Second International Workshop on Interdisciplinary Software Engineering Research (WISER’06)

Nikolay Mehandjiev  
School of Informatics  
The University of Manchester  
PO Box 88, Sackville Street,  
Manchester, M60 1QD, U.K.  
Tel: +44 161 200 3319  
mehandjiev@acm.org

Pearl Brereton  
School of Computing & Mathematics  
Keele University  
Keele, Staffordshire ST5 5BG,  
U.K.  
Tel: +44 1782 583079  
o.p.brereton@cs.keele.ac.uk

John Hosking  
Dept. of Computer Science  
The University of Auckland  
Auckland  
New Zealand  
Tel: +64 9 3737 599 Ext. 88297  
john@cs.auckland.ac.nz

ABSTRACT
WISER is a series of international workshops that focus on identifying and transferring techniques from other disciplines that might usefully be applied to software engineering research and practice.

The workshops address this topic through presentations and discussions of both actual case studies and of ways in which potentially useful approaches can be identified, adapted and adopted within software engineering.

The papers in the proceedings address topics ranging from a general approach to identifying domains that have similar experimental practices to software engineering to specific case studies of the application of techniques from, for example, graph theory, strategic planning, economics and social and cognitive theory.

Categories and Subject Descriptors
D.3.3 [Software Engineering]: General.

General Terms
Management, Design, Economics, Human Factors, Theory

Keywords
Interdisciplinary Software Engineering, Technology Transfer

1. INTRODUCTION
Software is now accepted as a key contributor to wealth creation and improving the quality of life, and this has led to increased demands from the software engineering to simultaneously increase productivity, flexibility, robustness and quality. The resultant new development paradigms, formalisms and methods of working have demonstrated remarkable success, but there is still room for improvement, especially in software value chains.

The idea for these workshops was borne out of the realisation that these problems and demands are not unique to software engineering, and that a number of cognate disciplines may have developed answers to similar problems which are useful for us. Transferring these answers will not only help us solve problems, but also lead us to new software engineering research areas by identifying assumptions which might be challenged in an interdisciplinary context.

The benefits seem clear, yet software engineers have been remarkably reluctant to look outside of own discipline for inspiration and answers both in terms of research and practice. The widely cited CACM paper by Glass, Ramesh and Vessey [1] pointed out that only 1.9% of the Software Engineering papers are using theories and models from other disciplines. Computer Science papers used other disciplines in 10.77% of the cases, whilst Information Systems papers used other disciplines in 67.9% of the cases!

The WISER series of workshops aims to redress this balance and formulate a research agenda focused on the future of software engineering as an interdisciplinary activity. The 1st workshop in the series, WISER 2004 [2], formulated a strategic programme of work to facilitate interdisciplinary transfer, and this second workshop will commence the implementation of the following strategic vision formulated by WISER 2004:

The Workshops on Interdisciplinary Software Engineering Research (WISER) aim to facilitate transfer of ideas and concepts from other disciplines to address contemporary issues of software engineering research and practice, and to eventually turn software engineering into more holistic and innovative discipline.

2. WORKSHOP FOCUS
The first workshop formulated a set of research priorities to reflect the series’ vision. These priorities suggested the following initial focus for the follow-up workshops:

- the study of the process of transfer, and the development of methods, models and techniques to aid transfer activities (see, for example, the paper by Walenstein at WISER 2004 which provides a general model for coordinating transfers of theories from cognate disciplines into software engineering [3]);
• identifying areas where transfers will be most fruitful;
• strategies for facilitating the diffusion of transfer ideas;
• examples of transfers where further research is necessary before they can be applied to software engineering.

3. EXAMPLES OF INTERDISCIPLINARY TRANSFER
The first workshop provided a diverse set of examples of interdisciplinary transfer into software engineering. These ranged from the application of aikido principles in a collaborative design situation [4] to the use of ecological formalism in understanding a software system’s fitness for purpose [5]. Other potential examples, both from the first workshop and elsewhere, include, but are not limited to:

• Cognitive design of representations and formalisms to meet growing productivity demands, e.g. [6]
• Application of service delivery concepts and methods of organising to replicate the flexibility of service-based organisations in software applications, e.g. [7];
• Software product flexibility via the application of industrial product engineering principles such as product families, multi-functional components, mass customisation and explicit variability, e.g. [8];
• Holistic approaches to organising the software engineering process as a human centred activity, for example the application of systemic concepts to encourage teamwork and knowledge sharing, or ethnography-informed approaches to studying practices of software design, e.g. [9];
• Socio-technical approaches to software engineering, for example co-design and co-optimisation of human, economic and software sub-systems, e.g. [10];
• Application of complexity theory and chaos theory models for managing the information explosion associated with contemporary software engineering projects, e.g. [11];
• Application of economic and financial techniques such as transaction cost models, trading crowd approximation and option pricing models to ensure effective software procurement, e.g. [12].

Collectively these examples were used to frame the request for participation in the second workshop.

4. WORKSHOP PAPERS
We have attempted to maintain a similar breadth of diversity in the set of papers selected for this second workshop. Discipline representations in these papers range from medicine, through to social and cognitive theory, business organization and planning, and formal graph theory. The majority of these papers focus on identification of areas where transfer is or has been fruitful and should act as a useful set of case studies for the workshop to explore the other areas of proposed focus described in Section 2. We are encouraged that the set of disciplines represented include several outside the exemplars introduced in Section 3.

In his keynote speech, Jeffery focuses on the use of theories external to software engineering to explain empirical findings regarding software engineering activities. He uses three case studies to illustrate how inter-disciplinary theory can bring about deeper understanding of the empirical data, and also help us in formulating a programme of research.

In their paper “Investigating the Applicability of the Evidence-Based Paradigm to Software Engineering”, Budgen et al advocate the development of a systematic analysis instrument which can help us judge if software engineering is a suitable domain for implementing evidence-based approaches to research and practice. In particular, it looks at how we might identify domains that have experimental practices that are similar to those of software engineering.

Ye, in his paper, discusses the importance of social collaboration in the Software Engineering development process seeing this process as one of “incremental crystalization of knowledge” into software systems. Ye argues that more support for knowledge collaboration between developers is needed, and introduces two prototype tools that aim to provide this: CodeBroker, which automatically provides context relevant documentation, and STeP_IN which supports display of such documentation. These follow the principle of layered information-on-demand to “focus on the economical utilization of human attention by presenting information in different levels of details and on demand”.

The idea of isomorphism between the structures and procedures underpinning the application domain and the application software itself is explored in the paper by Kollingbaum et al. It proposes the use of structures and techniques from the target domain of virtual organizations to underpin the construction and operation of software systems which can support the rapid construction and flexible coordination of such virtual organizations.

The paper by Chatzigeorgiou et al., focuses on analyzing the structure of software using Graph Theory, aiming to help design activities such as determining overloaded classes, good system decomposition clusterings and scale-freeness of OO systems, plus detection of design patterns. This brings into focus the usefulness of metrics in general in the software development process.

The idea of using strategic planning techniques to enhance early phases of software engineering is explored in the paper by Masao Kakihara. After a brief review of strategic management frameworks, the paper proposes the use of Eisenhardt’s framework of ‘Strategy as Simple Rules’ and explores strategic decision making in internet service development, using a case study from Japan.

The theme of decision making in software engineering is also found in the last of the six full papers, “A Multi-Disciplinary
In it Sassenburg uses economics, decision-making and software management perspectives to analyze decision-making involved in software releases, and proposes a methodology to improve strategic software release decisions using concepts of financial loss outcomes and decision reversal costs. In the first of the short papers, Hazzan and Dubinsky discuss the use of constructivism as a cognitive approach to analyzing software development methods, using extreme programming as an example method.

The focus on small-scale development activities within a software engineering lifecycle is retained in the second short paper by Christian Del Rosso. In it the author explores the boundaries of what we consider to be contributions to software engineering. Genetic algorithms are used to automate a small-scale manual activity (determining the configuration of a data structure) in the software engineering lifecycle.

A more general reflection on the impact which non-conventional programming technologies such as neural networks and even non-conventional interface devices such as an artificial nose can have on the software engineering lifecycle including development and testing activities is offered in the paper by Petrounias and Kodogiannis.

The last paper in the proceedings by Jansen and Brinkkemper covers an important area which, despite being a strong business concern, is not usually covered by the software engineering literature. It presents a model of for customer configuration updating focused on customer interaction, and demonstrates its use for analysis of capabilities of software vendors in relation to product release and update management.

5. SUMMARY

We hope that the contributions from this workshop, together with the organizing framework developed at WISER 2004 [2], will advance discussion of the role of inter-disciplinary diffusion of ideas and theories for transforming conventional software engineering into a modern discipline. This ability to "look outward" and learn from other disciplines will, we argue, be an important component in the maturing of software engineering. We also assert this to be an important precursor for software engineering practitioners to be able to tackle the forthcoming challenges associated with the growing role of software in our society.

6. ACKNOWLEDGMENTS

We gratefully acknowledge the help of the members of the Programme Committee and the rest of our Steering Committee who did their best to ensure that submitted papers were reviewed under tight deadlines.

7. REFERENCES