Jean-Paul van Brakel (14-720-262)

September 21, 2014

## Response paper 01

The paper *Code Context Models for Change tasks* made me quite skeptical about its proposed conclusion. This because the paper tried to tackle a hard-to-quantify concept: the construction of context models, as perceived by developers in general. Indeed, the paper appeared to struggle with bringing quantitative results and remained relatively superficial throughout, mainly describing already known characteristics of context models. Instead, it would have been interesting to use the proposed experiment, not just to describe how context models appear, but also how context models develop over time (longitudinal opposed to cross-sectional) and how the proposed tool (in the discussion) can use context models to more effectively guide the process of change tasks. For example, a statistical analysis concerning the most effective developers opposed to the least effective developers would have been a great addition. Even though the paper was quite persuasive in the methodology of the experiment, it seemed to lack a clear connection to drawn inferences (which according to both guideline papers should be avoided). The paper is therefore less powerful in its discussion section as the proposed inferences do not utilize the full potential of the performed experiment and subsequently, the acquired data. Terms like *personalised* and *the developer's current context model* are ill-defined in the context of the performed research and it's not clear how a tool could tailor-fit solutions (and be advantageous) to a developer when according to the paper "*Even for concise and successful changes, code navigation models can differ substantially on class as well as method level.*" (page 4, point 03). However, some of the inferences are elaborately discussed and have a powerful and (seemingly) valid motivation. Especially the inferences concerning *lexical similarities* and *lexical cohesion* are well-argued and provide useful insights and a strong basis for further research.

Opposed to the more objective methodology from paper 1, paper 2 (*Information Needs in Collocated Software Development Teams*) only draws from subjective observations. This created initial skepticism as the case study presented in the paper might not be very representative for the explored issue. However, my skepticism was quickly refuted as the paper proceded with an extremely detailed description (and limitations) of the performed experiment. Addition-

ally, the researchers tested their found inferences by presenting (backtesting) the results to the participants. This makes for an interesting extra perspective. For example "*Coworkers were the most frequent source of information..*", but "*.. developers rated coworker awareness (a2) as relatively unimportant, which conflicts with its frequency in our observations*". Concerning the conclusions, this paper seemed to largely agree with many of the conclusions from paper 1. Especially, it also emphasizes the difference in the thought processes among developers. However, instead of proposing tailor-fitted solutions like paper 1, it favours cooperative solutions such as *pair programming* and *development teams*. The investigated advantages are increased awareness and heightened effectiveness that cohesive teams are likely to exhibit. This paper is especially pursuasive in how it transforms the downsides of cooperation, interruptions and the lack of available information into valuable guidelines for further study and development teams in general. Eventhough the paper appears to have solid motivation for the drawn inferences, it could have been improved by employing additional statistical tools to strengthen these conclusions. Currently, the acquired data heavily relies on the capability of objective observation of the researchers (also mentioned in limitations). Post-analysis of screen capturing (like in research 1) or other objective observational methods (eye-tracking) could have decreased the subjective character of the employed analysis methodology.

Concerning the guideline papers for research in software engineering, both aforementioned papers seem to adhere to the guidelines pretty well. However, as described in *What makes good research in software engineering*, both papers seem to have a common problem: "*The low ratio of validated results appears to be a serious weakness in CS research.*". For example, paper 1 seems to lack a statistical model. This provokes the following question: *is it possible to predict how the context model of a developer looks like, given the explanatory variables?* As answers to questions like these are missing, it mismatches the advice that 'personal tools' should be developed, as the authors fail to describe analytically how such a tool would work. A similar problem exists for paper 2. According to the guideline papers, "*Good research requires not only a result, but also clear and convincing evidence that the result is sound.*". As an example, paper 2 makes strong inferences about the interdependence of coworkers while it fails to perform a sensitivity analysis on this inference (a crucial design guideline from *Preliminary guidelines for empirical research in software engineering*, page 729).
To conclude, both papers are interesting, well-written and contribute to computer science in its own way. Especially, both papers offer fresh perspectives in complex domains. ■