Research Topic Handout – Examples

Daniel Szymczak
McMaster University

June 28, 2018

Note: This handout was created as a companion-piece to my supervisory committee meeting presentation for June 28, 2018

1 Too much duplication!

In any piece of software, the same knowledge will appear across many different artifacts. Manually entering this knowledge multiple times increases the potential for errors to occur.

From our case study on a fuel pin in a nuclear reactor, we see h_g – a symbol which appears in the Software Requirements Specification (SRS), the Literate Programmer's Manual (LPM), and the source code.

The following table shows a (simplified) definition of h_g taken from the SRS.

Number	**DD1**
Label	h_g
Equation	$h_g = \frac{2k_c h_p}{2k_c + \tau_c h_p}$
Description	h_g is the gap conductance ...
...	

$$h_g = \frac{2k_c h_p}{2k_c + \tau_c h_p} \tag{1}$$

The defining equation for h_g also appears in Equation 1, taken from the LPM, and it appears again in the corresponding C code:

```
double calc_hg(double k_c,double h_b,double tau_c)
{
 return (2*k_c*h_p) / ((2*k_c) + (tau_c * h_p));
}
```

This kind of duplication is all too common and writing one piece of information in multiple places is tedious and time-consuming. Wouldn't it make more sense to encode a definition once and automatically reuse it wherever required in our software artifacts?

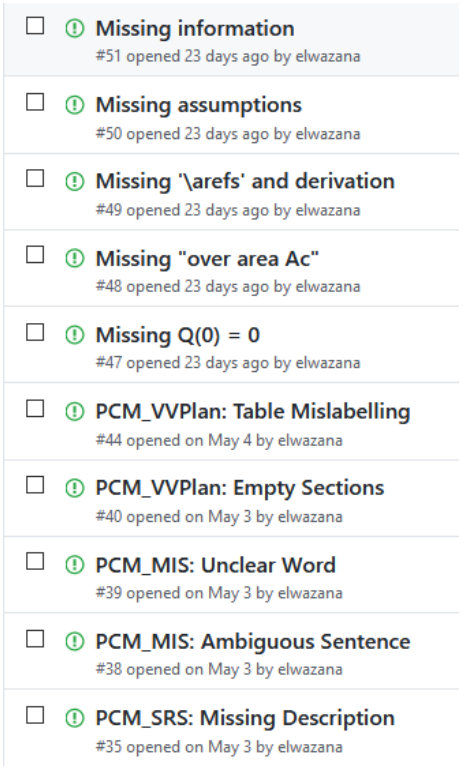

Figure 1: Subset of the currently open issues with PCM

2 Inter-/intra-artifact consistency

Our five case studies have been reviewed at least once per year by multiple people. Every time the documents are reviewed, new inconsistencies and problems with the artifacts are found. We show a subset of the issues that are currently open (Figure 1), or closed within the last two months (Figure 2) for *one* particular case study – the Solar Water Heating System with Phase Change Material, also known as PCM – but similar issues have been found across all of our case studies. If you would like to see more, the PCM case study is publicly accessible at `https://github.com/smiths/swhs`.

Each of those issues is related to some inconsistency found in the manually-produced software artifacts. Information is missing, out of date, unclear, or simply in-

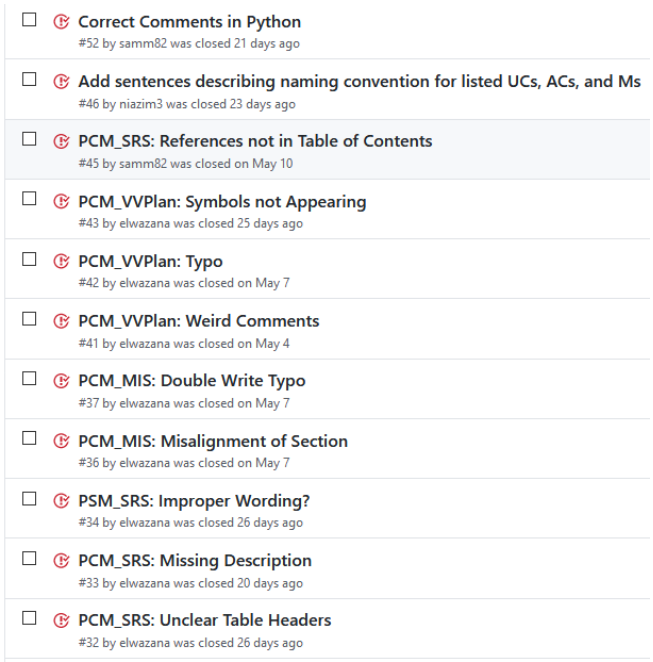

Figure 2: Closed issues within the last two months

correct. Every closed issue from Figure 2 had to be taken care of manually and took a non-trivial amount of time.

It is easy to see a problem exists with maintaining consistent artifacts. Changes made to the code, requirements, or design do not always have obvious repercussions (inter-artifacts) and inconsistencies can begin to appear. Code and documentation reviews allow us to find some such inconsistencies, but the majority of them should be avoidable in the first place if there were some way to test the effects of a change. Why do we build automated tests for code, but not other software artifacts? Why not automatically trace our artifacts and report where a given change will affect them? What if we could ensure consistency by construction?

Note: An example of intra-artifact consistency will be covered on slides #25-27 in the companion presentation.

3 Design for change

Consider the PCM case study mentioned in Section 2. It is a single member of a larger software family. If we decide to remove the phase change material from the problem, our solution will still be very similar with the obvious lack of any information related to phase change material – it becomes the noPCM case study.

A lot of PCM's design can (and should) be reused for noPCM. Not only that, but if we have designed the system with change in mind, it should be easy, if not trivial, to switch from PCM to noPCM. Why not make it easier to target software families instead of singular family members?

4 (Re-)Certification is expensive

Software certification is necessary by law in certain fields, particularly in safety-critical applications. Certifying bodies exist across domains and each have their own list of requirements. Looking at some examples [1, 2, 3, 5] there are many common requisite documentation artifacts including, but not limited to:

- Problem definition
- Theory manual
- Requirements specification
- Design description
- Verification and Validation (V&V) report

Overall certification is an expensive process. The exact costs can be hard to estimate [4], but many person-hours are spent working on ensuring full traceability and consistency between software artifacts. Getting re-certified should anything need to be updated is a similarly long and costly process as all artifacts must be updated and re-verified manually.

While we are not proposing to automatically verify our software artifacts, we want to relieve the burden of artifact creation. What if instead of manually maintaining each artifact, we could automatically generate them?

References

[1] Center for Devices and Radiological Health, CDRH. General principles of software validation; final guidance for industry and FDA staff. Technical report, US Department Of Health and Human Services Food and Drug Administration Center for Devices and Radiological Health Center for Biologics Evaluation and Research, York, England, January 2002.

[2] CSA. Quality assurance of analytical, scientific, and design computer programs for nuclear power plants. Technical Report N286.7-99, Canadian Standards Association, 178 Rexdale Blvd. Etobicoke, Ontario, Canada M9W 1R3, 1999.

[3] CSA. Guideline for the application of N286.7-99, quality assurance of analytical, scientific, and design computer programs for nuclear power plants. Technical Report N286.7.1-09, Canadian Standards Association, 5060 Spectrum Way, Suite 100, Mississauga, Ontario, Canada L4W 5N6, 1-800-463-6727, 2009.

[4] J. Hatcliff, M. Heimdahl, M. Lawford, T. Maibaum, A. Wassyng, and F. Wurden. A software certification consortium and its top 9 hurdles. *Electronic Notes in Theoretical Computer Science*, 238(4):11–17, 2009.

[5] U.S. Food and Drug Administration. Infusion pumps total product life cycle: Guidance for industry and fda staff. on-line, December 2014.