Literate Development of Families of Mathematical Models

Dan Szymczak

Computing and Software Department
Faculty of Engineering
McMaster University

July 13, 2015

Overview

Who am I?

Dan Szymczak

Education History

- Ph.D. Software Engineering
  - Currently in progress. Started Autumn 2014.
- M.A.Sc. Software Engineering
  - McMaster University 2014
  - Thesis – *Generating Learning Algorithms: Hidden Markov Models as a Case Study*
- B.Eng Software (Game Design)
  - McMaster University 2011

Current Program Progress

- Completed 3/4 necessary graduate courses
- Completed part one of comprehensive exam
- Research and prototype system development are underway

Research
Key Problem(s)

How can we

- improve the reuse of mathematical knowledge?
- handle knowledge duplication across software artifacts?
- improve the qualities of traceability, maintainability, verifiability and (re)usability?

Research

Musings

How can we improve the reuse of mathematical knowledge?

- Simplify the knowledge store
- Create a means of obtaining knowledge relevant to a project
- Make it accessible

Research

Musings Cont'd

Why is the duplication problem not solved yet?

- Existing tools and abstraction features only go so far
- Knowledge is shared across languages/artifacts/views
- No standard method for encoding knowledge or reusing it

Research
Musings Cont'd

Are these problems specific to math software?

No

Research

Solution Plan

- Focus: avoid knowledge duplication across artifacts through reuse
- Maintain: clear traceability between artifacts
- Utilize: generative programming to create artifacts from captured knowledge
- Expand: ideas from literate programming

Research

Solution Plan Cont'd

- Create: a domain-specific language for both knowledge capture & the artifact generator
- Implement: a feature for creating program families
- Test: apply the tool to real world problems in scientific computing

Prototype

Design and Development

- Taking a practical approach
- Focusing on knowledge reuse, as opposed to the formal capture of the knowledge itself

Prototype
How?

A practical approach

- Use existing artifacts as knowledge sources
- Motivated by concrete examples
- Avoid overdesigning and underdeveloping

Example: h_g

A simple example taken from the SRS for FP

h_g is a symbol which appears in several locations including:

- The Software Requirements Specification (SRS)
- The Literate Programmer's Manual (LPM)
- The Source Code

Let's take a look!

Example: h_g

SRS Definition for h_g (original)

Number	**DD1**
Label	h_g
Units	$ML^0 t^{-3} T^{-1}$
SI	$\frac{\mathrm{kW}}{\mathrm{m}^2(^\circ\mathrm{C})}$
Equation	$h_g = \frac{2k_c h_p}{2k_c + \tau_c h_p}$
Description	h_g is the gap conductance τ_c is the clad thickness h_p is initial gap film conductance k_c is the clad conductivity NOTE: Equation taken from the code
Sources	source code

Example: h_g

LPM Definition for h_g (original)

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

The corresponding C code is given by:

```
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)));
}
```

Example: h_g

A simple example taken from the SRS for FP

Modifying h_g to reflect changes in requirements is not simple. It involves the following steps:

- Update the definition in the SRS, LPM, etc.
- Modify the source code
- Trace all dependencies
- Modify dependents
- Ensure each artifact is now up to date and consistent

Example: h_g

Simplifying the process

Here is an example of a “chunk” for h_g:

```
--------------- --------------- --------------- ---------------
{--------------- Begin h_g ---------------}
--------------- --------------- --------------- ---------------
h_g :: Chunk
h_g = newChunk $
  [(Symbol, S "h" :-: S "g"),
   (Equation, E h_g_eq),
   (SIU, S "($\\mathrm{\\frac{kW}{m^2C}}$)"),
   (Description, S
    "effective heat transfer coefficient between clad and fuel surface")
  ]

h_g_dep :: Dependency
h_g_dep = get_dep h_g_eq

h_g_eq :: Expr
h_g_eq = ((Int 2):*(C k_c):*(C h_p)) :/ ((Int 2):*(C k_c):+((C tau_c):*(C h_p)))
```

Example: h_g

How do we generate?

What do we do with the “chunk”?
That depends on the “recipe”!

- To create our SRS we use the following recipe:

```
createSRS :: Doc
createSRS = spre $$ doctitle $$
            author auth $$ srsComms $$
            begin $$ srsBody $$ end
```

- To create our LPM we use the following recipe:

```
createLPM :: Doc
createLPM = lpre $$ doctitle $$
            author auth $$ lpmComms $$
            begin $$ lpmBody $$ endL
```

Example: h_g

Generated SRS Output

Number	**DD2**
Label	h_g
Units	$ML^0 t^{-3} T^{-1}$
SI	$\frac{\mathrm{kW}}{\mathrm{m}^{2\circ}\mathrm{C}}$
Equation	$h_g = \frac{2k_c h_p}{2k_c + \tau_c h_p}$
Description	h_g is the effective heat transfer coefficient between clad and fuel surface k_c is the clad conductivity h_p is the initial gap film conductance τ_c is the clad thickness NOTE: Equation taken from the code
Sources	source code

Example: h_g

Generated LPM Output

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

The corresponding C code is given by:

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

What next?

- Comprehensive examination part two
- Complete final graduate level course
  - Looking for a category theory course, but open to suggestions
- Complete paper for SPLASH conference
- Complete SEHPCCSE conference paper
- Complete default “recipe” for each software artifact
- Have at least on large example working from the prototype
- Create the external language for using the prototype
- Communicate with industry regarding prototype and example(s)