at least, a dialect whose semantics are similar enough to facilitate communication and understanding) in order to share knowledge about the system.

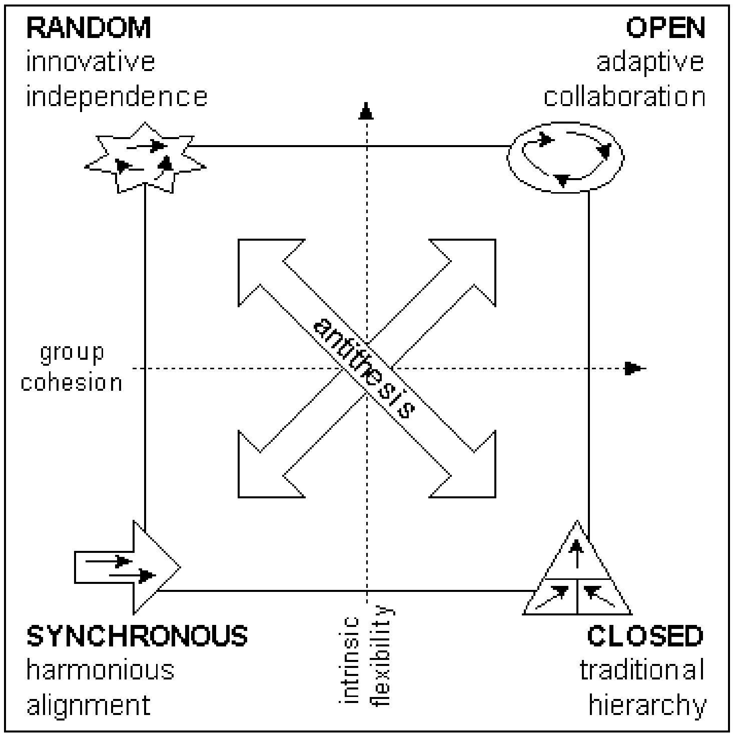
During the knowledge integration process, the manager is now faced with handling multiple, concurrent, and often conflicting goals. The manager needs to be able to leverage the specific strengths of individual people while still maintaining the uniformity and stability of the pending project.

It appears that although technical expertise and knowledge of participants are necessary for a successful software project, the actual "role" a team member plays becomes a critical factor in the interaction of team members, and how successful the team is in reaching project goals. In the team-building process, the manager is then faced with two very important questions: how do the group members interact to acquire, share, and integrate project relevant knowledge, and how do participation styles differ among team members?

Group Interaction Model

Several models have been published to aid managers understanding how group members interact to achieve project goals. One model, which has been directly applied to the software development process, is that of

Exhibit 1.



Constantine (1993). Constantine has devised a paradigmatic framework describing various types of groups. According to Constantine, this framework provides a model for "the organization and management of collective human activities" (1993). In other words, this model provides a general framework that can be applied toward building a software development team within the dynamics of the project, and within the corporate structure. Constantine sets the stage by observing that "everyday life affords us countless examples of groups of people carrying out a joint activity or tasks in a coordinated manner" (1993) To emphasize his point, he compares a family dinner to a software project. He points out that the common thread between the family dinner and the software project is that there is always "recognizably consistent patterns of interaction" (1993). From his observations, he aims to establish a conceptual framework for understanding how a group of people jointly involved in a common activity or goal carry it to completion.

Exhibit 1 is the paradigm that Constantine uses to describe how group members organize and coordinate daily activities.

The lower right corner of Exhibit 1 represents the traditional hierarchy or closed paradigm. Within this paradigm, continuity of a project is maintained through established standards and rules of operation. Stability is maintained by controlling for deviations from established norms and patterns. The purest example of a closed paradigm cited by Constantine is that of the military services; however, this type of paradigm may be applied to groups in a large corporate structure that repeatedly produce the same type of software.

On the complete opposite side of the model shown in Exhibit 1 (upper left) is the random paradigm, which is the antithesis of the closed paradigm. This paradigm heavily emphasizes innovation through creative autonomy of team members. By giving each team member the freedom to create and act independently, a manager can create a "breakthrough project team," which may lead to developing a new technology. This paradigm has been successfully applied by Hyman (1993) and will be discussed later.

The open paradigm (upper right corner) in Exhibit 1 is a synthesis of the closed and random paradigms. The open paradigm team is based on adaptive collaboration and the integrating of innovation while maintaining stability. Individuals with collective ideas and interests work together through negotiation and discussion. This type of team works best with small software development teams. Rettig and

Simons (1993) have published an excellent example of an open structured team that will be reviewed later in this paper.

The final paradigm in the model in Exhibit 1, opposite of the open paradigm, is the synchronous or harmonious model. This model is based on "harmonious and effortless coordination through the alignment of members with a common vision that reflects the collective." A synchronous or harmonious team is useful in simple projects, whereby each team member can program an independent piece of software, and the collective software completes the whole project.

From the various paradigms described in Constantine's model, a manager can begin to explore how effective project teams are established and how team roles fit within the organization of the project.

In a 1989 study, Larson and LaFasto interviewed team members to learn how the people who performed best within different kinds of teams were described. Constantine has reviewed Larson and LaFasto's conclusions, and has refined their results by incorporating his model coupled with his observations on programming teams:

> People who do best in closed paradigm (tactical) teams have been described as loyal, committed and action-oriented. They seem to have a strong sense of urgency and respond well to leadership. People who work best within the creative environment of a random (or breakthrough) team are independent thinkers, often artistic or intellectual. They are persevering self-starters who do not need orders to get going or close supervision to keep going. People who thrive in the collaborative consensus-building or open (problem-solving) teams are practical minded by sensitive to "people issues." They have integrity and are seen as trustworthy by peers, exhibiting intelligence coupled with good interpersonal skills. It appears that those who fit well in strongly synchronous teams are intuitive, somewhat introverted, yet people sensitive. They are good at linking the larger picture to specific action and work with quiet efficiency.

Constantine's model provides a manager with a more rational approach to building a team. It seems that each paradigm "reinforces and is reinforced by" the behaviors of individual team members.

Hyman (1993) has effectively applied Constantine's "random team" paradigm to her software development project. The Software Reusability Department of ARINC Research Corporation, Annapolis, Maryland, USA, began as a five-member team with a goal to create reusable libraries of software components. At the onset of the project, Hyman surmised that there was not enough budget, time, or resources to meet all of the project objectives. From this, she realistically stated that "order and simplicity are not the norm, chaos and complexity are the rule" (1993) To reach her project goals, she decided to "capitalize" on chaos through the individual talents of the team members. Each member was given the freedom to explore new technologies in order to solve complex problems. Hyman uses the term "innovative anarchy" to describe the dynamic of the project team:

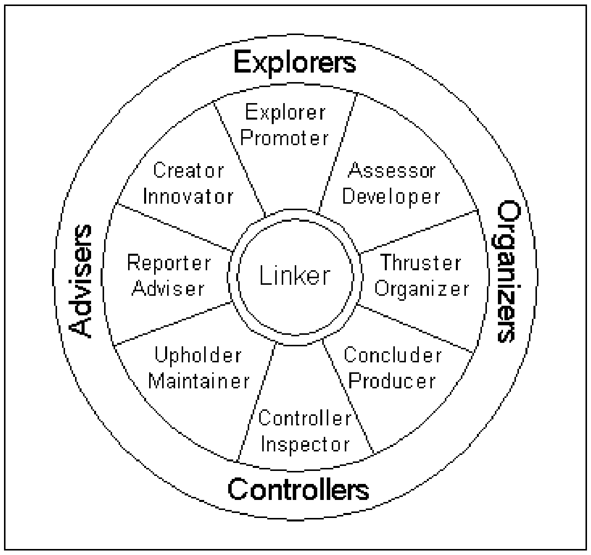
> This description [creative anarchy] fit our creative chaos in which there are no fixed roles, a strong sense of individuality, and problem solutions based on innovation and creativity. It was an exciting and fun work environment that amplified creativity and brought out the best in individuals.

The original five-person team relied heavily on creative independence, individual initiative, integrity, and trust. However, the team members possessed various attributes, which helped them maintain their roles as individuals while still working cohesively to achieve the project goals. Hyman notes that members of the team were "technically gifted, high-energy, self motivated risk takers." Individual goals and team goals were closely aligned, allowing for the reinforcement of individual achievements through successful participation.

It is worthwhile to note that as Hyman's team grew exponentially, the stability and cohesiveness of the team began to weaken. As new team members were added to the group, their "new perspectives" disrupted the harmony developed by the original group (1993). In software development, the size of the team is of utmost importance. According to Robbins, smaller teams lend themselves to an increased accountability on the part of the members (1989). When describing small teams, McCarthy cites that this increased accountability allows team members to develop a higher identity within the team—an identity that helps to enhance their role in the team (1995). As discussed next, Rettig and Simons were able to succeed in their project by maintaining small teams (1993). In order to avoid team failure in a software development project, a manager must build small teams (ten people or less) to maintain the cohesiveness among individual team members.

Rettig and Simons have directly applied Constantine's "open structured team" paradigm to a software development project (1993). This paradigm was successfully used by the Academic Computing department of the Summer Institute of Linguistics (SIL) in Dallas, Texas, USA. The goal of its project was to develop a computer system that would facilitate software applications that deal with large

Exhibit 2.



collections of structured multilingual information. It began to organize its software development team at the start of this five-year development project. Early in the formation of the team, roles were specifically assigned to team members. The assigned roles were as follows:

- Facilitator: schedule and lead team meetings.
- Archivist: Take minutes of team meetings and maintain an archive of project documents.
- Manager: Maintain a chart of progress toward project milestones to give the project team feedback on progress toward goals, to assist the project team in planning activities.
- Contact: Serve as the contact point between the project team and the guidance team.

Once the set roles were established, Rettig and Simons were able to observe the effectiveness of these set roles (1993). If, during meetings, various members did not seem to fit the assigned roles, the team members were allowed to rotate into other roles. With time, the set roles tended to become habits, and the role assignments became less formal. For example: during a meeting, if one member is observed to be taking notes (archivist), he may be asked to send everyone a copy. Rettig and Simons believe that the attention to interpersonal communication based on an open structured team became an important "ingredient" in the early successes of their project (1993).

There is an interesting similarity between the roles defined by Rettig and Simons, and those of Margerison and McCann in their 1990 book, *Team Management: Practical New Approaches*. Margerison and McCann develop a basic model of an integrated team and identify nine key team roles in the their study on team management.

Margerison and McCann discuss nine key roles as part of their Team Management Wheel (see Exhibit 2). There are eight sections of the wheel corresponding to eight different roles or work preferences, which are grouped into four areas. Standing out among the other roles is that of a *linker*, which can be most often associated with a Project Manager. Note that each role has a two word description, e.g. *reporter-advisor*. The first word is used to describe a behavior description and the second word is used to describe the work function. As Margerison and McCann describe in their book:

- Advisors enjoy reporting data.
- Innovators enjoy creating ideas.
- Promoters enjoy exploring opportunities.
- Developers enjoy assessing plans.
- Organizers enjoy thrusting into action.
- Producers enjoy concluding tasks.
- Inspectors enjoy controlling procedures.
- Maintainers enjoy upholding standards.

Each role is worth further discussion.

Producer

Provides direction and follow through. These people enjoy working to set procedures.

Creator

Initiates creative ideas. The role of creator provides innovation and often needs to be managed in such a way that the creator is not constrained by the corporate structure.

Controller

Examines details and enforces rules. Controllers enjoy doing detailed work and checking facts and figures.

Organizer

Provides structure. This person may be the project manager, the one to set up procedures and systems. An organizer will set up the process to ensure that deadlines are met.

Adviser

Encourages the search for more information. This may not necessarily be an immediate team member, but rather an experienced "old-timer" who can advise the team (which, in software development, may consist of younger, less-experienced people) on the right way to move forward.

Assessor

Offers insightful analysis of options. In software, an assessor may also be called an "information architect," a designer who puts practical application to the ideas of the initial creative team.

Maintainer

Fights external battles. An account executive for a software team often plays the role of a maintainer, someone who will defend the team against outside criticism and conflict.

Promoter

Champions ideas after they're initiated. These people play a key role, exploring new ways of doing things outside the organization. In software development, this person might investigate new development tools.

Linker

Coordinates and integrates. This is the role that best describes the project manager in software development. In their book, Margerison and McCann describe eleven key linking skills: 1) active listening, 2) communication, 3) problem solving and counseling, 4) team development, 5) work allocation, 6) team relationships, 7) delegation, 8) quality standards, 9) objectives setting, 10) interface management, and 11) participative decision-making.

In 1997 and 1998, I was involved in a software development project that Margerison and McCann might describe as a textbook example of how critical team roles are. This project was the development of an educational, historical CD-ROM. This project had several challenges from the onset:

- The content of the CD-ROM was of a sensitive nature that required great scrutiny from outside reviewers.
- Development demanded a wide range of skills and disciplines.
- The development team was large—the core internal team consisted of eight people, with up to thirty more people assisting throughout the various stages of development.
- Accepted as a new job during a slight downturn in business, the project was not looked upon as favorably as business began to pick up.

In retrospect, key team roles were associated with our corporate titles. Had these roles not been filled, the project would not have achieved the success that it did:

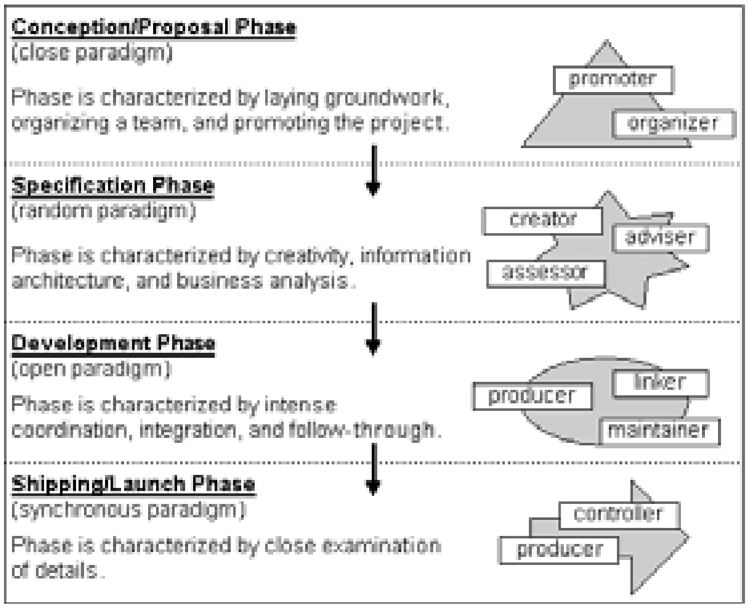
- Director of Project Management (Assessor and Organizer): This person set up the team structure, and defined the project boundaries.
- Creative Director (Creator and Promoter): Our Creative Director was brilliant in initiating ideas and finding new, exciting ways of presenting historical, and often dry, material.
- Lead Historian and Researchers (Advisor): This person, outside the corporation, encouraged the search for more information, ensuring the historical integrity of the CD-ROM.
- Account Executive (Maintainer): Kept the corporation feeling comfortable about the project and kept a nervous client feeling good about the project.
- Project Manager (Linker): This role was critical in ensuring that all the various groups (artists, programmers, writers, researchers, QA, and producers) were working together efficiently and effectively.
- Production Coordinator and Quality Assurance (Controllers): The CD-ROM contained an enormous number of facts, figures, names, and photographs. Detailed-oriented people were an absolute necessity, especially during shipping.

Although Margerison and McCann's team model has not been defined specifically for software development, it seems that the roles defined in their model can directly be applied to building teams within the software industry. To show how this may be accomplished, I have reorganized the roles described by Margerison and McCann (1990) into the framework provided by Constantine (1993). Software projects are often too complex to consider as a uniform model. To aid in future project planning, I have developed a model whereby each of the Constantine paradigms can be applied to development teams during each phase of a software project, as depicted in Exhibit 3.

Conception/Proposal Phase

This phase is characterized by laying the groundwork for the project, organizing a team, and promoting the project.

Exhibit 3.


Conception/Proposal Phase
(close paradigm)
Phase is characterized by laying groundwork, organizing a team, and promoting the project.
promoter
organizer
Specification Phase
(random paradigm)
Phase is characterized by creativity, information architecture, and business analysis.
creator
adviser
assessor
Development Phase
(open paradigm)
Phase is characterized by intense coordination, integration, and follow-through.
producer
linker
maintainer
Shipping/Launch Phase
(synchronous paradigm)
Phase is characterized by close examination of details.
controller
producer

Rules are enforced, standards are set, and parameters are established.

Specification Phase

This phase emphasizes innovation through creative autonomy of team members. As stated earlier, by giving each team member the freedom to create and act independently, a manager can create a "breakthrough project team," which may lead to developing a new technology.

Development Phase

"Adaptive collaboration and the integrating of innovation while maintaining stability" can characterize this phase. The intense coordination and integration of development demand leadership from the project manager and production coordinator in linker and producer-type roles.

Shipping/Launch Phase

Shipping, the final phase of software development, closely resembles a synchronous model, where the team is aligned toward a common vision—launching the product. The dominant responsibilities during this phase are that of quality assurance, in a controller or producer-type role.

This superimposed model is a comprehensive one for developing team roles in a software development project. This model considers the strengths and knowledge of each team member and provides a structured paradigm for each phase of the development process.

Conclusion

As reviewed in this paper, a project manager has several models to chose from in organizing a software development team at the upstream portion of a project. Additionally, it is clear that successful software teams are more than just the sum of their team members performing their assigned duties—team members also play important roles that allow the team to function as a dynamic unit. In addition, team members (in small teams) will gravitate toward the roles that fit their personalities.

Several guidebooks and publications can aid managers in forming a strong software development team. Margerison and McCann recommend a questionnaire for determining who on a team is best suited for a particular role. Their Team Management Index poses a series of questions (based on the principles of the Myers-Briggs personality test) to gather information on the personal strengths and weaknesses of each prospective team member.

In addition, section 9.1 of *A Guide to the Project Management Body of Knowledge (PMBOK™ Guide)* discusses Organizational Planning of which identifying team roles is a part. A project manager should consider what roles will be required and what roles will be dominant during the different phases of the project. Depending on how "responsibility-oriented" the roles on a project are, the manager may decide not to document the role assignments. For example, a manager may decide to recruit a certain individual in the organization to play the role of "Adviser." This decision may be more appropriate for one's own personal journal than for a project plan. While it may take considerable effort to plan for project team roles, it can make the difference between a mediocre and high-performance software development team**.**

References

Constantine, Larry L. 1993 Work Organization: Paradigms for Project Management and Organization. *Communications of the ACM* 36: 35–43.

Hyman, Risa B. 1993. Creative Chaos in High Performance Teams: An Experience Report. *Communications of_the ACM* 36: 57–60.

Larson, C. E., and F.M. J. LaFasto. *Teamwork: What Must Go Right/What Can Go Wrong*. Newbury Park, CA: Sage, 1989.

Margerison C., and D. McCann. 1990. *Team Management: Practical New Approaches*. London: Mercury Books.

McCarthy, Jim. 1995. *The Dynamics of Software Development*. Microsoft Press: Redmond CA.

Random House, Inc. 1979. *The Random House College Dictionary*.

Rettig, Marc, and Gary Simons. 1993. A Project Planning and Development Process for Small Teams. *Communications of the ACM* 36: 45–55.

Robbins, Stephen P. 1998. *Organizational Behavior*. Prentice Hall: Upper Saddle River. NJ.

Walz, D. B., J. J. Elam, and B. Curtis. 1993. Inside a Software Design Team: Knowledge Acquisition, Sharing, and Integration. *Communications of the ACM*