Editorial for the special section on Empirical Studies in Software Engineering

Selected, extended papers from the Eighteenth International Conference on Evaluation and Assessment in Software Engineering, May 13th-14th 2014, London, UK.

While it could be argued that every study of a computer-based system is empirical to a degree, the past twenty-five years or so has seen the assessment of systems and their associated practices emerge as a mainstream software engineering discipline. The growing depth and breadth of studies in the area demonstrates the value that exploring systems using quantitative and qualitative techniques can give. Empirical studies allow new software engineering techniques to be assessed, old techniques to be re-visited, old and new theories to be tested and past assumptions to be challenged and, potentially, revised.

Four papers comprise the special section of this Journal issue and they cover diverse areas. Two papers in the special section relate to replications of previous studies and provide insights into the state-of-the-art. Such studies are an important way in which a body of software engineering knowledge can be strengthened and new perspectives gained. Moreover, replication can validate existing findings and test those findings by applying a study's methods to other systems, but replication is not without its problems. The unique context of each empirical study and variations between original study and replication often present difficulties.

In their paper: "Investigations about replication of empirical studies in software engineering: A systematic mapping study", Cleyton de Magalhães, Fabio da Silva, Ronnie Santos and Marcos Suassuna provide a comprehensive assessment of replication studies dating back to 1996. Six research questions were posed and then answered relating to trends in the thirty-seven studies extracted from the search process. The research questions included detailed examination of the frequency of studies over that period, the composition of authors in the studies, the topics addressed and how study results had been presented. The paper provides insightful reflection; the number of replication studies is still very small and there is still a lack of consistency in the terminology used and understanding of the replication concept. A range of discussion points and questions are also raised by the study. Most notably, what is the exact definition of a replication? How do we gauge a successful replication? Finally, how should replications be reported? The authors suggest that 'replication work is being conducted without a solid conceptual background about replications, which could result in low quality results'. The study is therefore a reality check for the empirical software engineering community - replication is still under-valued and under-used and a change in attitudes towards replication is needed. While there are considerable practical benefits to replication if done well, more effort needs to be invested in the methodology and definitions underlying replication studies. Finally, the authors report that the number of authors engaged in replication is small; while in itself this is not a problem, a lack of dissemination activities may be hindering progress in the area.

The second replication paper: "On the Usefulness of Ownership Metrics in Open-Source Software Projects" by Matthieu Foucault, Cedric Teyton, David Lo, Xavier Blanc and Jean-Remy Falleri partially replicates the influential 2011 study by Bird et al[1]. The original study examined whether the code ownership of modules influenced faults in those modules. Using proprietary Microsoft code, the original Bird et al. study found that code ownership, measured by the ratio of contributions that developers make to a module, was related to faults. Bird et al. originally reported that modules with a higher number of major contributors had fewer faults. Bird et al. also originally reported that the performance of fault prediction models improved when code ownership metrics were added. Foucault and colleagues partially replicate the original study using seven open-source systems. They report that the impact of code ownership on faults was less clear cut than in the original study; code ownership does not seem to have the same influence on faults in open source code as it does in proprietary code, for the systems studied. This is an important finding and the paper allows us to draw several valuable conclusions. We already know that the context of software engineering is influential. Research findings based only on specific contexts need to be tested outside those contexts as these findings are unlikely to be generalisable. It is crucial that we develop a better understanding about the impact of context factors on software engineering. An understanding of the differences between open-source and proprietary systems is a key feature of this paper. Additional studies such as these are needed if we are to develop a more rigorous understanding of how to develop better software within specific contexts.

As software development becomes a more global activity and teams are often distributed across different global parts, new practical and research challenges in systems development have arisen. An increasingly important aspect of these challenges is in understanding how personality, communication and social aspects influence the productivity of teams; in particular, how knowledge is diffused through members of those teams. The third paper in the special section is devoted to these types of developments traits, entitled: “Communication and Personality Profiles of Global Software Developers” by Sherlock Licorish and Stephen MacDonell. The paper uses a sample of 146 practitioners drawn from ten teams of the IBM Rational Jazz repository as its empirical basis. Network analysis was used to study interactions between developers in the ten projects. Four research questions are posed relating to ‘Top Members’ – a cluster distinguished from ‘Others’ in a team by the relatively high number of messages they communicated and their engagement in task changes. The questions related to firstly, how important Top Members were to their team’s knowledge diffusion; secondly, the personality profiles of the same group; thirdly, whether those personality profiles differed from less active practitioners and, finally, how personality profiles were distributed across global teams. In response to question one: ‘Top Members communicated in nearly seven times as many software tasks as their team-mates, and were significantly more likely to disseminate more knowledge across their teams’ communication network’; for question two, some evidence of specific common personality profiles was found in Top Members. In response to research question three, the personality profiles of Top Members were not found to be significantly different to those of ‘Others’. Finally, software developers tended to share personality characteristics. These four results in themselves provide an interesting window into less well-known and understood features of development. However, the real value of the paper is in the potential for future work in the area. The authors state some insightful implications for project managers who they say: ‘…may use these trends to assemble and manage teams with individuals who are together both socially and mentally equipped for any given software situation’.

The final paper in the section addresses the highly topical and industry-relevant issue of team performance in an agile environment. The paper entitled: “Performance Alignment Work: How Software Developers Experience the Continuous Adaptation of Team Performance in Lean and Agile Environments", aims to understand factors characterising high-performing software development teams working in a fast-paced, volatile external environment with moving targets. The context chosen for the study is teams that adhere to the Lean and Agile software development paradigms. The investigation method employed a qualitative multiple-case study and thematic interviews with sixteen experienced practitioners in five organisations. The investigation identified thirty-three major categories of performance factors and relationships between the factors. The study provides some interesting insights into forming successful software development teams and also useful insights into the context of software process improvement. One interesting observation was that high-performing teams excelled at continuously negotiating with stakeholders at different levels in the organisation and adapting their goals and targets accordingly. This implies that one crucial skill in the team is the ability to communicate and interact with the various stakeholders.

Tracy Hall, Ingunn Myrtveit and Steve Counsell

{tracy.hall, steve.counsell}@brunel.ac.uk, ingunn.myrtveit@bi.no

[1]C. Bird, N. Nagappan, B. Murphy, H. Gall, P. Devanbu, Don't touch my code!: examining the effects of ownership on software Quality, in: Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering, ESEC/FSE '11, ACM, Szeged, Hungary, pages 4-14, 2011.