# Examples of REFSQ Structured Abstracts

Lombriser *et al*. REFSQ 2016 :

**Abstract.** **[Context & Motivation]** Engaging stakeholders in requirements engineering (RE) influences the quality of the requirements and ultimately of the system to-be. Unfortunately, stakeholder engagement is often insufficient, leading to too few, low-quality requirements. **[Question/problem]** We aim to evaluate the effectiveness of gamification to improve stakeholder engagement and ultimately performance in RE. We focus on agile requirements that are expressed as user stories and acceptance tests. **[Principal ideas/results]** We develop the gamified requirements engineering model (GREM) that relates gamification, stakeholder engagement, and RE performance. To evaluate GREM, we build an online gamified platform for requirements elicitation, and we report on a rigorous controlled experiment where two independent teams elicited requirements for the same system with and without gamification. The findings show that the performance of the treatment group is significantly higher, and their requirements are more numerous, have higher quality, and are more creative. **[Contribution]** The GREM model paves the way for further work in gamified RE. Our evaluation provides promising initial empirical insights, and leads us to the hypothesis that competitive game elements are advantageous for RE elicitation, while social game elements are favorable for RE phases where cooperation is demanded.

**Keywords:** Gamification · Requirements elicitation · Empirical study · Agile requirements · Gamified Requirements Engineering Model

Sadi and Yu REFSQ 2016

**Abstract. [Context and motivation]** Open innovation is becoming an important strategy in software development. Following this strategy, software companies are increasingly opening up their platforms to third-party products for extension and completion. **[Question / problem]** Opening up software platforms to third-party applications often involves difficult trade-offs between openness requirements and critical design concerns such as security, performance, privacy, and proprietary ownership. Deliberate assessment of these trade-offs is crucial to the ultimate quality and viability of an open software platform. **[Principal ideas / results]** We propose to treat openness as a distinct class of non-functional requirements, and to model and analyze openness requirements and related trade-offs using a goal-oriented approach. The proposed approach allows to refine and analyze openness requirements in parallel with other competing concerns in designing software platforms. The refined requirements are used as criteria for selecting appropriate design options. We demonstrate our approach using an example of designing an open embedded software platform for the automotive domain reported in the literature. **[Contributions]** The proposed approach allows to balance the fulfillment of interacting requirements in opening up platforms to third-party products, and to determine "good-enough" and "open-enough" platform design strategies.

**Keywords:** Requirements Engineering, Software Design, Decision Making, Open Software Platforms, Software Ecosystems, Open Innovation

Tjong and Berry REFSQ 2013

**Abstract.** **[Context and Motivation]** Many a tool for finding ambiguities in natural language (NL) requirements specifications (RSs) is based on a parser and a parts-of-speech identifier, which are inherently imperfect on real NL text. Therefore, any such tool inherently has less than 100% recall. Consequently, running such a tool on a NL RS for a highly critical system does not eliminate the need for a complete manual search for ambiguity in the RS. **[Question/Problem]** Can an ambiguity-finding tool (AFT) be built that has 100% recall on the types of ambiguities that are in the AFT's scope such that a manual search in an RS for ambiguities outside the AFT's scope is significantly easier than a manual search of the RS for *all* ambiguities? **[Principal Ideas/Results]** This paper presents the design of a prototype AFT, SREE (Systemized Requirements Engineering Environment), whose goal is achieving a 100% recall rate for the ambiguities in its scope, even at the cost of a precision rate of less than 100%. The ambiguities that SREE searches for by lexical analysis are the ones whose keyword indicators are found in SREE's ambiguity-indicator corpus that was constructed based on studies of several industrial strength RSs. SREE was run on two of these industrial strength RSs, and the time to do a completely manual search of these RSs is compared to the time to reject the false positives in SREE's output *plus* the time to do a manual search of these RSs for only ambiguities not in SREE's scope. **[Contribution]** SREE does not achieve its goals. However, the time comparison shows that the approach to divide ambiguity finding between an AFT with 100% recall for some types of ambiguity and a manual search for only the other types of ambiguity is promising enough to justify more work to improve the implementation of the approach. Some specific improvement suggestions are offered.