Human and Social Factors of Software Engineering – Workshop Summary

Michael John
Fraunhofer Institut for Computer Architecture and Software Technology (FIRST),
Kekuléstr. 7, 12489 Berlin, Germany
+49-30-6392-1782
mjohn@first.fraunhofer.de

Frank Maurer
Department of Computer Science,
University of Calgary,
2500 University Dr NW
T2N 1N4 Calgary, Alberta, Canada
+1-403-220-3531
maurer@.cpsc.ucalgary.ca

Bjørnar Tessem
Department of Information Science and Media Studies,
University of Bergen,
Pbox 7800, NO-5020 Bergen, Norway
+47-555-84-103
Bjornar.Tessem@uib.no

Categories and Subject Descriptors
D.2.3, D.2.9, K.4.3

General Terms
Human factors, software engineering, management

Keywords
knowledge dissemination and reuse, interaction theory, communication architectures, social software, decision support, human-centric collaboration support, agile methods, qualitative studies, group dynamics

1. Introduction

Software is developed for people and by people. Human and social factors have a very strong impact on the success of software development endeavours and the resulting system. Surprisingly, much of software engineering research in the last decade is technical, quantitative and deemphasizes the people aspect. Developers are replaceable resources that are utilized in repeatable processes. Processes need to be improved. Tools need to be used. The "right" notations are important. The workshop on Human and Social Factors in Software Engineering is picking up on the some of the soft aspects in software development that was highlighted in the early days of software engineering by Tom DeMarco and Timothy Lister in their work on Peopleware [1]. It also follows a recent trend in the software industry, namely the introduction of agile methods, and provides a scientific perspective on these. The workshop focuses on the human communication and the social environment of software developers. Including and combining approaches of software engineering with social science, the workshop looked at software engineering from a number of perspectives, including those of agile methods and communication theory, in order to point out solutions and conditions for human-centred software engineering. The workshop aimed to provide a forum in which the latest developments in the field of human-centred software engineering could be discussed. The research presented covered a broad spectrum. A reoccuring theme dealt with the benefits of qualitative research approaches for software engineering research. Workshop participants emphasized the deep insights generated by qualitative research while acknowledging benefits and limitations of quantitative approaches. In the following, we will summarize the results and discussions of the sessions during the workshop. The first session dealt with social factors in software process improvement and software development. The topics discussed were acceptability of software processes, client-developer communication and agile practices like Extreme programming. Session 2 focused on the impact of personality types on software development. The third session examined empirical and qualitative research approaches in the field. For these approaches different methodologies like Discourse and Social network analysis, Human decision support and collaborative method design were discussed. The last session reviewed communication and collaboration aspects of software development. Here communication architectures for Computer Supported Cooperative Work (CSCW) and aspects of usability design in the field of Human computer interface were discussed.

ICSE'05, May 15–21, 2005, St. Louis, Missouri, USA.
ACM 1-58113-963-2/05/0005.

2. Social factors in Software process improvement and Software development

Paul S. Grisham, Dewayne E. Perry in their paper "Customer Relationships and Extreme Programming" treat the social aspects of communication between programmers and clients and the potential impact on the customer relationships in Extreme Programming. In their ongoing research work, they want to elaborate under which conditions and concerns the agile method Extreme Programming might more satisfy the customer or might not. The question they raise is whether the on-site customer always leads to better relationships between the programming teams and the client. The customer's involvement might also bear a risk because of different perspectives on process transparency and different perceptions of expectations about applying pure Extreme programming methodology. The authors therefore want

to examine how customers perceive Extreme Programming as a method and what the customer's business value in Extreme Programming really is about. The used method will be surveys done on customer satisfaction in the service industry to the examine aspects in software development. The literature survey will be enhanced by interviews with Extreme Programming customers in order to develop measures of customer-developer expectations and measures for customer-developer communications.

Amy Law and Raylene Charron demonstrate in their paper "Effects of Agile Practices on Social Factors" two successful industrial projects that apply agile practices to address social factors like knowledge sharing, motivation, and customer collaboration. The comparative experience report regards the impact of agile practices through the issues of co-location, pair programming, sprint review and small releases. The study was done in two different teams with different starting points and under different working conditions. Project X was a new programming project using intensively parts of agile methods (e.g. pairing), Project Y was an enhancement of an already existing system adopting agile methods to their special needs of disruptive software development. Also team size and timeline for delivery in both projects differed. The paper discusses the impacts of different working environmental factors (e.g. "Open Environment", "Bullpen") and considers the different strategies for the introduction of new team members, minimal documentation and the customer process involvement. The lessons learned and recommendations are based on retrospective reviews and observations.

Medha Umarji and Carolyn Seaman include in their current research on "Predicting Acceptance of Software Process Improvement acceptance" the social psychology literature. Their paper focus on challenging personal factors like fear of adverse consequences or the degree of control over personnel working processes as well as organizational factors like visibility or transparency of processes. Applying the Technology Acceptance model (TAM) and the Theory of Planned Behaviour (TPB) to the organizational and personal context of software development the authors elaborate an enhanced model to predict acceptance of software metrics. The metrics acceptance model includes such factors as perceived usefulness, amount of learning, compatibility to work practices and Ease-of-Use. It serves for prediction of acceptance and identification of possible problem areas before implementation of SPI metrics. The metrics acceptance model is currently being validated. For future input it will be used as an instrument to collect data for evaluation of the influencing social factors.

Carol A. Wellington, Thomas Briggs and C. Dudley Girard describe in their paper "Examining Team Cohesion as an Effect of Software Engineering Methodology" an approach to measure the team cohesion in two student teams developing prototypical products. In order to improve their hypothesis that a higher team cohesion increases communication, knowledge sharing and leads to a better product quality they compare the Team Software Process (TSP) within an plan-driven Traditional Life Cycle (TCL) methodology and Extreme Programming (XP) as an agile methodology. The model for team cohesion is derived from the Group Environment Questionnaire Test Manual (GEQ) that consists of four measures: the group's attachment to the task (GI-T), the group's social connection (GI-S), individual attachment to the task (ATG-T) and the individual connection to the group (ATG-S). Additionally a teammate ranking scale was introduced in order to evaluate the GEQ approach. The results of the experiment show that the quality of the code delivered by the XP team was significantly greater than that of the TLC team. The GEQ was not as sensitive as the teammate ranking statistics.

The paper "A Communication Architecture from Rapid Prototyping" of Grace Tai reflects the communication architecture for Siemens Rapid Prototyping (S-RaP) used in a recent user interface design project with over 30 collaborators. The proposed communication architecture defines the structure of key communication roles and communication channels that dictate information flow in order to enhance understandability and efficiency during the project related communication process. The basic model defines the roles as client proxy, lead software engineer and lead user interface designer and the interfaces. For the facilitated communication between the different roles S-RaP introduces a storyboard as a communication artifact for capturing and presenting information in a tangible, cross-domain manner. By annotating the storyboard, the different domain experts as well as other stakeholders could provide feedback throughout the project and help evolve the understanding of marketing requirements and business domain.

3. Personality types and behavioural patterns in Software Engineering

S. Michelle Young, Helen M. Edwards, Sharon McDonald and J. Barrie Thompson describe in their paper "Personality Characteristics in an XP Team: A Repertory Grid Study" an explanatory study in nine systems development teams all based in the UK. The study was an integrated data assessment using qualitative and quantitative methods in order to work out stereotypical and specific personality characteristics appropriate to particular software development roles. As part of a larger study, an XP development team was investigated using a repertory grid approach. From each stakeholder an individual repertory grid was derived that shows the relationship between different roles (constructs) and personality characteristics (elements) on a bipolar scale assigned to the roles. The ratings in cells show strength of alignment of a construct with an element. The individual sets of grids collected from interviewees were evaluated using methods like cluster analysis, correlation analysis and multi-dimensional scaling. The repertory grids analysis identified significant personality characteristics (e.g. sharer/doer, organiser/leader) for different role types (e.g. XP team member, technical architect, operation manager). In addition the concepts for describing good and bad team members were determined.

Georgine Beranek, Wolfgang Zuser and Thomas Grechenig present in their paper "Functional Group Roles in Software Engineering Teams" an empirical examination of the informal role distribution in student software engineering teams in order to approve the concept of functional group roles as defined by Benne&Sheats. [2] The survey took place at the beginning of the group work and contained demographical data, self-assessment of the required technical skills and preferences for tasks allocation. The data concerning typical work styles and behavior in group work was collected based on Belbin. [3] In a second survey at the end of the software development project the participants should

estimate their contribution to each task (in percent). The data was analyzed using correlation analysis and Chi-Square tests in order to elaborate the four major aspects technical skills, conscientiousness, coordination skills, (un)cooperativeness as social factors of software engineering. The clusters were systematically associated with gender, age, group work experience, task skills, task preferences and contributions to specific tasks. As a result of the conducted survey the authors approve that the typical informal group roles of the software engineering teams basically match to the functional group roles. A fourth group of "typical programmers" has been identified.

The paper "Cultural Patterns in Software Process Mishaps: Incidents in Global Projects" of Eve MacGregor, Yvonne Hsieh and Phillipe Kruchten describes the impact of cultural factors and their role in global software development efforts. Therefore it summarizes past anthropological and sociological culture research of Kluckhohn&Strodtbeck [4], Hofstede [5], Edward Hall [6] and Trompenaars and Hampden-Turner [7]. In a first phase of grounded theory application to this field, the goal is to work out a criteria model of cultural mishaps and successes in projects. The space of global outsourcing and sub-contracting is explored interviewing project managers. A deeper understanding of diverse cultural patterns was gained by the comparative analysis of project collaboration, scheduling and system documentation in global software engineering projects. As preliminary results of this phase, cultural patterns have been identified, such as for example the Yes (but no) Pattern, Proxy Pattern, The-customer-is-king (Anti)Pattern. In Phase Two the research will be enhanced by a more quantitative approach providing a questionnaire and conducting semi-structured interviews in order to create a larger data set of incidents which will ensure external validity of the presumed cultural patterns.

Kevin C. Desouza and Yukika Awazu provide in their paper "Managing Radical Software Engineers: Between Order and Chaos" an approach for managing so called "radical engineers". Based on the insight of current work, the authors describe the different working environments in companies as order-based and chaotic. In this working environments software development is situated between inventions and innovations where the radical engineers tend to affect the different management styles with their typical behaviour. For validation of the conceptualization of "radical engineers" semi-structured interviews with 30 engineers were conducted. Based on the research the paper provides some baselines to identify and to manage radical engineers in order to release their potential for software developing organizations.

4. Empirical and qualitative approaches for social factors in Software development

Helen Sharp and Hugh Robinson present in their paper "Some Social Factors of Software Engineering: the maverick, community and technical practices" some findings from three empirical studies carried out during the last 10 years in order to illustrate the spectrum of social factors that arise from, and have consequences for, software development. They adopt a holistic empirical approach, based around studies of software development in practice with interviews, ethnographically-based field studies (with participatory observation), and discourse analysis of contemporaneous literature. No a priori assumptions are made as to which 'people factors' are important, and which are specific to software development. The different studies focus on the influence of hidden stakeholders, so called "Mavericks" in software quality management systems (SQMs), the power of community while promoting the object oriented engineering paradigm around the OOPSLA from the late 1970s to the early 1990s, and the social side of extreme programming practices as, for example, pairing.

Yvonne Dittrich, Kari Rönkkö, Olle Lindberg, Jeanette Erickson and Christina Hansson reflect in their paper "Co-operative Method Development Revisited" the lessons learned in the field of method development and method deployment in software development projects. In their work on three different project settings they integrated ethnographically inspired empirical research in order to understand software development from a participants point of views. Co-operative method development is a domain-specific adaptation of action research in evolutionary cycles of 2 or 3 phases. The advantage of the Co-operative method development is that researchers and involved practitioners together develop a common understanding of problems and possible solutions. The cooperative technical and methodological innovation followed by iterative implementation leads to a higher acceptability by the practitioners. On the basis of the different projects scenarios like agile development, adaptable systems and interaction design they propose the lessons learned by complementing Co-operative method development with prototyping or experiments and draw on the earned understanding how practitioners manage software development projects or why sometimes methods do not work out.

Carmen Zannier and Frank Maurer present in their paper "A Qualitative Empirical Evaluation of Design Decisions" the results of an empirical study on rationalist and naturalistic software design studies [8, 9, 10, 11]. The objective is to provide qualitative results indicative of rational or naturalistic software design decision making. The research process proposed in the paper is twofold. Firstly, a traditional scientific research perspective is introduced in order to work out decision patterns. Secondly the propositions are evaluated by field studies using the Critical Decision Method as an inductive interview technique and observations of group dynamics for design decision in the organisational context of a software development organization that allow the examination of the decision process through an individual perspective [12, 13]. The goal of this qualitative empirical research is to develop and evaluate a model for the influencing factors of design decisions. The hypothesis categories with impact on design decision are presented in the paper.

The paper "The Effect of Human Memory Organization on Code Reviews under Different Single and Pair Programming Scenarios" of A. Günes. Koru, A. Ant Ozok, Anthony F. Norcio describes the design of a case study on human memory organization during the process of pair programming. Based on the hypothesis that programmers can more easily understand and recall programs that are written in chunks, e.g. well-structured code, the authors raises the question under what conditions chunking might have an affect on the efficiency of pair programming. The number of defects discovered at the end of the experiment and the defect discovery time can be used as measures. The authors assume that the expertise level of the participating individuals will have an impact on defect discovery. For validation of this hypothesis, the experiment will be set up for 5 groups with different expertise in

Extreme Programming. Using a broad set of documentation techniques, e.g. paper-pencil, think aloud protocols and video recording, the experiment will empirically describe the potential advantages and possibly disadvantages of chunking from the perspective of code-review performance and satisfaction under different programming scenarios.

Chintan Amrit tackles in the paper "Coordination in Software Development: The problem of Task Allocation" the problem of task assignment in a team as one crucial part of the Coordination problem in Software Development. In a pilot case study based on 4 teams of 30 Masters Student working in a globally distributed environment (Holland and India), the social network structures along with the task distributions in each of the teams were analyzed. Based on social network analysis, Amrit assesses the degrees of network density and network centralization in the advice network and task network. In the pilot survey the central hypotheses that the performance of a team is positively related to the density of the task network, when the density in the advice network is high was approved. Although the pilot survey is limited to a small sample rate the pilot survey shows that it is possible to use social network analysis in order to test hypothesis and propositions related to team performance in a Software Development project. Future research will improve the hypothesis on larger groups in the Software Industry.

5. Tool-based communication and collaboration in software engineering teams

Alistair Sutcliffe presents in his paper "Applying Small Group Theory to Analysis and Design of CSCW Systems" a modelling approach base on the complex adaptive system theory (CAS theory). The theory of small groups as complex systems (SGACS) contains taxonomies of groups for intra-group modelling (so called local dynamics) and whole group modelling (so called global dynamics). [14] The contextual dynamics allow assessing the influences of the group's environment on its composition, coherence and behaviour. The theory is composed of two layers, a bottom-up analysis driven from modelling the composition of groups, and an upper layer of emergent properties that characterise the group as a whole. The lower level, local dynamics, provides an internal view of the group composed of agents, goals, tasks, tools and communication channels. The group-level, global dynamics view describes emergent properties of whole groups such as social cohesion, motivation, shared beliefs, image, goals, satisfaction of members, effectiveness in achieving tasks. At the end the author deduce generic requirements for designing social interaction in groupware systems, e.g. Computer-Supported Cooperative Work systems (CSCW).

Uri Dekel strives in his paper "Supporting distributed software design meetings: What can we learn from collocated meetings?" to build successful tools for supporting distributed software design meetings by addressing the real needs of designers as uncovered by careful observations. Based on a detailed study of two collocated design meetings, the author identifies unique activities of co-located software design which must be translated into the virtual world. The paper discuss issues that must be tackled in the transition to virtual settings, outlines requirements for such tools, and proposes strategies for meeting these requirements. By applying contextual design methodology to face-to-face software design meetings, it becomes possible to describe the cognitive interaction with artifacts, such as whiteboards, and social factors determining interaction between the participants. This investigation uncovers problems in existing collocated meeting procedures, and Dekel also stipulates how virtual meeting tools could alleviated some of these problems.

Ashraf Gaffar, Ahmed Seffah and John A. van der Poll discuss in their paper "HCI Pattern Semantics in XML: a Pragmatic Approach"how user interface design patterns, also called HCI patterns, can be used as an alternative approach to design guidelines for user interface practitioners. Although user interface- and style guidelines have been proposed as one tool for disseminating the design knowledge, the ambiguity, textual representation and lack of formality makes it difficult to understand and apply them effectively in a concrete software artefact, especially when attempting to integrate them with other design (CASE) tools. The 7C's methodology seems to be appropriate in order to create a common format and a shared conceptualization of HCI pattern because it aims to create a mainstream pattern approach that encourages and guides both pattern writers and pattern users to "talk the same language". [15] For a better reuse of HCI patterns the authors propose an approach for pattern representation in XML to effectively support their dissemination and assimilation in a programmable environment. The paper also describes the way to a formal model and a pattern-assisted design environment based on a schema for pattern attributes.

Susan L. Spraragen in her paper "The Challenges in Creating Tools for Improving the Software Development Lifecycle" shows that the basic design principles of user centred design approaches effectively can be applied to build software tools. The success of software tools relies on how well the technical community that builds software tools understands the needs of the technical community that uses these tools [16]. Based on the experiences of past and recent user interface design projects, Spraragen shows the need for a deeper understanding of the designer's, programmer's and user's communities for software tools. Therefore, it is useful to get an insight about the needs, work flows, and styles of a different technical community using a broad variety of different qualitative research techniques like interviews, exercises and acceptance tests. In order to establish continuous user feedback on features and design the concept of "partnering" or participation is introduced. The lessons learned shows that an early engagement of the target community leads to a wider acceptance of the tool.

6. Workshop Results

In addition to the presented papers the workshop resulted in a collection of insights on how to proceed with research within the field of human and social factors as applied to software engineering

6.1 Foundations

To be successful, research within this field has to a great extent draw upon existing knowledge and research approaches from psychology and sociology. Theories ranging from group psychology to management science will constitute helpful foundations to establish knowledge about how software engineering teams can improve their works practices not only considering the technical choices.

Traditionally, most empirical software engineering research is quantitative, focusing on simple, well-defined, measurable factors and their relations. In fact the overall idea of controlled experiments usually takes subjects out of their work context into controlled environments to enable establishing cause-effect relationships. However, to get a full understanding of a social system like a software project, one needs qualitative research in order to get a holistic understanding of what the important factors are and how and why they may influence. Longitudinal effects are hard to examine in controlled environments and short term effects are not extremely relevant for long-term industrial projects. Within the social sciences this has been understood for many years, and believing that software engineering teams are less complex to study than other organisations would be simplistic. Therefore, we should make ourselves acquainted the full range of qualitative methodology and perspectives found in the social sciences, and thus apply ethnography, qualitative interviews, discourse analysis, action research, and other methods. Longitudinal case studies gathering quantitative data together with qualitative data that allows to make sense of the former might provide deeper insights that hard-core controlled experiments where non-random samples are drawn from student populations while the study implies statements about industrial settings.

A fundamental question that was illuminated during the workshop is what constitutes a human or social factor. Personality, skills, and team cohesion are obvious. But is a technological practice like refactoring a social factor, as Sharp et al. suggested in one of the presentations. If this is accepted, we may again ask whether any technological tool, like a compiler, or any practice, like debugging, should be considered a social factor. At least, how these are used has a human and/or social dimension. At the moment there is no definite answer to this, but this makes is even more important to be reflective about the relations between technological solutions and personal as well as team behaviour.

In the continuation of the two previous paragraphs, we also see a need for ways of measuring (if possible) some of these factors, in order to do quantitative research. A lot of measures already exist for instance for skills and personality types, but we need a continuous debate on the relevance and validity of these measures.

6.2 Topics

One of the most important topics of human and social factors research in software engineering is the relations between personality, skills, and roles in a software team. This was the focus of several presentations, and it is obvious that an improved understanding of these relations will be of great benefit to the practice.

Another important issue is the forms of management and levels of freedom for the software engineers in software companies. A statement like "*Happy programmers write better code*", which was expressed at the workshop, seems to indicate that there is a belief that agile practices, as well as management practices oriented towards guiding instead of directing, would improve quality and productivity in software engineering. We have not very much evidence regarding the correctness of such conceptions in software engineering. However, it is a current trend in the management literature, so empirical studies of management style and participation in decision making would probably bring forth knowledge of interest not only or software engineering, but also other knowledge based industries. Related to this are also studies of software process improvement and their influence and acceptance among software engineers.

Another often discussed topic at the workshop was agile approaches. In our opinion, this indicates that these methods (a) are becoming more widely used in industry and (b) that they shift the focus from technical aspects of software development to human and social aspects. In our opinion, this shift in emphasis will dominate this decade as discussions of software process improvement approaches were front and centre in the last and CASE tools discussions stood out in the 1980ies.

As we see that development teams become globally distributed and culturally dissimilar, we encounter new organisational challenges, like awareness of other team members' activities, cultural misfits, different language, etc. For example, MacGregor et al. presented an approach where they use the concept of cultural patterns and anti-patterns to get an understanding of these issues. This is one way to study this challenging topic, and there is a lot of potential here in examining the phenomenon using ethnographic methods and knowledge about cultural differences.

Many of the presentations at the workshop integrated human-computer interaction (HCI) aspects with software engineering, both with the focus on making HCI knowledge available to the software engineer, but also on how HCI issues should be handled in the development process. It must be a research goal to understand how we may integrate the interaction design angle found in HCI and the software design process. In particular, we see a potential for embedding HCI design practices into agile methods for an even higher focus on user needs in the development process.

6.3 Community

It should be a goal to attract more commercial interest in this type of research: getting the human and social aspects right in software development will increase productivity, improve quality and customer as well as user satisfaction. All of these can be translated in dollars. Knowledge about human and social factors will most likely be highly valued as improved organisation of work practices is essential in order to be competitive. Mature organisations know about this, so there is a high potential for getting valuable empirical knowledge if we are able to define projects that match business needs.

The workshop participants agreed that the workshop had been a positive experience. It is a goal to organize a second workshop on this research topic in 2006.

7. Literature

[1] Tom Demarco, Timothy Lister; Peopleware. Productive Projects and Teams, 2cnd edition, New York, 1999.

[2] Benne, K. D., and Sheats, P.; Functional Roles of Group Members. The Journal of Social Issues, vol. 3, no. 2, 1948, pp. 41-49.

[3] Belbin, M.; Team Roles at Work. Butterworth-Heinemann, 1993.

[4] F. Kluckhohn and F. Strodtbeck; Variations in Value Orientations. Evanston, IL: Row Peterson, 1961.

[5] G. Hofstede; Culture and organizations - Software of the mind: McGraw-Hill, 1997.

[6] E. T. Hall; Beyond culture. New York: Anchor Books/Doubleday, 1976.

[7] J. S. Olson and G. M. Olson; "Culture Surprises in Remote Software Development Teams," ACM Queue, vol. 1, pp. 52-59, 2004.

[8] Klein G; Sources of Power, MIT Press Camb. MA; 1998.

[9] Patton M.Q; Qualitative Research & Evaluation Methods 3rd Ed.; Sage Publications, California; 2002.

[10] Simon H; "A Behavioural Model of Rational Choice"; Quarterly J. of Econ.; V69 Issue 1 99-118; 1955.

[11] Simon H; "The Structure of Ill Structured Problems"; Artificial Intelligence 4, 181-201; 1973.

[12] Klein G; Calderwood R, MacGregor D; "Critical Decision Method for Eliciting Knowledge"; IEEE Trans. on Sys.,Man and Cybernetics; V.19, No.3, May/June; 1989.

[13] Klein M; "Capturing Design Rationale in Concurrent Engineering Teams", IEEE Comp. V26 No.1 93-47,Jan1993.

[14] Arrow, H., McGrath, J. E., and Berdahl, J. L.; Small as complex systems: Formation, coordination, development and adaptation. Thousand Oaks CA: Sage, 2000.

[15] Gaffar, A., Seffah A., Javahery, H., Sinnig, D.; MOUDIL: A Platform for Capturing and Sharing Patterns. Patterns in Practice Workshop, ACM CHI Conference, Florida, April 2003.

[16] Winograd, Terry; From Programming Environments to Environments for Designing, Communications of the ACM, 38,6, (June 1995), 65-74.