ABSTRACT

Empirical research has shown that social network structure is a critical success factor for various kinds of work groups. The authors extended this research to a new type of work group—the open source software project community—with the objective of exploring the role of communication networks within these intriguing projects. Using archival data from 143 open source project groups, the authors compiled six measures of social network structure and analyzed these in relation to four measures of group success. This study found that the social network structures of these project communities did not appear to be critical success factors at all, but rather they had no significant impact on success or their effect was opposite of that seen in prior studies of work groups. Various conjectures were suggested that might explain these results, offering opportunities for further research.

Keywords: Open Source Software, Social Capital Theory, Social Network Analysis, Success Factors, Teams, Work Groups

INTRODUCTION

In the Art History (Kunsthistorische) Museum in Vienna hangs a famous painting by Pieter Breughel (1563), his vision of the Tower of Babel, supposedly built around 2000 B.C. According to the Biblical account, construction on the tower failed when God scrambled up the languages of the workers. They could no longer coordinate. Work groups, then as now, depend on effective communication for their success. This seems to be especially true of software projects (Figure 1).

Back in the 1970’s, Fred Brooks, project manager for what was then the largest software project ever, the operating system for the IBM 360, wrote a classic book, The Mythical Man Month (1975/1995). In it, he formulated Brook’s Law: “adding manpower to a late software project makes it later”. This has two aspects. One is the learning curve of new staff: it takes time to learn the complexities of the project. The other is the combinatorics of communication. The number of different communication chan-
A key factor for project management of any kind is the complexity of the artifact being produced. Brooks (1986) claims that large software systems are among the most complex: “Software entities are more complex for their size than perhaps any other human construct because no two parts are alike (at least above the statement level). If they are, we make the two similar parts into a subroutine—open or closed. … The complexity of software is an essential property, not an accidental one” (Brooks, 1986). As one would expect, composite systems are correspondingly more complex: “A programming systems product takes about nine times as much effort as the component programs written separately for private use” (1995, Ch1).

Brooks, and many others after him, have argued that the only way around the software productivity bottleneck is improved design: “My rule of thumb is 1/3 of the schedule for design, 1/6 for coding, 1/4 for component testing, and 1/4 for system testing” (Brooks, 1995).

Recent research, for example Pendlharkar and Rodger (2009), indicates that current software work still encounters many of the same communication difficulties as in the 1970’s. However, they note that these may be mitigated by improved methods such as the use of CASE tools. Paulley (2009) notes that these gains may nonetheless be offset by the added communication difficulties of geographically dispersed software development teams.

Against the backdrop of conventional software project management, the notion that software might be developed based on an ad-hoc, evolutionary style framework seems implausible. Eric Raymond (1999b) comments:

In The Mythical Man-Month, Fred Brooks ... argued that the complexity and communication costs of a project rise with the square of the number of developers, while work done only rises linearly. Brooks’s Law has been widely regarded as a truism. But … the process of open-source development falsifies the assumptions behind it—and, empirically, if Brooks’s Law were the whole picture Linux would be impossible.

This is a remarkable observation. Is it really the case that open source projects do not
Related Content

An Empirical Comparison of Machine Learning Techniques in Predicting the Bug Severity of Open and Closed Source Projects
[www.igi-global.com/article/empirical-comparison-machine-learning-techniques/78560?camid=4v1a](www.igi-global.com/article/empirical-comparison-machine-learning-techniques/78560?camid=4v1a)

Rethinking Expertise in the Web 2.0 Era: Lessons Learned from Project Durian
[www.igi-global.com/chapter/rethinking-expertise-in-the-web-20-era/121012?camid=4v1a](www.igi-global.com/chapter/rethinking-expertise-in-the-web-20-era/121012?camid=4v1a)

Helping to Bridge the Digital Divide with Free Software and Services
[www.igi-global.com/article/helping-bridge-digital-divide-free/53875?camid=4v1a](www.igi-global.com/article/helping-bridge-digital-divide-free/53875?camid=4v1a)
Understanding the Development of Free E-Commerce/E-Business Software: A Resource-Based View
www.igi-global.com/chapter/understanding-development-free-commerce-business/10087?camid=4v1a