SOFTWARE DEVELOPMENT PRODUCTIVITY: CONSIDERING THE SOCIO-TECHNICAL SIDE OF SOFTWARE DEVELOPMENT

Tyson R. Henry, California State University Chico, tyson@ecst.csuchico.edu

ABSTRACT

Software engineering has long promised to improve software development productivity. However, nearly four decades after the birth of software engineering, software projects still fail to meet deadlines at an alarming rate. One reason the promised increase in productivity has not been achieved is the lack of attention to the sociological and psychological aspects of software engineering (called socio-technical software engineering). In order to fully understand software development, we must consider the complete picture. This includes the software engineering process, tools, business rules, organizational structure, management methods, politics, work environments, ***and*** *the socio-technical aspects. This paper first establishes the importance of socio-technical software engineering. Then it presents an extensive review of previous socio-technical research. Finally, it presents directions for future research on socio-technical software engineering.*

Keywords: Software Engineering; Socio-technical Software Engineering; Empirical Studies

INTRODUCTION

Software engineering is a socio-technical activity that depends on a long list of human issues: communication, collaboration, motivation, work environment, team harmony, sense of purpose, engagement, training, education, etc. These issues affect everything about software engineering, including the adoption and implementation of software engineering processes and technologies. A high level of software development productivity cannot be achieved until the socio-technical aspects of software engineering are well understood and integrated into the software engineering process. Unfortunately, it is difficult to study the socio-technical aspects of software engineering. Many software engineering researchers believe the socio-technical issues are important (see next section), but most would rather develop new technologies and processes than attempt the complex process of studying the people that perform the work.

THE NEED FOR SOCIO-TECHNICAL SOFTWARE ENGINEERING

The literature contains many mentions of the importance of the human side of software engineering. For example, Brooks describes software engineering as “an exercise in complex [human] interrelationships [6].” Boehm gives credit to the people, “Good people, with good skills and good judgment, are what makes projects work [4].” DeMarco wrote, “It is not just thing One that needs to be concerned with project sociology; it's also things Two through Two Hundred [11].” Humphrey calls people an organization's “most important asset [19].” Linberg analyzed project management literature and found the success factors of software projects include effective leadership, conductive organizational climate, and a diverse and synergistic team [20]. Perry et al. believe that “the elements of the organization are as important as technology, if not more so [24].” Glass calls it a fundamental software engineering fact that programmers are more important than the tools and techniques [14]. The most important

practices listed in Paulk's summary of the practices of high maturity organizations are socio-technical [23]. Curtis et al. point out that since software is created by humans rather than machines, "their creation must be analyzed as a behavioral process [10]."

Even though many have mentioned it, the socio-technical side of software engineering has mostly been ignored by main-stream researchers. In a 2001 study of 369 software engineering papers, it was found that software engineering papers are fundamentally about technical issues and seldom address behavioral issues [15]. The Bourque et al. report on the ACM/IEEE Computer Society Software Engineering Body of Knowledge (SWEBOK) project does not make any mention of the human side of software engineering [5]. Widely used software engineering textbooks have little or no mention of socio-technical issues [25, 26].

Glass points out that while nearly everyone agrees that socio-technical issues are important, the majority of software engineering researchers are simply paying lip service. He suggests that even though the people issues are the most important, they are being ignored because it is easier to invent new tools, techniques, and processes [16]. He suggests that research on the accepted *facts* of software engineering is long over due [17]. Zvegintzov suggests that the tasks performed by software engineers on a daily basis need to be better understood. Decisions about method, management, quality and process/method improvement rely on the understanding of what software engineers do [37]. Thomas advocates more research into the human side of software quality [34]. Ott et al. suggest that we should identify ways to learn from the human aspect rather than ignoring it [22]. Tichy argues that experiments are central to the scientific process and thus should be a part of computer science. "Only experiments can bring new phenomena to light so that theories can be formulated and corrected [35]."

SURVEY OF PREVIOUS WORK

There is a growing body of empirical software engineering that touches on socio-technical issues, but it is very disjointed and hard to find. One of the oldest studies was performed by Grant and Sackman in 1966. They performed a timed debugging experiment with 12 programmers and found that the best of these 12 programmers was 28 times faster at debugging than the slowest [29]. Even though the authors cautioned that the study was too small to be considered decisive, and others have performed similar studies resulting in lower ratios, the 28:1 ratio has become a *fact* (or myth depending on your point of view) in software engineering. There are three problems with this type of work. First, simple results are often taken to be absolute truths and become a never questioned *fact*. Second, the results are too simple to be useful. The results from these studies provide simple quantitative data rather than qualitative data. In the book *The Hitchhiker's Guide to the Galaxy*, the answer to the question, "What is the meaning of life?" was calculated to be 42 by a futuristic computer [2]. Much like the programming productivity ratio of 28:1, 42 is too simple an answer. Third, software engineers do more than write code, and they very rarely write programs in isolation and without formal processes. Socio-technical research needs to have a broader scope and look for more complex answers. The remainder of this section provides a summary of several such studies.

Berander and Wohlin performed a case study to find the key success factors in process management. They conducted their study at a 700 person company, and used interviews to gather qualitative data and questionnaires to gather quantitative data. They identified six key areas for successful process management: baselining the current way of working, synchronization between

processes, user involvement, management commitment, change management, and documentation. Process synchronization was the most important [3].

Sousa et al. performed an exploratory informal field study to determine how the interdependencies that occur during the software development process are managed. They studied 31 co-located software engineers and found that both formal and informal practices had been adopted by the team. The formal practices included the software development process, software development tools, design meetings, and a clear division of labor. The informal approaches identified were partial code check-ins, problem reports that cross work boundaries, holding onto check-ins, e-mail, naming conventions, and the deliberate speeding up of the process [13].

Aaen et al. performed a questionnaire-based study of the perceived impact of CASE tools. He found that the only factor that could be attributed to the tool was response time. The remaining factors were sociological in nature. He found that the perceived impact of a CASE tool is related to organizational complexities, programmer experience, and organizational commitment [1].

Rainer et al. performed a study to explore what level of evidence software developers need in order to buy into a software process improvement. They conducted 42 focus group discussions with participants from 13 companies. Their primary finding was that there is an apparent contradiction between the evidence developers say will satisfy them, and the evidence they will actually accept. This is an important finding because without the buy-in of developers, there is little hope of implementing any process improvements [28]. Mahaney and Lederer conducted structured interviews with 12 IS project managers to see if agency theory (agency theory explains problems that occur when one party hires another party to perform work) could be used to predict project outcomes. Their findings suggest that changes in management procedure could reduce the failure rate of development projects [21].

Linberg performed a case study of a project group of 13 developers to explore three questions: (1) How do software developers define software development project success and failure? (2) How is software developer job satisfaction affected by software development project failure? (3) What was the impact of individual software developer temperament styles on software development project failure? He found that there may be a large gap between how the software industry defines success and how software developers define success and also that job satisfaction did not seem to be associated with meeting the organization's schedule or cost goals. In addition, he found that programmer temperament may have explained some team dynamics, but it is unlikely to have affected the outcome of the project [20].

Procaccino and Verner did a questionnaire-based study to determine what 31 software practitioners (programmers, analysis, network engineers, db administrators, management, and team leaders) perceive as project success. They found practitioners considered personal factors associated with the work and customer/user factors to be the main indicators of success. This is in stark contrast to the typical measures of success: budget, schedule, and requirements [27]. Solingen et al. used a structured questionnaire-based study to measure the frequency and effects of interruptions in a software engineering environment. They found that although interruptions often result in advancing the project, when and how interruptions occur can determine the extent of its negative impact. In addition, they found that interrupt awareness can lead to both a reduction in interrupts and more efficient handling of interrupts [32].

Wu et al. studied collaboration in software design. Their goal was to identify common activities that are not well supported by tools. They used a combination of methods for collecting data: shadowing (25 hours), interviews (17 subjects), and communication event logging (18 subjects). Their contributions include the development of a novel PDA approach for communication event logging. They also identified several collaboration behavior patterns: (1) a considerable amount of an individual's time was spent in communication, (2) team members communicated frequently and extensively, (3) the nature of communications changed in terms of synchronicity, location, and modality, and (4) face-to-face interactions were strongly preferred. Finally, they found that designers preferred to use general purpose tools for design and communication [36].

Tang performed a socio-technical experiment in a laboratory setting. He videotaped eight groups of three to four people working together to perform a shared drawing task. His observations relevant to software engineering tools were: (1) tools should support concurrent access, (2) tools should support modeless operations between different actions, and (3) spatial orientation is important, thus tools should support a common view for participants [33].

Singer and Lethbridge provide a good example of applying a socio-technical study by using the results to design new tools. In order to provide new tools to help a specific group of software engineers maintain their system more effectively, they used a four-fold approach to study work practices: web questionnaire, shadowing of an experienced software engineer new to the project, studied the entire group, and collected company-wide tool usage statistics. The questionnaire study was used to gain a rough understanding of how software engineers spend their time. One individual was intensively shadowed to gain a more detailed understanding of how time is spent. The group study included interviewing and shadowing eight more individuals. Finally, one week's worth of tool usage statistics were gathered from the company's tool group. These studies revealed two primary activities: search and navigation. They shadowed an additional engineer using two observers to gain a better understanding of the components of the search and navigation tasks. Their findings were that four search and navigation tasks were very common: searching using grep, saving the results of a search, suspending a search to perform a new search, and jumping back and forth between using grep and an editor. They used their results to build a search and navigation tool that fit the current work practices of the engineers [31].

Perry et al. performed two people-oriented experiments to categorize how programmers spend their time. In the first experiment, they had 13 programmers fill out a modified time card (called a time diary) to record how they spent their time. In the second experiment, they directly observed five of the people in the first experiment to calibrate and validate the time diaries. Among their findings, they measured that a large percentage of programmers' time was spent on the socio-technical aspects of the project. Much of this time was spent in some form of communication [24].

Seaman and Basili performed an empirical study to build understanding of communication among members of a software development group. They used semi-structured interviews, observations, and analysis of official documents. They observed 23 code inspection meetings which included a total of 20 different technical leads and developers, and performed semi-structured interviews with the participants. Their results indicate that inspection participants who are *close* (in terms of organizational distance, physical distance, or familiarity) spend less time discussing global issues. In this study, it was also the case that developers will report fewer defects in material authored or inspected by other developers with whom they are close. Since the actual number of defects in the

code was not known, it is not possible to say conclusively that inspectors will always find fewer defects in material authored by developers with whom they are close, but it certainly illuminates a socio-technical aspect of code inspections that is not intuitive [30].

RESEARCHING SOCIO-TECHNICAL SOFTWARE ENGINEERING

Since people are involved, socio-technical software engineering research must continue to be based on empirical studies. Given that people are so variable and that there are so many socio-technical factors, we will need a large body of results to be able to determine what relationships actually exist [9]. Furthermore, unlike the previous work which is nearly all based on very small sample sets, future research will need to include larger data sets—in medical research large data sets are required and conclusions are never based on studies of a couple dozen subjects.

Devising studies that include large populations is going to be challenging. Ideally we would like to systematically sample the entire population of software projects. Unfortunately, it is unlikely that statistical sampling will work. Research will have to continue relying on volunteers. Finding volunteers will be difficult, however there have been some encouraging findings. Hoch et al. interviewed 450 top executives from 100 software companies in order to uncover the secrets of software success. They initially planned to look at 60 companies, but so many companies wanted to volunteer that they expanded to 100 companies. The reason given was that the executives were so anxious to learn the results of the research that they were willing to donate their time and participate [18]. Perry et al. also found that programmers are willing to be observed and measured if the proper precautions are taken [24]. Motivating people so they participate initially and throughout the software development process will be a difficult task. The key will be to provide something of value to the participants. In Hoch's study, the participants felt that the results of the study provided value. Researchers will have to provide enough value throughout the study to keep participants involved.

Much like the bodies of knowledge in psychology and sociology, the body of knowledge in socio-technical software engineering will grow slowly. Researchers must be very careful not to jump to early conclusions. It will be difficult to develop analysis techniques that provide solid evidence to build our understanding. Analysis techniques will have to be borrowed from the psychology, sociology, and medical research communities.

There are countless aspects of socio-technical software engineering. It will be a challenge to decide which areas are the most important and should be studied first. Our preliminary analysis of the literature has shown that the most often referenced socio-technical areas of software engineering are: collaboration (often called teamwork), communication, management practices, and office environment [12,7,8,23].

CONCLUSION

The implicit promise of software engineering is that a well developed systematic process would allow any group of people to efficiently develop any software system in any corporate environment. Clearly this is not true. In addition to research on processes, tools, and techniques, software engineering researchers need to focus attention on the complete picture; they must include the human side of software engineering.

The potential benefits of socio-technical software engineering research are tremendous. Software engineering processes could be modified so they are more readily accepted, more effective, and easier to implement. CASE tools that integrate technical and socio-technical tasks could be developed. Business rules, organizational structure, and management methods could be updated to allow for greater productivity. Work environments could be modified to facilitate software engineering instead of inhibiting it. Verbal, written, and on-line communications could be used more effectively. People could receive effective education and training. Ultimately, programmers could become more productive.

The software engineering research community is aware of the need to understand the socio-technical aspects of software engineering; however, few researchers are pursuing this line of research. This paper motivated developing empirical studies to expand our understanding of the socio-technical aspects of software engineering and provided a survey of existing work. Most software engineering researchers are trained to create and apply software development technologies and processes, not to study the sociology and psychology of software engineering. We will need to borrow tools, techniques, and researchers from psychology and sociology to advance our understanding of the complete software engineering process.

REFERENCES

1. Aaen, I., (1993, July). CASE User Satisfaction-Impact Evaluations in User Organizations, *Proc. Sixth International Workshop on Computer-Aided Software Engineering.*
2. Adams, D. (1989). *The Hitchhiker's Guide to the Galaxy*, Harmony Books, London, ENG.
3. Berander, P. & Wohlin, C. (2003, September). Identification of Key Factors in Software Process Management - a Case Study, *Proc. International Symposium on Empirical Software Engineering*.
4. Boehm, B. W. (1991, January). Software Risk Management: Principles and Practices, *IEEE Software, 8*, 32-41.
5. Bourque, P., Dupuis, R., Abran, A., Moore, J. W. & Tripp, L. (1999, Fall). The Guide to the Software Engineering Body of Knowledge, *IEEE Software, 16*, 35-44.
6. Brooks, F. B. (1995). *The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition*, Addison-Wesley.
7. Constantine, L. (1995). *Constantine on Peopleware*, Prentice Hall, Upper Saddle River, NJ.
8. Constantine, L. (2001). *Beyond Chaos: The Expert Edge in Managing Software Development*, Addison-Wesley, NY, NY.
9. Courtney, R. E. & Gustafson, D. A. (1993, January). Shotgun Correlations in Software Measures, *Software Engineering Journal, 8*, 5-13.
10. Curtis, B., Krasner, H. & Iscoe, N. (1988, November). A Field Study of the Software Design Process for Large Systems, *Communications of the ACM, 31*, 1268-1287.
11. DeMarco, T. (1991, May). Non-Technological issues in software engineering, *Proc. 13th International Conference on Software Engineering*, 13-16.
12. DeMarco, T. & Lister, T. (1999). *Peopleware: Productive Projects and Teams*, Dorset House, NY, NY.
13. de Souza, C. R. B., Redmiles, D., Mark, G., Penix, J. & Sierhuis, M. (2003, September). Management of Interdependencies in Collaborative Software Development, *Proc. International Symposium on Empirical Software Engineering*.
14. Glass, R. L. (2001, May/June). Frequently Forgotten Fundamental Facts about Software Engineering, *IEEE Software, 18*, 112-111.

15. Glass, R.L., Vessey, I. & Ramesh, V., (2002, June). Research in Software Engineering: an Analysis of the Literature, *Information and Software Technology, 44*, 491-506.
16. Glass, R. L. (2003). *Facts and Fallacies of Software Engineering*, Addison-Wesley, NY, NY.
17. Glass, R. L. (2003, May/June). Questioning the Software Engineering Unquestionables, *IEEE Software*, *20*, 119-120.
18. Hoch, D. J., Roeding, C. R., Purkert, G. & Lindner, S. K. (2000). *Secrets of Software Success*, Harvard Business School Press, Boston, MA.
19. Humphrey, W. S. (1997). *Introduction to the Personal Software Process*, Addison-Wesley, NY, NY.
20. Linberg, K. R. (1999). Software Developer Perceptions about Software Project Failure: a Case Study, *Journal of Systems and Software*, *49*, 177-192.
21. Mahaney, R. C. (2003, October). Information Systems Project Management: An Agency Theory Interpretation, *Journal of Systems and Software*, *68*, 1-9.
22. Ott, L. M., Kinnula, A., Seaman, C. & Wohlin, C. (1999). The Role of Empirical Studies in Process Improvement, *Empirical Software Engineering*, *4*, 381-386.
23. Paulk, M. (1999, March). Practices of High Maturity Organizations, *Proc. Software Engineering Process Group Conference*, 1-13.
24. Perry, D. E., Staudenmayer, N. A. & Votta, L. G. (1994, July). People, Organizations, and Process Improvement, *IEEE Software*, *11*, 36-45.
25. Pfleeger, S. L. (2001). *Software Engineering Theory and Practice*, Prentice Hall, Saddle River, NJ.
26. Pressman, R. (2001). *Software Engineering a Practitioner's Approach*, McGraw Hill, NY.
27. Procaccino, J. D. & Verner, J. M. (2002, January). Software Practitioner's Perception of Project Success: a Pilot Study, *Intl Journal of the Computer, the Internet, and Management*, *10*, 1-11.
28. Rainer, A., Hall, T. & Baddoo, N. (2003, September). Persuading Developers to 'Buy into' Software Process Improvement: Local Opinion and Empirical Evidence, *Proc. International Symposium on Empirical Software Engineering*.
29. Sackman, H., Erikson, W. J. & Grant, E. E. (1968, Jan). Exploratory Experimental Studies Comparing Online and Offline Programming Performance, *CACM*, *11*, 3-11.
30. Seaman, C. B. & Basili, V. R. (1997, May). An Empirical Study of Communication in Code Inspections, *Proc. 19th International Conference on Software Engineering*, 96-106.
31. Singer, J. & Lethbridge, T. (1998). Studying Work Practices to Assist Tool Design in Software Engineering, *Proc. 6th Intl Workshop on Program Comprehension*, 173-179.
32. van Solingen, R., Berghout, E. & van Latum, F. (1998, September/October). Interrupts: Just a Minute Never Is, *IEEE Software*, *15*, 97-103.
33. Tang, J. C. (1991). Findings from Observational Studies of Collaborative Work, *International Journal of Man-Machine Studies*, *34*, 143-160.
34. Thomas, S. A., Hurley, S. F. & Barnes, D. J. (1996). Looking for the Human Factors in Software Quality Management, *Proc. International Conference on Software Engineering: Education and Practice*, 474-480.
35. Tichy, W. F. (1998, May). Should Computer Scientists Experiment More? *IEEE Computer*, *31*, 32-40.
36. Wu, J., Graman, T. C. N. & Smith, P. W. (2003, September). A Study of Collaboration in Software Design, *Proc. International Symposium on Empirical Software Engineering*.
37. Zvegintzov, N. (1998). Frequently Begged Questions and How To Answer Them, *IEEE Software*, *15*, 93-96.