# Approaches to Software Engineering: A human-centred perspective

Liam J. Bannon

Interaction Design Centre
Dept. of Computer Science & Information Systems
University of Limerick
Limerick, Ireland
Liam.bannon@ul.ie

**Abstract.** The field of software engineering has been evolving since its inception in 1968. Arguments as to the exact nature of the field, whether it should be conceived as a real engineering profession, the role of formal methods, whether it is as much an art as a science, etc., continue to divide both practitioners and academics. My purpose here is not to debate these particular topics, but rather to approach the field from the outside, coming as I do from a long period of involvement in the human and social side of the computing discipline, namely, from the fields of Human-Computer Interaction, Computer Supported Cooperative Work, Participative Design, Interaction Design, and Social Informatics, more generally. I wish to examine how this "human-centred" perspective might shed a new light on some issues within the SE field, perhaps opening up topics for further discussion and examination.

**Keywords:** CSCW, human-centred computing, requirements, sociology, software engineering

## Extended Abstract of the Keynote

It is difficult to talk about issues in the Software Engineering (SE) field without first noting the larger landscape of computing and information systems in which it is embedded. Computing traditionally has focused on answering the question : What can be automated? (e.g. Arden, 1980). While the term computer was originally used to describe real people performing numerical calculations, the human side of computing has tended to be ignored within the emerging discipline of computer science, which has focused on hardware and software issues. Emphasizing this, one of the first professional organizations for people involved in computing was titled The Association for Computing Machinery (ACM). As focus has shifted from mainframe computing to personal and now ubiquitous computing, there has been a slowly increasing awareness of the need to pay greater attention to the human aspects of computing. This implies much more than simply noting the social implications of computing technology, but rather seeks to view the activities of people involved in various aspects of computing, especially systems development and programming, as a legitimate, and necessary part of a computing curriculum.

Many people have been involved in the attempt to shift the focus of computing - and informatics more generally – away from a purely technical approach concerned with hardware and software only, to one that considers the human activities of design and use of information systems as being of central concern. Interestingly, many of these people have come from the Nordic countries. My own selection of pioneers in this space would include people such as Kristen Nygaard, who argued for a perspective on systems development that included the social and political, as well as the technical. People like Peter Naur, whose compilation of papers was published by ACM under the title Computing: A Human Activity, which emphasized the human side of programming and systems development. People like Christiane Floyd, from Germany, who presciently wrote of different paradigms in software engineering and the need to allow for multiple perspectives in the field. In the US, perhaps one of the earliest popular publications that promoted a human-centred approach to software was the 1971 book by Gerry Weinberg, a practitioner and consultant, entitled The Psychology of Computer Programming. Rob Kling spent many years as an advocate of a more open computer science discipline that he labelled "Social Informatics". In recent years, a number of senior figures in the field have also put their hats in the ring: Peter Denning, former President of ACM, arguing for a new and more expansive computing profession; Denis Tsichritzis, head of GMD, the former German national research centre for IT, critiquing much old-fashioned computer science as being akin to "electric motor" science; Peter Wegner, in theoretical computer science, arguing that the concept of interaction in computing is fundamentally more powerful than algorithms; and Terry Winograd, one of a number of people involved in bringing the larger field of Design into computing, and developing the Interaction Design field. All of these authors, despite significant differences in their messages, to my mind share a critique of how the field of computing and the academic discipline of computer science has been defined, circumscribed, and taught to students, and all advocate a more "human-centred" approach, in one form or another. For example, in reflecting on our educational system, Denning (1992) notes: "A curriculum capable of preparing students for the shifting world must incorporate new elements emphasing design, demonstrated proficiency, effective interaction with others, and a greater sensitivity toward the historical and cultural spaces in which we all live and work". The issue here is not simply providing computer science students with a rounded education, but more fundamentally questions the very nature of the discipline, arguing that human activities and interests are part of the core of the computing discipline, whenever we conceptualize, design, build, and test new technologies. It is this tradition that I wish to discuss in the context of human-centred software engineering.

These alternative views of the computing field have, I believe, contributed to the slow emergence of what is beginning to be termed, in some quarters, "human-centred" computing (HCC). The label may appear somewhat meaningless, as who would subscribe to an alternative "system-centred" computing label? However, just as the label "user-centred design" in the field of human-computer interaction hit a chord in the 1980's, it may be the case that the "human-centred computing" label will have similar re-orienting effect on the field of computing today. Likewise with other new terms that are appearing currently. For example, the emergence of new terms and research areas, such as the “new informatics” to augment traditional information systems research, and “interaction design” augmenting traditional HCI, are, in my opinion, examples of shifts in perspective towards a more wholistic view of human-systems interaction that begins to pay more attention to the inextricable inter-weaving of the human, social and cultural with the technical aspects of computing. Note that these are not simply surface changes, nor should they be viewed simply as ancillary issues in relation to the dominant computational approach, but rather they raise foundational issues for the field of computing per se. While this is not the place to further develop this argument, I wish now to briefly examine how this human-centred perspective, loosely described above, might be of interest within the software engineering field. The primary area I will focus on my keynote is in the requirements engineering phase of software development.

Early textbooks on software engineering provided scant coverage of any “human” issues, with perhaps a brief mention concerning meetings with user representatives in the derivation of requirements, and in designing the user interface. However, we can observe an increasing concern with “user issues” in standard SE textbooks over the years. The increasing prominence of Participative Design approaches to system development, involving close cooperation with users in all phases of an iterative design process, and the prominent role of prototyping and testing, was starting to be felt in the HCI arena in the late 80’s. Also, the rise of the CSCW field was occurring at this time. The CSCW area brought in researchers from other human sciences than psychology, such as sociology and anthropology, to better understand the everyday lives of people, with a view to providing insights that might be useful in the design of more habitable systems. In the case of the classic Sommerville (2010) text on SE, this can clearly be linked to the rise of the CSCW field and the establishment of a CSCW Centre at Lancaster where sociologists and software engineers were involved in joint projects. However, as I will detail in the keynote, this marriage of social and computing science has not been without some difficulties, especially in the context of “producing requirements”. There is an issue as to whether the developing relations between such unlikely bedfellows as technical systems developers and social scientists, particularly ethnographers, and more narrowly ethnomethodological ethnographers, should be seen as a virtuous coupling or a “deadly embrace”. While it should be obvious that I am in favor of any and all approaches to requirements that open-up this phase to a richer appreciation of the work context and work practices of people, I also feel that this recent courtship between developers and sociologists may turn sour due to a misalignment of motives and interests. If we are to have a useful interplay between these two professions then perhaps we also need to be aware of their different agendas, so as to reduce confusions and misunderstandings. I will explore this issue in greater detail in the keynote.

Returning to this issue of “requirements” in SE, one finds a number of perspectives on them, as evidenced by the different language used. So, for some people, systems design begins with the need for “requirements capture” - which to me inspires an image of requirements as well-defined entities just waiting to be plucked from the environment. It goes without saying that this particular viewpoint is less widely held today than heretofore. A less extreme view, yet one which is still quite popular in the engineering community is the notion of requirements “gathering”, which again has an implicit, if not explicit, conception of requirements as things that are waiting to be harvested. Continuing on this line, one can hear discussion of requirements “elicitation” which begins to acknowledge that requirements may not be immediately apparent, or accessible, and may require some effort to “bring forth” from the user community. Going one step further, we can argue that requirements are not “out there” awaiting collection, but are themselves constructions, jointly and severally produced by a range of actors, including users and developers in specific contexts of discussion, observation and analysis. This view thus requires that we pay close attention to the ways in which we investigate the use situation and work context, and take into account the social, political and economic factors involved in the requirements process. (In this regard, the edited collection by Jirotka and Goguen (1994) provides an interesting range of positions on social and technical issues in requirements engineering.)

A number of commentators have noted how requirements as fixed “texts” can impede a good design process. The designer Chris Jones (1988) argues: “...[we must] recognize that the ‘right’ requirements are in principle unknowable by users, customers, or designers at the start.” This position calls into question the nature of most formal software development contracts today. Similarly, the consultant Tom Gilb (1990) stresses the need to focus on process, not method or static product. He notes that current development methodologies “...are based on a static product model. They do not adequately consider our work to be a continuous process—derived from the past and being maintained into the future.” Yet another voice in support of this shift, coming from academic software engineering, is that of Floyd (1987). She argues for more emphasis on the process of software development than on the efficiency of the resulting code: “The product-oriented perspective regards software as a product standing on its own, consisting of a set of programs and related defining texts... considers the usage context of the product to be fixed and well understood, thus allowing software requirements to be determined in advance,” while the process-oriented perspective “views software in connection with human learning, work and communication, taking place in an evolving world with changing needs... the actual product is perceived as emerging from the totality of interleaved processes of analysis, design, implementation, evaluation and feedback, carried out by different groups of people involved in system development in various roles.” It is interesting that some of the recent moves to Agile Methods in software development and the rise of the Extreme Programming movement would seem to provide support to aspects of the above viewpoints, and thus show, in some respects, a focus on a more “human-centred” approach.

## References

1. Arden, Bruce W. (Ed.) (1980) What can be Automated? The Computer Science & Engineering Research Study (COSERS). MIT Press series in Computer Science: 3, Cambridge, Mass.: MIT Press.
2. Denning, Peter (1992) Educating a new engineer. Communications of the ACM, vol. 35, no.12 , December, pp.83-97.
3. Floyd, C. (1987). Outline of a paradigm change in software engineering. In G. Bjerknes, P. Ehn, & M. Kyng (Eds.), Computers and democracy – A Scandinavian challenge (pp. 191-212). Aldershot, UK: Avebury.
4. Gilb, T. (1990). Project Management for the 1990s. The American Programmer, 16-30.
5. Jirotka, M. & J. Goguen (Eds.) (1994) Requirements Engineering: Social & Technical Issues. London: Academic Press.
6. Jones, J.C. (1988). Softecnica. In J. Thackara (Ed.), Design after modernism: Beyond the object (pp. 216-226). London: Thames & Hudson.
7. Naur, P. (1992) Programming: A Human Activity. New York: ACM Press.
8. Sommerville, I. (2010) Software Engineering (9th Edition). Reading, Mass: Addison-Wesley.
9. Wegner, P. (1997) Why interaction is more powerful than algorithms. Communications of the ACM, 40 (5), 80-91.
10. Weinberg, G. (1971) The Psychology of Computer Programming. New York: Van Nostrand Reinhold.