Toward Automatic Generation of Scientific Software Artifacts

Spencer Smith, Jacques Carette, Dan Szymczak, Steven Palmer

Computing and Software Department
Faculty of Engineering
McMaster University

SIAM CS19 MS2 Scientific Software: Practices, Concerns, and Solution Strategies, February 25, 2019

Motivation

Solution: Drasil

Bottom-up Example

Complete, Consistent, Traceable

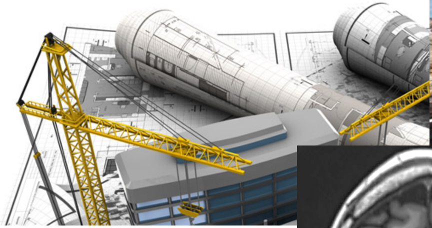
Concluding Remarks

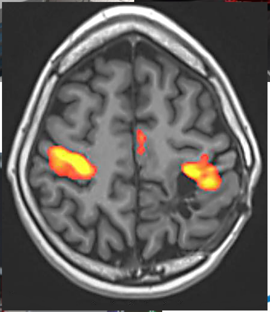
References

Abstract

- **Goal** – Improve quality of SCS
- **Idea** – Adapt ideas from SE
- **Document Driven Design**
  - Good – improves quality
  - Bad – "manual" approach is too much work
- **Solution**
  - Capture knowledge
  - Generate all things
  - Complete, consistent and traceable by construction
- **Showing great promise**
  - Significant work yet to do
  - Looking for examples/partners/enthusiasm

Scope





Motivation

Solution: Drasil

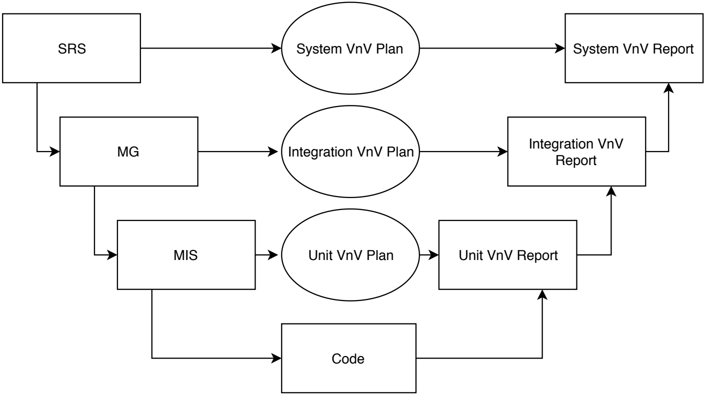
Bottom-up Example

Complete, Consistent, Traceable

Concluding Remarks

References

"Faked" Rational Design Documentation



Motivation

Solution: Drasil

Bottom-up Example

Complete, Consistent, Traceable

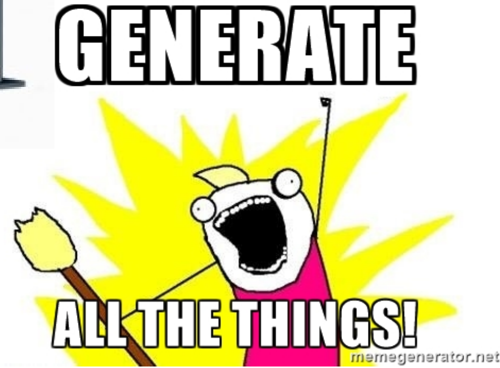
Concluding Remarks

References

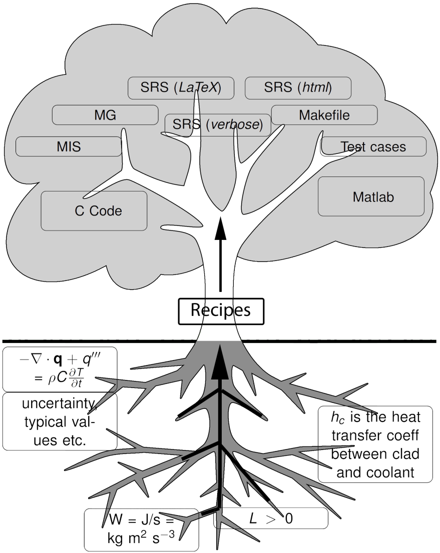
The Challenge/Problem

- Documentation provides advantages
  - Improves verifiability, reusability, reproducibility, etc.
  - From Parnas (2010)
    - easier reuse of old designs
    - better communication about requirements
    - more useful design reviews
    - etc.
  - New doc found 27 errors (Smith and Koothoor, 2016)
  - Developers see advantage (Smith et al., 2016)
- But documentation is felt to be ...
  - Too long
  - Too difficult to maintain
  - Not amenable to change
  - Too tied to waterfall process
  - Reports counterproductive (Roache, 1998)
- **The Solution?**

Knowledge Capture and Generation



Drasil



https://github.com/JacquesCarette/Drasil

Existing Approaches

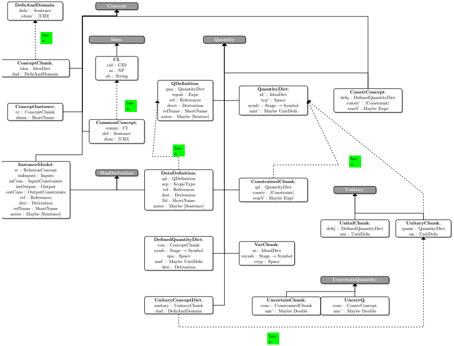
- Literate programming
  - Only code and its documentation
  - Knowledge and documentation interleaved
  - Not particularly popular
- Jupyter notebook
  - One particular view of code and documentation
  - Should be able to generate this view
- Code generation (ATLAS, Spiral, Dolphin, etc)
  - Emphasis on code
- Model driven development
  - Focus on abstract model of software design, not as much on domain knowledge
- Knowledge engineering
  - Modelling domain is not enough, need to use the knowledge
  - As model is formalized, structure is revealed

Motivation
Solution: Drasil
Bottom-up Example
Complete, Consistent, Traceable
Concluding Remarks
References

Development of Drasil

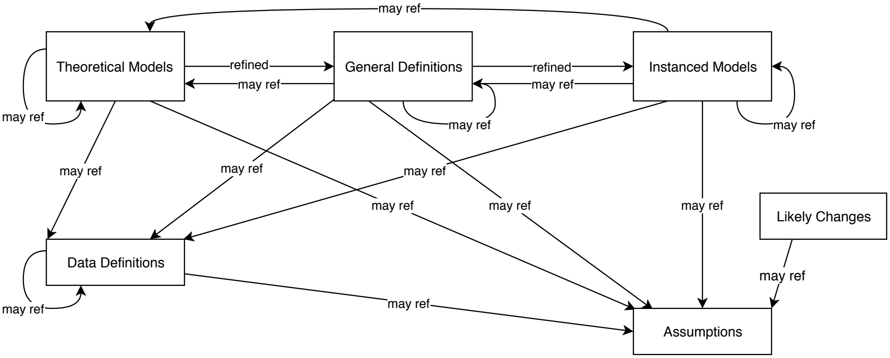
- Continual refactoring of generation to reproduce 5 “manual” case studies
  - Solar water heating system with PCM
  - Solar water heating system without PCM
  - Game physics
  - Glass pane safety analysis
  - Slope stability analysis
- Start with simple automations
  - Table of symbols
  - Cross-references, etc,
  - Scientists makes design decisions
  - etc.
- Later move to more difficult automations
  - Algorithm selection automation
  - Specified reader characteristics determine detail level of documentation
  - etc.

Implementation of Drasil



- DSLs embedded in Haskell
- Separation of concerns
  - Scientific knowledge
  - Computing knowledge
  - Documentation knowledge
  - Programming language knowledge (GOOL)

Relationship Between SRS Knowledge



J_{tol} in SRS.pdf

Refname	**DD:sdf.tol**
Label	J_{tol}
Units	
Equation	$J_{tol} = \log\left(\log\left(\frac{1}{1-P_{btol}}\right)\frac{\left(\frac{a}{1000}\frac{b}{1000}\right)^{m-1}}{k\left((E*1000)\left(\frac{h}{1000}\right)^2\right)^m * LDF}\right)$
Description	J_{tol} is the stress distribution factor (Function) based on Pbtol P_{btol} is the tolerable probability of breakage a is the plate length (long dimension) b is the plate width (short dimension) m is the surface flaw parameter k is the surface flaw parameter E is the modulus of elasticity of glass h is the actual thickness LDF is the load duration factor

J_{tol} in SRS.tex

```
\noindent \begin{minipage}{\textwidth}
\begin{tabular}{p{0.2\textwidth} p{0.73\textwidth}}
\toprule \textbf{Refname} & \textbf{DD:sdf.tol}
\phantomsection
\label{DD:sdf.tol}
\\ \midrule \\
Label & $J_{tol}$
\\ \midrule \\
Units &
\\ \midrule \\
Equation & $J_{tol}$ = $\log\left(\log\left(\frac{1}{1-P_
    {btol}}\right)\frac{\left(\frac{a}{1000}\frac{b
    }{1000}\right)^{m-1}}{k\left(\left(E*1000\right)\left
    (\frac{h}{1000}\right)^{2}\right)^{m}*LDF}\right)$
\\ \midrule \\
Description & $J_{tol}$ is the stress distribution factor
     (Function) based on
               Pbtol\newline$P_{btol}$ is the tolerable
                   probability of breakage ...
\end{minipage}\\
```

J_{tol} in SRS.html

```
<a id="">
<div class="equation">
<em>J<sub>tol</sub></em> = log(log(<div class="fraction">
<span class="fup">
1
</span>
<span class="fdn">
1 &minus; <em>P<sub>btol</sub></em>
</span>
</div>)<div class="fraction">
<span class="fup">
(<div class="fraction">
<span class="fup">
<em>a</em>
</span>
<span class="fdn">
1000
</span>
</div><div class="fraction">
...
```

J_{tol} in Python

```
def calc_j_tol(inparams):
    j_tol = math.log((math.log(1.0 / (1.0 - inparams.
        pbtol))) * ((((inparams.a / 1000.0) * (inparams.b
         / 1000.0)) ** (inparams.m - 1.0)) / ((inparams.k
         * (((inparams.E * 1000.0) * ((inparams.h /
        1000.0) ** 2.0)) ** inparams.m)) * inparams.ldf))
        )
    return j_tol
```

J_{tol} in Java

```
public static double calc_j_tol(InputParameters inparams)
    {
        double j_tol = Math.log((Math.log(1.0 / (1.0 -
            inparams.pbtol))) * ((Math.pow((inparams.a /
            1000.0) * (inparams.b / 1000.0), inparams.m -
             1.0)) / ((inparams.k * (Math.pow((inparams.E
             * 1000.0) * (Math.pow(inparams.h / 1000.0,
            2.0)), inparams.m))) * inparams.ldf)));
        return j_tol;
    }
```

Code with Comments

```
def func_B(inParams, J):
    # function 'func_B': risk of failure
    # parameter 'inParams':
    # parameter 'J': stress distribution factor (Function)

    return ((((2.86 * (10 ** (-(53)))) / ((inParams.a * inParams.b) **
      (7 - 1))) * ((((7.17 * (10 ** 7)) * 1000) *
      (inParams.h ** 2)) ** 7)) * ((3 / 60) ** (7 / 16))) *
      (math.exp(J))
```

Code with Logging

```
def func_B(inParams, J):
    # function 'func_B': risk of failure
    # parameter 'inParams':
    # parameter 'J': stress distribution factor (Function)

    outfile = open("log.txt", "w")
    print("function func_B(", end='', file=outfile)
    print(inParams, end='', file=outfile)
    print(", ", end='', file=outfile)
    print(J, end='', file=outfile)
    print(") called", file=outfile)
    outfile.close()

    return ((((2.86 * (10 ** (-(53)))) / ((inParams.a * inParams.b) **
      (7 - 1))) * ((((7.17 * (10 ** 7)) * 1000) *
      (inParams.h ** 2)) ** 7)) * ((3 / 60) ** (7 / 16))) *
      (math.exp(J))
```

J_{tol} in Drasil (Haskell)

```
tolStrDisFac_eq :: Expr
tolStrDisFac_eq = ln (ln (1 / (1 - (sy pb_tol)))
  * (((((sy plate_len)/1000.0) * ((sy plate_width)
      /1000.0)) $^ (sy sflawParamM - 1) /
    ((sy sflawParamK) * (((sy mod_elas * 1000.0) *
    (square ((sy min_thick)/1000.0))))) $^ (sy
       sflawParamM) * (sy lDurFac)))))
```

J_{tol} without Unit Conversion

```
tolStrDisFac_eq :: Expr
tolStrDisFac_eq = ln (ln (1 / (1 - (sy pb_tol)))
  * ((((sy plate_len) * (sy plate_width)) $^ (sy
      sflawParamM - 1) /
    ((sy sflawParamK) * ((sy mod_elas *
    (square (sy min_thick)))) $^ (sy sflawParamM) * (sy
        lDurFac)))))
```

Complete and Consistent

Symbol	Description	Units
A_C	Heating coil surface area	m^2
A_{in}	Surface area over which heat is transferred in	m^2
A_{out}	Surface area over which heat is transferred out	m^2
A_P	Phase change material surface area	m^2
C	Specific heat capacity	J/(kg·°C)
C^L	Specific heat capacity of a liquid	J/(kg·°C)
C^S	Specific heat capacity of a solid	J/(kg·°C)
C^V	Specific heat capacity of a vapour	J/(kg·°C)
C_W	Specific heat capacity of water	J/(kg·°C)
$C_P{}^L$	Specific heat capacity of PCM as a liquid	J/(kg·°C)
$C_P{}^S$	Specific heat capacity of PCM as a solid	J/(kg·°C)
D	Diameter of tank	m
E	Sensible heat	J
E_P	Change in heat energy in the PCM	J
E_W	Change in heat energy in the water	J
[illegible]	[illegible]	[illegible]

Complete and Consistent Cont'd

Assumptions

This section simplifies the original problem and helps in developing the theoretical model by filling in the missing information for the physical system. The numbers given in the square brackets refer to the Theoretical Models Section: Theoretical Models, General Definitions Section: General Definitions, Data Definitions Section: Data Definitions, Instance Models Section: Instance Models, Likely Changes Section: Likely Changes, or Unlikely Changes Section: Unlikely Changes, in which the respective assumption is used.

Thermal-Energy-Only: The only form of energy that is relevant for this problem is thermal energy. All other forms of energy, such as mechanical energy, are assumed to be negligible. TM: consThermE.

Heat-Transfer-Coeffs-Constant: All heat transfer coefficients are constant over time. GD: nwtnCooling.

Constant-Water-Temp-Across-Tank: The water in the tank is fully mixed, so the temperature of the water is the same throughout the entire tank. GD: rocTempSimp DD: ht_flux_P.

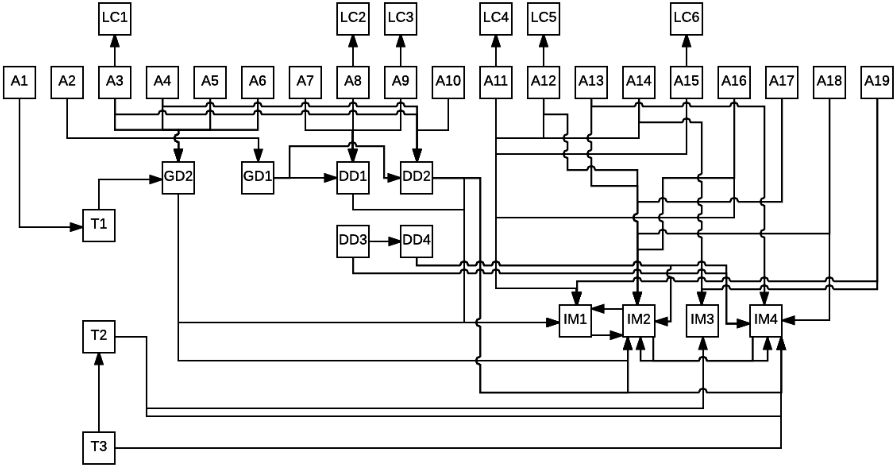
Temp-PCM-Constant-Across-Volume: The temperature of the phase change material is the same throughout the volume of PCM. GD: rocTempSimp LC: Uniform-Temperature-PCM DD: ht_flux_P.

Density-Water-PCM-Constant-over-Volume: The density of water and density of PCM have no spatial variation; that is, they are each constant over their entire volume. GD: rocTempSimp.

Specific-Heat-Energy-Constant-over-Volume: The specific heat capacity of water, specific heat capacity of PCM as a solid, and specific heat capacity of PCM as a liquid have no spatial variation; that is, they are each constant over their entire volume. GD: rocTempSimp.

Newton-Law-Convective-Cooling-Coil-Water: Newton's law of convective cooling applies between the heating coil and the water. DD: ht_flux_C.

Traceability Graph



Advantages of Drasil

- Supports changing requirements and design
  - Generation
  - Complete by construction
  - Consistent by construction
  - Automated traceability
- Supports duplication
  - Knowledge is entered once, generated/transformed
  - Eases maintenance
  - If incorrect, incorrect everywhere

Motivation

Solution: Drasil

Bottom-up Example

Complete, Consistent, Traceable

Concluding Remarks

References

Concluding Remarks

- SCS has the opportunity to lead other software fields
- Document driven design is feasible
- Requires an investment of time
- Documentation does not have to be painful
- Develop/refactor via practical case studies
- Ontology may naturally emerge
- Future work
  - Design language
  - Testing
    - Guards on input
    - Sanity checks
    - Metamorphic testing
    - Computational variability testing

References I

David Lorge Parnas. Precise documentation: The key to better software. In *The Future of Software Engineering*, pages 125–148, 2010. doi: 10.1007/978-3-642-15187-3_8. URL http://dx.doi.org/10.1007/978-3-642-15187-3_8.

Patrick J. Roache. *Verification and Validation in Computational Science and Engineering*. Hermosa Publishers, Albuquerque, New Mexico, 1998.

W. Spencer Smith and Nirmitha Koothoor. A document-driven method for certifying scientific computing software for use in nuclear safety analysis. *Nuclear Engineering and Technology*, 48(2):404–418, April 2016. ISSN 1738-5733. doi: http://dx.doi.org/10.1016/j.net.2015.11.008. URL http://www.sciencedirect.com/science/article/pii/S1738573315002582.

Motivation
Solution: Drasil
Bottom-up Example
Complete, Consistent, Traceable
Concluding Remarks
References

References II

W. Spencer Smith, Thulasi Jegatheesan, and Diane F. Kelly. Advantages, disadvantages and misunderstandings about document driven design for scientific software. In *Proceedings of the Fourth International Workshop on Software Engineering for High Performance Computing in Computational Science and Engineering (SE-HPCCE)*, November 2016. 8 pp.