Dissertation Topics

Thoughts for discussion

Andrea Huber

13th of December 2018

Starting point

- Prechelt, Lutz, What Keeps the Software Systems World From Breaking Down? A qualitative analysis of knowledge gaps
- Insight 2: We lack understanding of what makes software good enough.
- Insight 3: We don't know much about what software engineers are good at.
- Insight 6: We lack a shared taxonomy of software engineering contexts.

1. Requirements Engineering in Interdisciplinary Contexts (tech – non tech)

- Problem:
- What's 'good enough' should in an ideal world be stated in the requirements, or in the Definition of Done (if working with Scrum/Kanban). Most of the time, though, the documented requirements and what's considered "good enough" do not match. The problem imo increases if there are parties involved who have different domain knowledge, such as tech vs. other domain knowledge.

1. Requirements Engineering in Interdisciplinary Contexts (tech – non tech)

- Questions:
- Who is involved when defining requirements? Who is not?
- Which items are being discussed together (e.g. to which degree are technology choices discussed with the customer)? Which are not?
- How are domain-specific requirements communicated (e.g. for a tax application)? Which images are used? Which ones work, which ones do not work?
- To which degree do both parties (software engineers and software owners) have to understand the other‘s domain?

1. Requirements Engineering in Interdisciplinary Contexts (tech – non tech)

- Questions:
- Can we describe certain problems that appear in most requirements engineering situations?
- Can we extract common best practices of
  - factors to be considered
  - people to be involved
  - imagery/language to be used

  that lead to better results?
- Can we describe certain types of context where certain strategies can be applied?

1. Requirements Engineering in Interdisciplinary Contexts (tech – non tech)

- Hypotheses:

  A) It is necessary that - to a certain degree (which? how can this be defined? can it be measured?) - the developers need to gain understanding of this domain.

  B) In the same way, the Product Owner on the customer side/customer/the other party needs to be aware (or be made aware) of the possibilities and shortcomings of software.

  C) Probably the more domain specific an application is (e.g. our tax application), the more domain specific knowledge exchange is needed.

  D) Probably it helps to develop a common glossary.

2. What are software engineers good at?

- Problem (obvious):
- We lack the terminoloy when describing what software engineers are good at / not good at. The only categories that are often used (e.g. in job adverts) concern the technology stack an engineer has experience with. Plus maybe the years of experience and previous working contexts.

2. What are software engineers good at?

- Question:
- How can we define a set of skills / a terminology or even taxonomy for skills that software engineers have?

2. What are software engineers good at?

- Hypotheses:

  A) I suspect to find (at least) these items:

  - Structure of code (more in terms of "beauty" -> creating a readable, sensible, re-usable well-organized code base),
  - Knowledge of tools (both in-depth knowledge of the tools currently at hand and a broader knowledge of the tools available),
  - Handling of requirements (interest in and grasp of domain specific knowledge other than tech), communication with stakeholders and colleagues,
  - Collaboration and work organization skills,
  - Knowledge of the architecture at all times (e.g. if I turn a screw here, in what way does it affect the other parts of my sourcecode? or even possible future parts of my software?),
  - Speed of understanding existing code and creating a mental "map" of the project

2. What are software engineers good at?

- Hypotheses:

  B) Some of these items will surely correspond or be dependent on one another.

  C) Knowing one's skills and shortcomings (and having a taxonomy at hand to describe them) will probably help a software engineer to 1. choose an appropriate team / work environment, 2. get better in the things that are considered shortcomings, 3. stress her own skills, e.g. in salary negotiation or interviews, or in order to establish an expert status (-> how could this assumption be (dis-)proved?)

  D) It is not so much important how "good" a software engineer is, if she can specify her skills/shortcomings AND she works in a team of people with the counterpart skills -> (when does a team actually do a good job?) and in the right environment (-> how can we describe context?)

No conclusion, but an excursion

- Both topics I have pursued lead to the topic of context,
- I.e. directly to Insight 6: "We lack a shared taxonomy of software engineering contexts."