Computer Supported Cooperative Work (2009) 18:393–399 © Springer 2009
DOI 10.1007/s10606-009-9106-y

Software Engineering as Cooperative Work
Editorial

Yvonne Dittrich[1], Dave W. Randall[2] & Janice Singer[3]
[1]*Software Development Group, IT University of Copenhagen, Rued Langgaardsvej 7, DK-2300 Copenhagen S, Denmark (E-mail: ydi@itu.dk);* [2]*Department of Sociology Manchester Metropolitan University, Geoffrey Manton Building off Oxford Road, Manchester M15 6BH, UK (E-mail: D.Randall@mmu.ac.uk);* [3]*Industrial Research Assistance Program, National Research Council Canada, Communications Research Centre, 3701 Carling Avenue, Shirley's Bay, Ottawa, ON K2H 8S2, Canada (E-mail: janice.singer@nrc-cnrc.gc.ca)*

Software engineering research (SE) and research on computer supported cooperative work (CSCW) seem to have had a difficult-even non-existent-relationship at various times. The reasons might vary, but are likely to include the fact that, while depending on each other to achieve the common goal of developing useful software, the methods and the theoretic underpinnings they depend upon differ considerably. CSCW, as is well-known, is firmly based on a worldview that stresses the role of human actors in the production of technical artefacts, and in particular how best to understand the cooperative engagements that inevitably underpin human activity. This has meant, in turn, that analysis has largely come from the province of the human sciences. Unsurprisingly, this has methodological consequences. Although by no means ubiquitous, so-called 'qualitative' methods have become accepted to such a degree that they are entirely unremarkable in the CSCW community. Ethnographers now apply their trade in a large variety of contexts, and groups involved in the production of software artefacts have become one of the contexts in which they are increasingly commonly found.

Software engineering has, in contrast, historically emphasised the nature of the *process* entailed in production. We should not forget that there are good reasons for this. Software development teams became progressively larger and their coordination more complex. Equally, business demands meant that these engineering processes became increasingly subject to various forms of accountability. Hence the need for strict, controlled processes of documentation became paramount and resulted in the demand for 'structure'. In addition, of course, a worldview that was steeped in engineering and science recognised the primacy of methods which were 'objective' and which produced acceptable generalisations. Anyone trained in such a background would—quite understandably—have had some difficulty coming to terms with the context-specific, seemingly anecdotal, and frequently hedged conclusions one arguably gets from a more qualitative orientation.

Nevertheless, from about the early 1990s, something began to change, The sheer number of scare stories about the failure of software projects, many involving billions of dollars, led to a recognition that a rigorous engineering-based approach might be creating problems of its own. Put simply, it was argued that engineering work tended to be done using a problem-solution matrix that underestimated the complexity of problems, the dynamic quality of human understanding of those problems, and the variety of perspectives coming to play in organisational reality (see e.g. Checkland 1981). These resulting problems meant that 'waterfall' assumptions became increasingly problematic and the resultant strain began to show. The search for any and all resources with which to deal with this included a more benevolent view of the qualitative. Ethnographers and other experts in qualitative approaches became 'licensed' to provide additional data which could be used to support software engineering work, to such a degree that the recent issue of one of the leading course books on Software Engineering contains a chapter on the requirements engineering process recommending work place studies, ethnographically inspired investigations and interviews (Sommerville 2007). CSCW studies, in short, became in some sense studies *for* software engineers (this is not the place to enter debates about how 'in some sense' might be deconstructed—see Dourish 2006; Crabtree et al. 2009).

CSCW research also began to address the design and development of software as a cooperative work practice itself. Knowing about the complex relation between plans and situated action, changing the methods for requirements engineering and giving design recommendations for computer applications and especially their interfaces might not be enough. To get to terms with the design side of Computer Supported Cooperative Work, the cooperative practices of designing and developing software needs to be understood. The 'working relationships of technology production and use' (Suchman 1994) have become a major subject of research and publication. A special issue of the CSCW journal in 1996 (Schmidt and Sharrock 1996) as well as a number of edited volumes like Dittrich et al. (2002) and Voss et al. (2008) indicated this growing interest. More or less at the same time, the understanding of the need for flexibility and support for continuously ongoing articulation and design in use led to investigation of programming technologies that allow for an End-User Development (Lieberman et al. 2006). Software design and development in this way becomes regarded as embedded in everyday work practice.

In other words, it turned out to be a relatively short step to studies *of as well as for* software development. From a small number of studies conducted in the 1990s (see e.g. Button and Sharrock 1994) which were of interest to the CSCW community but arguably less so elsewhere, we now have a concerted effort to mobilise qualitative approaches to understand software engineering better and make some contribution towards an assessment of its problems and putative solutions, an effort which trades on the methodological insights proffered by CSCW and cognates but is being made by communities of software engineers as well.

Of course, reflective practitioners of whatever kind are wont to examine their practices in order to, if possible, improve it. The traditional understanding of software development methods and processes has been more and more problematised by software engineers and designers themselves. They share with CSCW practitioners a common interest in how software engineering actually takes place, and how software developers construct and use tools, methods, and processes to cooperatively develop software that fulfils its purpose. While some part of this entails a shift to the methodological commitments rehearsed above, it is also a function of some radical developments in the philosophy of software development. This has included, from fairly early on, concerns with participatory design, evolutionary development etc. (Bjerknes et al. 1987; Floyd et al. 1992), though to more the contemporary developments associated with open source development, Xtreme programming, agile processes and so on. As, there is a considerable diversity to be found today in how software is actually developed and how software development is conceptualised, there is an agreement that one size does *not* fit all and that software development methods and processes need to be devised according to the contingencies and circumstances in which software development takes place. Given this development, it is certainly not surprising that they should also be implicated in the qualitative turn. The growing interest in detailed studies of how development structures, methods and tools are used to co-construct the process and the product of software development dovetails nicely with the increased complexity and variegation of those tools and practices.

Regardless, and as indicated, the software engineering community has begun to engage in a concerted way. At the International Conference on Software Engineering in 2000 a workshop explored the usage of social science methods for software engineering research (Sim et al. 2001). Together with the work by Carolyn Seaman (Seaman), this workshop influenced a growing community of software engineering researchers complementing existing empirical research with studies applying qualitative methods. Since then, a number of workshops served as meeting places at the leading International Conference on Software Engineering. (John et al. 2005; Cheng et al. 2008; de Souza et al. 2009) interestingly the first workshop in the Cooperative and Human Aspects of Software Engineering (CHASE) series took place 2006 at the CSCW conference in Banff (Cheng et al. 2006). A special issue on Qualitative Software Engineering Research in one of the leading Software Engineering journals (Dittrich et al. 2007) indicates another milestone to the successful establishment of this research discourse as part of software engineering. This Software Engineering research discourse contains a wide spectrum of qualitative research. Qualitative methods can be used as means to formulate hypotheses that later can be tested by quantitative research. (Seaman 1999) e.g. uses qualitative methods this way. Others, like (De Souza et al. 2005) use an in depth understanding of software developers' work practice to design tools supporting this practice. A recent article presented at the International Conference on Software Engineering established the importance of rich data to provide a sound base for understanding Software Engineering practices (Aranda and Venolia 2009). At the same time, the growing geographically distribution of software engineering provided new challenges for the software development processes, method and tool support, resulting in a growing body of research. Similarly, agile software engineering and open source development have been the subject of studies aimed at understanding these relatively new approaches to software development. As the respective communities increasingly emphasise communication and cooperation rather than (quantitative) control, qualitative methods often have been chosen as *default* research methods for understanding this shift. This special edition of the journal is designed to showcase a number of the studies which reflect these trends.

In short, and regardless of which community is actually engaged in particular research endeavours, the analytic preferences associated with the development of CSCW (and which are to be found now in many communities outside of CSCW itself, including for instance IS; HCI, and so on) have become appreciably more accepted elsewhere, not least among software engineers and designers themselves. We are arguably at the point where a collaboration predicated on experiences over a long period of time is at last bearing fruit:

As mentioned above, there is a growing interest in detailed studies into software development as co-construction. The first article in this special issue, 'What Counts as Software Process? Negotiating the Boundary of Software Work through Artifacts and Conversation' by Cohn et al. explores how an agile development process is co-constructed within the development team and how the boundaries of what is inside and what is outside of the process are negotiated in everyday conversation. In a similar detailed analysis, the second article 'On The Roles of APIs in the Coordination of Collaborative Software Development' by De Souza et al. addresses how software design methods and guidelines provide mediation mechanisms for cooperation between sub-teams working with different parts of the same projects.

The challenges of globally distributed research are at the center of another cluster of studies that, in recent years, has generated enough critical mass its own conference on 'Global Software Development'. Research studies addressing distributed development are often done with the aim of supporting cooperation distributed across time and space. The article by Avram et al. shows how the use of tools enables specific work practices and—on the other hand—how in order to 'keep the local work flowing', the tools and practices that support distance cooperation are patched locally. Computer mediated awareness tools become increasingly relevant. One of our articles argues for the innovative design of a tool that can be extended to not only support developers with awareness information related to code changes, but also related artefacts. Omorania et al. propose 'Using Developer Activity Data to Enhance Awareness during Collaborative Software Development'.

The last three articles in this volume point to a growing interest in cooperation beyond the project team and individual project. Rooksby et al. address in their article ' 'Testing in the Wild' The Social and Organisational Dimensions of Testing Practices'. In the field material, the influence both of project specific constituencies as well as cooperation with customers and users. The interaction and cooperation between software developers and other communities is in the centre of Segal's article 'Software development cultures and cooperation problems: a field study of the early stages of development of software for a scientific community' as well. Here especially the difference in practices and values of End-User Developers on the one side and professional software engineers is addressed. Last but not least, Johannessen et al. address the cross-organisational cooperation between heterogeneous user communities, third parties and developers around the evolution of a software product. 'Integration and generification—Agile software development in the healthcare market'. Especially these later articles provide a new challenge for the CSCW community in general: They can be read as accounts of how the cooperation around design, development and evolution of computer support becomes part of cooperative work practices in general, both within an organisation and between organisations, which provides a change in the 'techno-methodologies' (Button and Dourish 1998) of software and computer applications.

Thanks to all the reviewers who assured the quality of this Special Issue.
Jørgen Bansler, Technical University of Denmark, Denmark
Wolf G. Bleek, blueCarat AG, Germany
Tone Bratteteig, University of Oslo, Norway
Peter Carstensen, Alexandra Institute, Denmark
Jeffrey Carver, University of Alabama, USA
Daniela Damian, University of Victoria, Canada
Pirjo Elovaara, Blekinge Institute of Technology, Sweden
Sara Eriksén, Blekinge Institute of Technology, Sweden
Erling Havn, Technical University of Denmark, Denmark
Sampsa Hyysalo, University of Helsinki, Finnland
Michael John, Fraunhofer Institute Berlin, Germany
Anne Mari Kanstrup, University of Aalborg, Denmark
Catherine Letondal, Pasteur Institute, France
Wayne G. Lutters, University of Maryland, USA
Gail Murphy, University of British Columbia, Canada
Brad Myers, Carnegie Mellom University, USA
Peter A. Nielsen, University of Aalborg, Denmark
Jacob Nørbjerg, Copenhagen Business School, Denmark
Volkmar Pipek, University of Siegen, Germany
Neil Pollock, University of Edinburgh, UK
Tony Robertson, University of Technology Sidney, Australia
Hugh Robinson, The Open University, UK
Kari Rönkkö, Blekinge Insitute of Technology, Sweden

Anita Sarma, University of California Irvine, USA
Carolyn Seaman, University of Maryland, USA
Helen Sharp, The Open University, UK
Margaret-Anne Storey, University of Victoria, Canada
Bjørnar Tessem, University of Bergen, Norway
Mike Twidale, University of Illinois, USA
Gina Venolia, Microsoft Research, USA
Thomas Zimmermann, University of Calgary, Canada.

References

Aranda, J. & Venolia, G. (2009). The secret life of bugs: Going past the errors and omissions in software repositories. Proceedings of the 31st International Conference on Software Engineering. Vancouver 2009: IEEE pp. 298–308.

Bjerknes, G., Ehn, P., Kyng, M. (Eds.). (1987). *Computers and democracy.* Aldershot 1987.

Button, G. & Sharrock, W. W. (1994). Occasioned practices in the work of software engineers. In Goguen & M. Jirotka (Eds.), *Requirements engineering: Social and technical issues*. San Diego: Academic.

Button, G., & Dourish, P. (1998). Technomethodology: Paradoxes and possibilities, Proceedings of the SIGCHI conference on Human factors in computing systems: common ground, pp. 19–26. Vancouver.

Checkland, P. (1981). *Systems thinking, systems practice*. London: Wiley.

Cheng, L.-T., Cox, A., DeLine, R., de Souza, C., Schneider, K., Singer, J., et al. (2006). *Supporting the social side of large-scale software development*. Banff: Workshop at the CSCW 06.

Cheng, L.-T., de Souza, C., Dittrich, Y., John, M., Hazzan, O. et al. (Eds.) (2008). Proceedings of the 2008 international workshop on Cooperative and Human Aspects of Software Engineering CHASE '08. 30th International Conference on Software Engineering, Leipzig 10–18 May 2008, ACM.

Crabtree, A., Rodden, T., Tolmie, P., & Button, G. (2009). Ethnography considered harmful. Proceedings of CHI '09. Boston: ACM.

de Souza, C., Froehlich, J., & Dourish, P. (2005). Seeking the source: software source code as a social and technical artifact. Proceedings of the 2005 international ACM SIGGROUP conference on Supporting group work, Sanibel Island, Florida, pp. 197–206.

de Souza, C., Sharp, H., Dittrich, Y., & Singer, J. (2009). Proceedings of the 2009 LCSE Workshop on *Cooperative and human aspects of software engineering (CHASE 2009)* IEEE.

Dittrich, Y., Floyd, C., & Klischewski, R. (Eds.). (2002). *Social thinking: Software practice*. MIT Press.

Dittrich, Y., John, M., Singer, J., & Tessem, B. (2007). Editorial for the special issue on qualitative software engineering research. *Information and Software Technology, 49*(6), 531–539.

Dourish, P. (2006). Implications for design, Proceedings of CHI '06. Montreal: ACM.

Floyd, C., Züllighoven, H., Budde, R., & Keil-Slawik, R. (eds). (1992). *Software development and reality construction*. Berlin: Springer Verlag.

John, M., Maurer, F., & Tessem, B. (2005). Human and social factors of software engineering: workshop summary. SIGSOFT Softw. Eng. Notes 30(4)1–6.

Lieberman, H., Paternó, F., & Wulf, V. (Eds.). (2006). *End user development: Empowering people to flexibly employ advanced information and communication technology*. Springer 2006.

Schmidt, K., & Sharrock, W. (Eds.). (1996). Special issue on studies of cooperative design. Computer Supported Cooperative Work, vol. 5, no.4, 1996.

Seaman, C. (1999). Qualitative methods in empirical studies of software engineering. *IEEE Transactions on Software Engineering, 25*(4), 557–572.

Sim, S. E., Singer, J., & Storey, M.-A. (2001). Beg, borrow or steal. Using multidisciplinary methods in empirical software engineering research. An ICSE 2000 Workshop Report Limerick, Ireland, 5 June 2000. *Empirical Software Engineering, 6*(1), 85–93.

Sommerville, I. (2007). *Software engineering*. 8th edition, Pearson Education.

Suchman, L. (1994). Working relations of technology production and use. *Computer Supported Cooperative Work, 2*(1–2), 21–39.

Voss, A., Hartswood, M., Procter, R., Rouncefield, M., Slack, R. S., & Büscher, M. (2008). *Configuring user-designer relations: Interdisciplinary perspectives*. Springer.