IST-Africa 2006 Conference Proceedings
Paul Cunningham and Miriam Cunningham (Eds)
IIMC International Information Management Corporation, 2006
ISBN: 1-905824-01-7

Socially Aware Software Engineering for the Developing World

Edwin BLAKE[1], William TUCKER[2]
[1]*Dept Computer Science, University of Cape Town, Rondebosch, 7701, South Africa*
Tel: +27 21 650 3661, Fax: + 27 21 689 9465, Email: edwin@cs.uct.ac.za
[2] *Dept Computer Science, University of the Western Cape, Bellville, 7535, South Africa*
Tel: +27 21 959 2516, Fax: +27 21 959 1274, Email: btucker@uwc.ac.za

Abstract: While the social effects of Information Technology (IT) have received much attention there is very little work on targeted methodologies to develop IT applications and content in a developing world environment. This paper describes a methodology called *Socially Aware Software Engineering* we are busy formulating based on firsthand experience building Information and Communication Technology solutions. Our method is based on a classical user-centred approach from Human Computer Interaction combined with aspects of Participatory Design and cyclical software engineering practises. These approaches are wrapped into an iterative Action Research paradigm in order to directly include the community-based users of our systems. We outline three cases studies based on our evolving method. The paper concludes with suggestions on changing the nature of tertiary curricula in developing countries in a way that integrates this socially aware software engineering methodology.

Keywords: Action Research, Computer Science, Developing world, Human Computer Interaction, Participatory Design, Software Engineering, Free and Open Source Software.

1. Introduction

How can we develop software for rural and disadvantaged communities in the developing world? Making useful and useable Information and Communication Technology (ICT) applications for characteristic developing world situations turns out to be quite difficult. The problems that arise are different from the typical software engineering issues that confront software analysts and developers working in the paradigmatic situations covered by most computer science training, even in developing world universities.

This research set out to develop and deploy useful systems for the kinds of users one meets in the developing world. Our method is to build systems and then reflect on the design method that will enable us to have a useful and sustainable impact.

We explore software engineering and methodological development issues in this paper with reference to three case studies from South Africa. The paper presents our alternative approach to software engineering for these situations, and provides some indicators of success and failure. The paper concludes with implications for training ICT developers in tertiary educational institutions situated within the developing world. While we speak of software in general the issues apply to Free and Open Source Software as well.

2. How to solve local problems in the developing world?

Ironically, the students nurtured by Computer Science departments in developing countries are well positioned to create ICT solutions that meet the needs of users in the *developed* world. Training projects often focus on applications that are far removed from the needs of local communities. Computer Science innovation, when applied with a too narrow and technical focus, produces applications suitable for developed countries instead of solving local problems.

Students come from diverse backgrounds and they have some understanding of the needs in local communities and how technology can help. Students who emerge from Computer Science programmes are able to produce software for export but unfortunately they are often simply trained to emigrate and support the economies of developed countries. These students actually possess the social and technical abilities to address problems that arise in *local* user communities, and our aim is to leverage their skills toward local solutions.

We are busy deriving an application development methodology suited to community-oriented development in a developing country environment, mainly targeted to underprivileged urban and remote rural areas. Aside from dealing with well-understood and easily addressed technical challenges, we also focus on ways to steer Computer Science students and researchers towards understanding community needs and social mechanisms, and how these social implications affect technological choices and development.

We believe that there are clear differences between the requirements of ICT solutions targeted towards problems in developing countries *versus* those for developed countries. However, this research is not designed to undercover those domain differences in a systematic way: we doubt that such an exercise is really possible - it would imply that one could analyze every kind of problem and categorize them. For example, we would certainly not suggest the kind of technology that developing world applications should use: it could be old established technology, but frequently the most advanced technology is the easiest to use. Instead we strive to develop a method whereby such differences can be uncovered for a particular chosen problem domain by developers. Our emphasis is on training for design in the developing world.

We believe that a Computer Science point of view is a fruitful one to bring to the discussion on the use of ICT for development, because Computer Scientists have a profound realization that ICT is completely adaptable and malleable. People who are trained in developing new software have less of a tendency to take existing systems as immutable and given. Rather, systems are seen as adaptable, and are often constructed in components that lend themselves to usage in innovative ways. We do not have to make do with fixed applications and content that is aimed at specific user communities. Instead, we can easily adapt software technology to cater for social needs. Although we all know this about ICT, we have not yet learnt how to exploit this flexibility for the typical user groups found in the developing world.

3. Software Development Methodology

The potential impact of Computer Science comes from the task that is facing developing countries: ICT helps to make scarce knowledge resources available in a widespread fashion. For this to happen we must have applications and content that address local needs. We maintain that such development of new applications, as well as developing tools for creating content and new software engineering methods is the task of Computer Scientists.

We believe that Computer Scientists in the developing world (or those interested in tackling developing world ICT challenges) have to learn to target their applications to the needs of local users. This requires an appropriately situated methodology for systems

development that can elicit user requirements from user communities and produce solutions that work effectively. Computer Scientists will have to overcome a mindset that is fascinated by technological issues and somewhat disdains social involvement. As Software Engineers we have to find partners from social disciplines to assist us in this and we have to train our students to work accordingly.

The needs that arise in underdeveloped communities are not served by merely providing access to equipment, or to applications developed for first world users, or simply access to the Internet. A serious process of co-development has to be followed that involves a community as a whole together with socially sensitive ICT specialists. It is the responsibility of the ICT specialists to initiate and guide this cooperative process. It is the responsibility of Computer Science departments to research and develop this process, investigate it with pilot sites and then train students according to the outcomes.

In order to avoid the technical bias associated with traditional software engineering approaches we are reaching towards a synthesis of several approaches. A bottom-up research approach was chosen to understand and address real community needs. It takes into account the issues related to developing and using software in the community aside from only the technical ones. This process lead to a *Socially Aware Software Engineering* approach based on a combination of the following:

- *User-centred* methods taken from the field of *Human Computer Interaction* (HCI)
- *Participatory Design* methods to ensure that solutions meet user requirements
- *Action Research* cycles to guide the process of working with actual communities

Essentially, the socially aware software engineering framework adopted for this project represents a customized version of the Action Research process described by Susman and Evered [16], with pertinent HCI and participatory design principles included in an iterative development process. This is an alternative to the standard waterfall model of software development [15].

Human Computer Interaction: The field of HCI has a long history of user involvement, or user-centred design. Many classical HCI heuristics and techniques can be tailored for use in developing world situations. We have employed techniques such as paper prototyping and mapping work processes but have come to see that many HCI heuristics fail to capture the social complexities involved when designing ICT solutions for the developing world [14]. In a developing world context, Dray *et al.* [9] recommend working the answers to questions like “How to improve the fit between technology, specific human needs, and human contexts; how to design technology to facilitate human interaction with it; and how best to manage the process of technology introduction” into the software design process.

Participatory Design: Participatory Design evolved from attempts to empower workers in industrial settings [7]. Contemporary Participatory Design specifies a set of techniques to increase user involvement in the software development life cycle. This is done to increase the chance that software solutions are appropriate for the people they serve. Participatory Design provides guidelines on the techniques to ensure that the software prototypes developed addressed user needs. These include discussion groups and paper prototyping amongst others. However, as we found when trying to deploy these methods in a developing world context, the users are not ICT savvy enough to fully participate in the process. This is addressed in our conclusions with *intermediaries.*

Action Research: Action Research is a methodology geared towards solving a problem for a target group of people by involving them as equal partners in the process and using their expertise in their area of work [1]. It aims to empower groups by creating relevant solutions for their problems and benefits both participants and researchers in the process. Action Research provides the steps needed for engineering a locally relevant application. It describes the overall process for approaching a target community and guidelines on how to

work with that community in order to discover a problem area and provide a solution for this problem. The key aspect, however, is the personal participation of the researchers in the community [13].

Socially Aware Software Engineering: The process we used (in the case studies presented below) was essentially user-centred and participatory since it involved the target community in the entire software development process. The development process was cyclical and iterative, involving traditional Action Research stages to analyse the circumstances of the target community, identify a problem to work on, plan an intervention, implement an intervention and evaluate the outcomes [2]. Finally, after each Action Research cycle, there was a period of reflection on the results of that cycle in order to plan the next cycle correctly. At each stage of the process, requirements were continually gathered for the design of prototype applications. The entire process was documented using notes, semi-structured interviews, questionnaires and audio records as suggested by [1][13][16]. Additionally, the different versions of the software prototypes developed acted as documents of the changes that were undertaken throughout the Action Research cycles.

This contrasts with the (much critiqued) waterfall model of software development that is commonly used for software engineering and has five pre-determined stages consisting of requirements definition, system and software design, implementation and unit testing, integration and system testing and operation and maintenance [15].

The socially aware software engineering methodology does not include specific markers or indicators to measure success. Rather these need to be developed on a per-project basis. Three such projects are described below. In addition, a number of different evaluation methodologies are compatible with socially aware computing, including Outcome Mapping [10] and the Real Access/Real Impact criteria prescribed by bridges.org [4].

4. Case Studies

4.1. Cyber Tracker

This project enabled 'illiterate' animal trackers to use a PDA and GPS system to record field observations in wildlife parks. These trackers are literate in reading the tracks and signs left by animals as they moved through the park. By re-designing the interface of a PDA (Personal Digital Assistant) - a device designed for business executives - we created a system which the trackers could understand and use to record their observations. This made their expertise available to the park management and scientific communities and also improved their position: a key empowerment/indigenous knowledge system.

This project was developed over several critical action research cycles. The iterative process involved the target users at every stage. The end solution was one developed jointly by the researchers and the trackers who were involved as users [3]. The target users felt comfortable using the system because they had a significant stake in designing the system and the system was appropriate to their needs.

4.2. Rural Tele-Health

We developed a tele-health system for rural communities in South Africa. The system made uses of VoIP technology to enable nurses at a rural clinic to consult with doctors in order to provide consistent and improved health care to local people [6]. An intriguing feature of rural areas in South Africa is that people are not concentrated in small towns and villages but are rather spread out as scattered settlements throughout the countryside. Rural hospitals tend to support 10-12 satellite clinics scattered in a roughly 20km radius. Each clinic then supports a somewhat scattered population of up to 20,000. We built a long-range WiFi network and designed a mixed synchronous (real-time) and asynchronous (store-and-forward) communication system to support remote tele-consultation. The asynchronous

feature requirements emerged from the end-users as the researchers learned that frequent power outages, and more importantly, over-burdened doctors and nurses meant that end-users could rarely talk in real-time when they wanted to.

Now in the third year of active field trials, consisting of a long series of Action Research cycles, we have come to focus on the human computer interface [14]. The long-term iterative process allowed the target users to participate and guide the development of the system, even though they largely remain limited in ICT skills. We have incorporated a continuing ICT training element to the process in order to move toward the goal of employing more and more Participatory Design techniques as user skills improve.

4.3. Deaf Telephony

The Deaf Community of Cape Town (DCCT) is a doubly disadvantaged community due to both poverty and hearing disorders. Using our development methodology we have built a telecommunication bridge between Deaf and hearing users using PCs, the Public Switched Telephone Network (PSTN) and various open source Internet technologies [11]. A Deaf user types and reads text with an Instant Messaging client on a PC. The system automatically converts text to voice and relays it to the PSTN via VoIP. The hearing user replies as usual but the speech is intercepted and translated to text with the help of a human relay operator (instead of automated speech recognition, of which open source tools are still not up to task). Usage of the tool can be problematic. From repeated weekly contact with Deaf users, we have come to learn that Deaf users use a different grammar (specifically related to South African Sign Language) and this causes problems for text communication in English. Deaf users would prefer to use sign language, which means we must include video into the tool (at the expense of losing the limited automation already provided since sign language recognition and translation remains one of the most difficult natural language challenges). Deaf users also prefer to use cell phones as input/output devices instead of physically travelling to the Deaf community centre to access the system on a PC.

The Deaf users suffer from poor literacy due to poverty and poor education and are also computer illiterate. As with the rural tele-health project, an ICT training programme was put into place. In addition, DCCT also provides daily access to an essentially free Internet café. With repeated exposure to training and open access to technology, a small core group of users has emerged to be able to participate in the software requirement process. We have incorporated many of their suggestions into the software to improve the human computer interface, such as an audio “isTyping” for hearing users.

5. Challenges in Transforming the Computer Science Curriculum

Computer Science departments in South Africa tend to be rather small, are generally understaffed and often suffer from high staff turnover rates. At an institutional level the obstacles to be overcome include the lack of recognition given to work that spills over traditional discipline boundaries (in spite of years of claims that interdisciplinary research is a "good thing") and to largely qualitative research with its concomitant lack of hard research results. Students that come to Computer Science are frequently interested in technology and not initially interested in learning how to communicate with members of disadvantaged communities. These difficulties in communication have many aspects, including large cultural and language differences.

There are four major influences on Computer Science curricula in South Africa:

1. *Tradition*: this tradition has dictated a four-year Bachelors (Honours) degree followed by a two-year purely research based Masters Degree. The Honours degree typically has a large self-study development project in the final year. The course of study is based on contributions from several largely autonomous departments in the Science Faculty.

2. *Industry*: emphasizes immediately useful practical skills for companies that have very often not appreciated the implications of globalization. These are companies whose idea of innovation is to be the first adopter in the country of an established practice from the US or Europe.
3. *Curricula*: the work of the US bodies (ACM & IEEE) on Computer Science curricula has been influential in determining content. The amount of material placed in the "core" of discipline is large and leaves little room for local-orientated content.
4. *Process*: the impact of the British Computer Society (BCS) with its emphasis on the process of teaching (rather than content) has been felt recently. The influence is generally very positive except for the reduced emphasis on large group projects in the final year in favour of individually assessable work.

None of these influences is particularly favourable to the kinds of innovations that are needed to provide the kind of training for the approach outlined in the previous section and developed in the rest of this paper.

6. Conclusions

In this paper we have argued for a new method of Software Engineering that is required in the developing world. It is one that is equally aware of social issues as it is of technical issues. We have outlined the process by which we arrived at our conclusion via a number of practical software development projects that we have undertaken. We have also looked at issues that will have to be addressed to change the Computer Science and Software Engineering curriculum in the case of South Africa. We address these issues below.

Finally we believe that advancing *Free and Open Source Software* (FOSS) for the developing world will require the kind of Software Engineering processes and skills that we advocate in this paper. The strength of FOSS is in part predicated on access to the source but that of course implies the need for local Software Engineering skills that can use and modify the source appropriately. As Gabriella Coleman points out [8] there is significant lock-in to proprietary software in the developing world due to a lack of skills in exploiting FOSS. The report on FOSS by bridges.org [5] points out "*specific software applications (whether FOSS or proprietary) that could make computers more useful to local communities — such as putting ICT to work to improve healthcare and education, and designed with cultural factors in mind — are still missing*". If the developing world is to take ownership of FOSS we will have to address such issues of technical skills and useful and useable software.

6.1. Academic Policy for Community-Based Computer Science

The following conclusions and recommendations are intended to shape academic policy to make computer science research more relevant to the needs of local communities. They are specifically targeted at Computer Science Departments at Universities in developing countries.

- Software Engineering as a profession has to change to emphasize the social and economic needs of local communities. A Software Engineering method that extends beyond purely technical aspects, as demonstrated in this paper, is needed. An Ethics focussed on dealing with development priorities has to replace the emphasis on first world professional issues and values.
- IT professionals have to accept a new interdisciplinary approach to Software Engineering that involves co-development of applications in a socially sensitive fashion. In practice these projects are difficult to manage and further work is needed on this. The members of the team have to understand and value the different roles and perspectives.

- Universities (and NGO's) have great opportunities to design and implement new approaches to using technology to support local communities in developing countries. The responsibility incurred to avoid "forever pilots" [12] and to ensure that solutions and projects that work are continued beyond the initial engagement.
- Innovative niche products arise from such IT developments and social entrepreneurship can turn the outputs of community-based development into products and businesses.

6.2. Software Engineering Methodology for Developing Countries

We have outlined our method above for a *Socially Aware Software Engineering* methodology. The basis of Critical Action Research gives a strong emphasis on the empowerment of groups: facilitating change in a community through facilitating action. The precepts of Participatory Design require the end user to participate in the software design process. The flaw from our point of view is that it assumes at least a degree of sophistication of the user community in relation to technological possibilities and an ability of software designers to bridge large cultural and linguistic gaps. This may not be possible. The cultural gaps can be enormous. The technological requirements exist within a complex web of other needs, relationships and societal obligations. Misinterpretation (on both sides) and unexpected needs are common. It is difficult for IT practitioners to appreciate, for example, how an IT empowerment exercise may threaten power relations in such communities with dangerous consequences for several participants.

Our tentative solution to this is to have local "interpreters" or champions who can bridge the gaps. These act as our *intermediaries* into the communities. These are often school teachers or students at the University who come from the target community groups. Through them, we are better able to carry out Action Research cycles incorporating the most appropriate aspects of participatory design and user-centred HCI into the software engineering process.

6.3. Future work

The socially aware software engineering methodology needs to be refined and documented. This means more work on integrating the Action Research, Participatory Design and HCI aspects, and then relating those issues specifically to the software design and development process. Furthermore, the study has to be expanded to include Computer Science in other developing world nations: our interest is particularly in those in Sub-Saharan Africa.

A further development would be producing an integrated method that encompasses design for developed world and the developing world. Currently our thinking is that we are proposing an extension of existing software engineering methods. In that sense the methodology proposed here is a superset of what is required for projects in the developed world. The task in the developing world is harder! However one would have to be able to recognize just where on the spectrum of user involvement it is necessary to be for a particular project: this is not an issue that we have considered yet.

Acknowledgements

Our thanks to the dedicated staff of the various organisations that we worked with: South Africans (CSIR, Eskom) and NGO's (bridges.org, Transcape, DCCT). Our appreciation extends especially to the people in all the communities who were our co-workers on this project and our research students. Various aspects of this work were funded by the IDRC (Canada), SANPAD (Netherlands), NRF (South African National Research Foundation) via its THRIP and Focus Area programmes, Cisco South Africa, Siemens South Africa and Telkom.

References

[1] E.R. Babbie and J. Mouton (2001). *The Practice of Social Research*. Oxford University Press.

[2] R.L. Baskerville and A.T. Wood-Harper (1996). A Critical Perspective on Action Research as a Method for Information Systems Research. *Journal of Information Technology*, 11:235-246.

[3] E.H. Blake (2001). A field computer for animal trackers. *Proc. 2nd South African Conference on Human-Computer Interaction*. (CHI-SA 2001), published in ACM CHI '02 Extended Abstracts on Human Factors in Computing Systems, ACM Press, 532-533.

[4] Bridges.org (2004). The Real Access/Real Impact Framework for Improving The Way ICT Is Used In Development. Available at: www.bridges.org/real_access/RealAccess_overview_bridges_03Aug04ii.pdf.

[5] Bridges.org (2005). Comparison Study of Free/Open Source and Proprietary Software in an African Context. Available at http://www.bridges.org/software_comparison/report.html

[6] M. Chetty, W. Tucker and E. Blake (2004). Developing Locally Relevant Applications for Rural Areas: A South African Example. *Proc. SAICSIT 2004*, Cape Town, South Africa, ACM Press, 234-239.

[7] A. Clement and P. Van den Besselaar (1993). A Retrospective Look at Participatory Design Projects. *Communications of the ACM*, 36(6):29-37.

[8] Gabriella Coleman (2004) The Politics of Open Source Adoption, NGO's in the Developing World. Published by Tactical Technology Collective, found at http://www.tacticaltech.org/SSRC_Report.

[9] S. Dray, D.A. Siegel and P. Kotze (2003). Indra's Net: HCI in the Developing World. *Interactions*, ACM Press, 11(2):28-37.

[10] S. Earl, F. Carden and T. Smutylo (2001). *Outcome Mapping: Building Learning and Reflection into Development Programs*. Ottowa, Canada: International Development Research Centre.

[11] M. Glaser and W.D. Tucker (2004) Telecommunications bridging between Deaf and hearing users in South Africa. *Proc. Conference and Workshop on Assistive Technologies for Vision and Hearing Impairment*, (CVHI 2004), Granada, Spain, (CD-ROM publication).

[12] N. Gunawardene (2005) Waiting for Pilots to Land in Tunis. Islam Online. www.islamonline.net/English/Science/2005/11/article10.shtml

[13] S. Kemmis and R. McTaggart (2000). Participatory Action Research. Handbook of Qualitative Research, N.K. Denzin and Y.S. Lincoln (Eds), 567-606.

[14] A. Maunder, G. Marsden and W. D. Tucker (2006). Evaluating the relevance of the 'Real Access' criteria as a framework for rural HCI research. *Proc. Computer Human Interaction in South Africa, CHI-SA 2006*, Cape Town, South Africa, ACM Press.

[15] I. Sommerville (1998). *Software Engineering*. Addison Wesley, Fifth edition.

[16] G.I. Susman and R.D. Evered (1978). An assessment of the Scientific Merits of Action Research. *Administrative Science Quarterly*, 23:582-603.