Supporting Requirements Traceability through Refactoring

Paper by Anas Mahmoud and Nan Niu
Reviewed by Ryan Darras

I. Suggestion for acceptance

Mildly accept

II. Summary

The authors discuss how after time, a once clean and organized project will lose its lexical structure due to inconsistent terminology and/or style. Their proposed solution describes a method of refactoring that can bring these projects back to a clean and organized state. The authors test their solution by applying various types of refactoring techniques and then testing the performance of requirements-to-code automated tracing methods. By refactoring, the authors hope that these automated tracing methods will be able to perform their tasks much more accurately. The authors try to find cases of problems that include missing, misplaced, and duplicated signs in software artifacts.

The authors conclude that by refactoring to restore textual information in the system, automated tracing methods perform better. They also discovered that refactoring to remove redundant information negatively impacts automated tracing methods. Also, they discovered that moving information among system modules have no significant effect.

III. Positive points

Very well formatted paper. The figures are incredibly useful as well as the mathematics.

The results and impact section is really awesome. It isn't just a generic conclusion where they list results from the previous part of the paper... It gives me something, as the reader, to get a clear description of the problem so that I can decide whether or not I need to go read the appropriate sections.

IV. Negative points

This paper used a lot of "big words" that could have easily been said in a more understandable way.

I feel as if this paper could be as descriptive and informative as it is with about 2/3rds the total number of words. Sometimes it is nice to have all this extra fluff, but not in a research paper in my opinion.

The authors only use three datasets to test their hypothesis.

V. Potential future work

The authors used three datasets and systems to test there proposed solution. I would like to see this built upon many more to get more universal results, even though the three datasets were quite different.

I think this could be a starting point to research and develop a requirements tracing system that attempts to understand the poor practices of developers such that it can continue to link requirements to code even if the updated code is garbage and ugly. This sounds like an incredibly large task to tackle, but I think it would be incredibly useful.

I'm a very stingy developer in the sense that I would prefer not to include third party applications in my solutions unless absolutely necessary. That being said, I would pay money if I could have a tracing method that would essentially build out lists for each requirement describing every modification made to the code that altered that requirement (for good or for bad). Similar to SVN or source control where you can see changes made to files, having a way to automatically organize that kind of stuff would be incredible.