Position Paper: A Literate Framework for Scientific Software Development *

[Extended Abstract] †

Dan Szymczak
McMaster University
1280 Main Street W
Hamilton, Ontario
szymczdm@mcmaster.ca

Spencer Smith
McMaster University
1280 Main Street W
Hamilton, Ontario
smiths@mcmaster.ca

Jacques Carette
McMaster University
1280 Main Street W
Hamilton, Ontario
carette@mcmaster.ca

ABSTRACT

CCS Concepts

•**Computer systems organization** → **Embedded systems;** *Redundancy;* Robotics; •**Networks** → Network reliability;

Keywords

ACM proceedings; LaTeX; text tagging

1. INTRODUCTION

Context - first application of computers was scientific computing (SC) - the importance of SC code - used for decisions that impact health, safety and the economy - important enough to follow certification standards in some cases, CSA, Roache - problem with standards is the time and money required to meet them, resistance from practitioners (can cite Roache to support this) - many scientists seem to prefer a more agile development process (cite Carver, Kelly and Segal - see Smith book chapter for specific references), but this does not mean that this is the best process - with the proper methods and tools, scientists can follow a more structured approach, and still focus on frequent feedback and course correction - can meet documentation requirements for certification, and improve productivity.

Our goal is to have our cake and eat it too. We want to improve the qualities (verifiability, reliability, understandability etc.) especially traceability. Moreover, we want to improve productivity. Save time and money on SC software development, certification and re-certification. Improving traceability also allows us to improve reproducibility (cite Davison2012) because the decisions and details are explicitly available in the future.

To accomplish this we need to do the following:

- Remove duplication between software artifacts for scientific computing software (can cite Wilson et al DRY principle) - Provide a means for complete traceability between all artifacts

To achieve the above two goals, we propose the following:

- Provide tools to support developing scientific software using a literate process - Use artifact generation

- roadmap of the sections that are coming

2. BACKGROUND

- introductory blurb for this section

2.1 Challenges for Scientific Computing Software Quality

From Yu (2011) (PhD thesis, now in our repository) - approximation challenge [Smith advice - remove, we donâĂŹt really address this challenge with our solution] - unknown solution challenge - no test oracle [Smith advice - remove, we donâĂŹt really address this challenge with our solution] - technique selection challenge - [I think we can keep this challenge, if we later mention how LSS encourages a separation of concerns between requirements (model) and design (numerical algorithm selection). LSS also allows for experimentation with different numerical techniques.] - input output challenge [I think we should keep this one, but we should characterize it as an understandability challenge. Programmers do not reuse libraries as often as they could because they do not believe that the interface needs to be as complicated as it appears (Dubois2002). We can mention this challenge again later, because LSS will allow us to create programs, and interfaces, that are only as complicated as they need to be.] - modification challenge [I think we could relabel this as a maintainability challenge.]

2.2 Literate Programming

- overview of LP, starting with Knuth - a similar background to what we need here is given in SmithKoothoorAndNedialkov.pdf

3. INTRODUCING LSS

- Tie the introduction of the paper into the tool. - Introduce the ideas of recipes and chunks. - Explain some of

*(Produces the permission block, and copyright information). For use with SIG-ALTERNATE.CLS. Supported by ACM.

†A full version of this paper is available as *Author's Guide to Preparing ACM SIG Proceedings Using LaTeX2ε and BibTeX* at `www.acm.org/eaddress.htm`

WOODSTOCK '97 El Paso, Texas USA

DOI: 10.475/123_4

the current design? - i.e. Chunk hierarchy diagram, recipe micro/macro layout languages, code gen

3.1 Advantages

Move into examples/discussion on advantages.

3.1.1 Software Certification

- need to generate required documentation, without impeding the work of the scientists - need to be able to make changes at reasonable cost - this requires traceability

- Discuss the following documents as being "par for the course" but maybe don't include full descriptions: Start with a default set of documentation, as follows - Problem Statement - Development Plan - Requirements Specification - Verification and Validation plan - Design Specification - Code - Verification and Validation Report - User Manual

3.1.2 Knowledge Capture

- conservation of thermal energy equation - used for thermal analysis of fuel pins and then reused for solar water heating tank - Build a library of artifacts that can be reused in many different contexts - Prime example: SI Units are used everywhere.

3.1.3 "Everything should be made as simple as possible, but not simpler." (Einstein quote)

- although powerful/general commercial finite element programs are available, they are often not used to develop new âĂIJwidgetsâĂİ - reasons are cost, and complexity - rather than use simulation, engineers often resort to building prototypes and testing - engineers would greatly benefit from tools to assist their design efforts that are customized to their exact set of problems - with a literate family approach family members can be generated to fit their needs - if an engineer designs parts for strength, they could have a general stress analysis program - the program could be 3D if needed, or specialized for plane stress or plane strain, if that was the appropriate assumption - the program could even be customized to the parameterized shape of the part they are interested in, with only the degrees of freedom, like material properties, or the specific dimensions, they can change being exposed. [This section is also where we can mention the understandability challenge from Section 2.1. LSS lets us build components that provide exactly what is needed, and no more.]

3.1.4 Verification

- requirements include so-called âĂIJsanityâĂİ checks that can be reused when they come up in subsequent phases - for instance, requirement would state conservation of mass, or the fact that lengths are always positive - the first used to test output, the second to guard against invalid input

- computational variability testing, from Yu (2011), FEM example - usual to do grid refinement tests - same order of interpolation, but more points - code generation allows for increases in the order of interpolation, for the same grid - Yu discusses in section 6.3 of her thesis

3.1.5 Testing

- ???

3.2 Bringing it together with an example

- Introduce fuel pin example. - Use appendix for SRS and generated SRS?

3.2.1 How it works

- Explain the source (recipe, common knowledge, specific knowledge) - Show example recipe - Example of common knowledge (SI Units to tie into section 3.1.2) - Example of specific knowledge (h_c or h_g chunk)

3.2.2 The result?

- Bring it all back to the arguments made in 3.1.1-3.1.4 and justify them.

- 3.1.1: (Ideally) No cost to make changes, no longer impedes the scientists. - Explain in-depth

- 3.1.2: SI Units shows that common knowledge is easily stored to be used elsewhere - No need to reinvent the wheel! Keeps things consistent across projects.

- 3.1.3: (Ideally) Prototyping becomes trivial due to code gen. - Any changes to spec can be seen in (essentially) real-time.

- 3.1.4: Any mistakes that occur in the software artifacts occur everywhere. - Easier to find errors since they propagate through all the artifacts. - Documents are never out of sync with the source.

4. FUTURE WORK

- Generate more document types (more default recipes) - Include more types of information in chunks (PCM SRS example of constraints, etc.) - Auto-generate test cases! Use constraint/typical value info to create tests. - Use constraints for errors and typical values for warnings. - Expand the tool practically through examples.

5. CONCLUDING REMARKS