Research Wish List – Stephen MacDonell

Most of my own research, and that which I supervise, addresses two concepts that are increasingly important in software engineering: uncertainty and evidence. With respect to uncertainty I am interested in how we recognise it, capture it, represent it, and reduce it. In regard to evidence I want to know how we should collect it, how we should analyse and present it, and how it is used. In relation to software systems these interests are typically expressed in the form of questions such as: What can we deliver in the next release? How long will it take to develop these new features? How can we cost-effectively decrease the defect rate in our code? Who are 'the users'? What decisions do Product Owners need help with?

In terms of research methods, I am relaxed about what people use as long as it enables them to deliver against the objectives of the work in a rational and robust manner. For instance, if a student is building a tool to address a problem then this work would follow a constructive, design science approach. If the project involves the analysis of data to evaluate a new prediction model, then a more experimental approach would be used. If the aim was to assess user satisfaction, then interviews would be more appropriate.

A. Managing uncertainty in software engineering

The development and management of software systems is inherently uncertain. Software is an abstract entity, representing items or concepts. Software that is yet to be developed or is incomplete is normally described in an abstract way, often requiring extensive reliance on models. Thus when considering software requirements we are considering an abstract representation of an abstract entity that does not yet exist!!

Sample research topics:

A1. Practitioner knowledge capture and codification
A2. Manager/developer influence on project planning and execution
A3. Planning/replanning software projects and activities
A4. Building a fuzzy logic/AHP/visualisation toolset for software analytics
A5. Modeling processes and systems using system dynamics and simulation

B. Empirical software engineering

Collecting and analysing data from software products, processes and resources can help individuals, groups and organizations to set and achieve improvement goals – for instance, reducing defect density in code artifacts, or improving cost estimation accuracy by selecting more influential features.

Sample research topics:

B1. Impact of sampling on empirical modeling outcomes
B2. Optimising estimation accuracy using multiple methods
B3. Assessing the accuracy and sensitivity of recorded effort data
B4. Extent of change in metrics data from project inception to project closure

C. Evidence-based software engineering 2.0

Numerous decisions are made when developing and deploying software-intensive service ecosystems (SISE) – which technology stack is best, should we adopt Scrum…? Such decisions should – ideally – be based on sound, relevant and timely evidence. However, the infrastructure to enable evidence-based software engineering, beyond systematic reviews and proprietary boundaries, is lacking.

Sample research topics:

C1. Understanding the evidence needs of practitioners
C2. Architecture design for an EBSE infrastructure
C3. Can science evidence be 'packaged' so that it appeals to practitioners?
C4. Moving EBSE past the SLR – EBSE as BAU…

D. ICT systems success and failure

Public attention naturally falls to the failures in systems, and it is quite right that we should try to learn lessons from those failures. However, many software systems are delivered successfully, in the eyes of at least one group of stakeholders. There are opportunities to learn from these experiences in order to identify processes, events and actions that lead to successful outcomes.

Sample research topics:

D1. Identifying and classifying patterns in successful projects
D2. Best practice vs. common practice – avoiding mediocrity, maintaining creativity, facilitating innovation
D3. Negotiated notions of success among stakeholders
D4. Process/methodology assessment and refinement in use

E. The software/system boundary, and the evolution of autonomous systems

There is a growing gap between what we know about software systems and how we develop and manage them. We know, for instance, that:

- 'simple' systems become packages; those we will build will be increasingly complex, and will be deployed into contexts that are also increasingly complex
- this complexity is already well beyond any individual's ability to understand, meaning that we will increasingly need systems to understand themselves
- the bulk of this complexity comes not from within but from 'the edges' – associated with interoperability, interactions, interfaces
- today's software engineering methods, that are founded on the premise of providing internal control, are increasingly inadequate in this context
- today's approaches for managing complex projects, that are also founded on the premise of providing control, are misdirected and outdated.

The scale and complexity of tomorrow's systems, the growing volumes and diversity of sources of data, the ongoing dissatisfaction with development and management methods, and the limited scope of 'software engineering' to the software rather than the solution or service mean that the future usefulness of this paradigm should be questioned and complementary or alternative paradigms sought.

Sample research topics:

E1. New metaphors suitable for considering the development, deployment and management of future software systems
E2. Observing systems as they evolve over time
E3. Treating software systems as continuous rather than discrete
E4. Viewing systems as autonomous organic beings
E5. Applying and evaluating agile management practices

F. Software forensics

The ability to identify the author of software has, rather sadly, become increasingly important, as fraud and malicious attacks via software become more common. Less dramatic but just as important are disputes over software ownership or software negligence. In all of these circumstances it is useful to identify, characterize or discriminate between those writing the software. This endeavour is known as software forensics, and it represents the coming together of elements of software engineering, security and the law.

Sample research topics:

F1. Assessing the robustness of authorship analysis methods
F2. Investigating authorship patterns in open source software
F3. Building a software forensics toolkit