The Severity of Undetected Ambiguity in Software Engineering Requirements

Cristina Ribeiro, Daniel Berry, {cribeiro, dberry}@uwaterloo.ca

1. RESEARCH PROBLEM

- Requirements specifications (RSs) must be unambiguous so that different stakeholders don't interpret them differently.
- RSs are written mostly in natural language (NL).
- NLs are inherently ambiguous.
- So all RSs must be disambiguated by stakeholders.
- For many ambiguities, just all the stakeholders' talking about the RS disambiguates them.
- But there is subconscious disambiguation (SD) and some ambiguities in an RS will survive analysis, only to show up later when the running program surprises the customer.
- The incorrect software can cause EXPENSIVE damage.
- But not all ambiguities are subconsciously disambiguated incorrectly.
- Since fixing bugs late is EXPENSIVE, it is important to try to find ALL wrongly disambiguated ambiguities EARLY.
- BUT finding ambiguities is EXPENSIVE involving multiple focused inspections.

2. RESEARCH QUESTION

Which is more EXPENSIVE?

1. Letting subconsciously disambiguated ambiguities cause their damage and then be fixed late.

OR

2. Doing enough focused inspections on the RS to find the ambiguities early before development starts.

3. RELATED WORK

de Bruijn [1], attempted to answer the same question.

de Bruijn logged all ambiguities in a random sampling of sentences in the RS for one failed project.

I am logging only all occurrences of types of ambiguities likely to suffer SD in the RSs for at least 3 successful projects.

4. METHOD

1. Get early RSs for already completed implementations.
2. Remain ignorant of their later histories.
3. For each RS:
   - A. Using checklist of ambiguity types that are most likely to be subject to SD, find as many of them as possible in RS.
   - B. Examine history of development from this RS to determine which of the found ambiguities were:
     1. found during development and were fixed and at what cost.
     2. not found during the development and are latent in the code.

        For each ambiguity in class 2:

        Determine by talking with developers
        - which ambiguities could lead to damage and
        - get estimate of cost of potential damage and cost to fix code to avoid damage.

        Note that developers should be happy (:-)) to learn of potential problems before they happen.
4. With these data, answer the RQ for these RSs.

5. REFERENCES

[1] de Bruijn, F., and Dekkers, H.: 'Ambiguity in Natural Language Software Requirements: A Case Study', Requirements Engineering: Foundation for Software Quality, pp. 233-247.