Progress Report on Drasil: A Framework for Scientific Knowledge Capture and Artifact Generation

Daniel Szymczak
Computing and Software Department, McMaster University
1280 Main Street West
Hamilton, Ontario L8S 4K1
szymczdm@mcmaster.ca

Spencer Smith
Computing and Software Department, McMaster University
1280 Main Street West
Hamilton, Ontario L8S 4K1
smiths@mcmaster.ca

Jacques Carette
Computing and Software Department, McMaster University
1280 Main Street West
Hamilton, Ontario L8S 4K1
carette@mcmaster.ca

Steven Palmer
Computing and Software Department, McMaster University
1280 Main Street West
Hamilton, Ontario L8S 4K1
palmes4@mcmaster.ca

ABSTRACT

abstract here

CCS CONCEPTS

• **Mathematics of computing** → *Mathematical software*; • **Software and its engineering** → *Software development techniques*; *Automatic programming*;

KEYWORDS

scientific computing, software quality, software engineering, document driven design, code generation

ACM Reference format:
Daniel Szymczak, Spencer Smith, Jacques Carette, and Steven Palmer. 2017. Progress Report on Drasil: A Framework for Scientific Knowledge Capture and Artifact Generation. In *Proceedings of 2017 International Workshop on Software Engineering for High Performance Computing in Computational and Data-Enabled Science and Engineering, Denver, Colorado, USA, November 2017 (SE-CSE_SE-CoDeSE),* 4 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

- goal to improve the quality of Scientific Computing Software (SCS) - in particular the qualities of certifiability, reusability and reproducibility for large multi-year, multi-developer projects where the end users do much of the development - especially important to improve quality where correctness impacts safety - examples of nuclear safety analysis and medical imaging

- improved documentation is an important part of improved quality, but too much time and effort for SCS developers, especially when they are often dealing with rapid change. SCS developers tend to dislike documentation. As observed by Carver [1], scientists do not view rigid, process-heavy approaches, favourably. Moreover, they often consider reports for each stage of software development as counterproductive [4, p. 373].

- Documentation provides advantages
  - Improves verifiability, reusability, reproducibility, etc.
  - From [3]
    - easier reuse of old designs
    - better communication about requirements
    - more useful design reviews
    - etc.
  - New doc found 27 errors [6]
  - Developers see advantage [5]
- But documentation is felt to be ...
  - Too long
  - Too difficult to maintain
  - Not amenable to change
  - Too tied to waterfall process
  - Reports counterproductive [4]
- **The Solution?**

Drasil - a solution proposed in a position paper [7] - capture the scientific and documentation knowledge and then generate the code and other software artifacts

- work on Drasil has continued since the position paper, as described below - provide a roadmap of the paper

2 DESIGN OF DRASIL

- an overview, with an emphasis on the notion of chunks would be good - should mention which DSLs are part of DSL - maybe the ontology figure? - should mention GOOL

3 DEVELOPMENT PROCESS FOR DRASIL

- concurrent development of 5 different examples - list the examples - mention that they overlap with [5] - use of GitHub - peer review of code - issue tracking - refactoring - finding patterns - knowledge extraction - reduction of duplication

SE-CSE_SE-CoDeSE, November 2017, Denver, Colorado, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

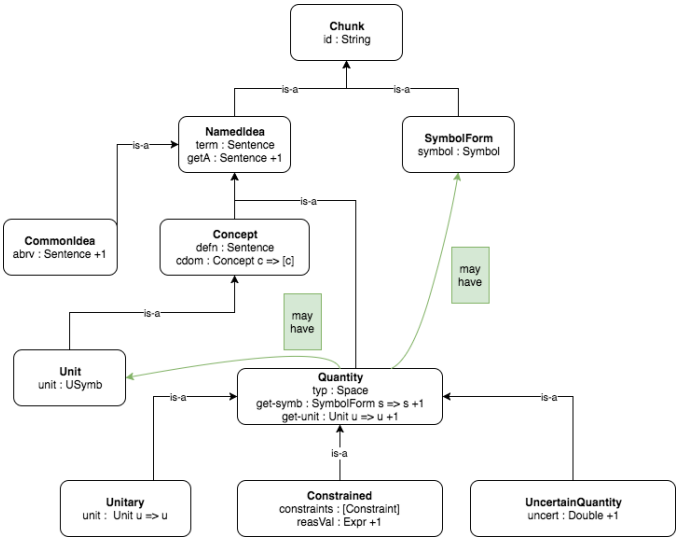

Figure 1: Relationships Between Drasil Classes

4 GLASSBR EXAMPLE

- introduce example from Civil Engineering - say what the inputs are to GlassBR and what it calculates - bottom up approach to presentation - start with chunks, build up to SRS, traceability

- start with data definition for Jtol generated by Drasil

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

Figure 2: J_{tol} from GlassBR Requirements

- figure comes from tex generated by Drasil. Can also generate html. This figure is part of the documentation of the requirements. Eventually need code to calculate Jtol. Can generate code like in the next figure. [this needs to be fit into one column —SS]

- can also generate Java, Lua, etc.

- next show the source file in Drasil. [need to fit in one column —SS]

Now notice a mistake in code - shouldn't divide by 1000 - redo - fixes the mistake everywhere

All of the knowledge on GlassBR can be put together to generate the software requirements specification. Can point to a figure

```
def calc_j_tol(inparams):
    j_tol = math.log((math.log(1.0/(1.0 - inparams.pbto
* ((((inparams.a / 1000.0) * (inparams.b / 1000.0)) **
    return j_tol
```

Figure 3: Python code to Calculate J_{tol}

```
stressDistFac = makeVC "stressDistFac" (nounPhraseSP
  $ "stress distribution" ++ " factor (Function)") cJ

sdf_tol = makeVC "sdf_tol" (nounPhraseSP $
  "stress distribution" ++
  " factor (Function) based on Pbtol")
  (sub (stressDistFac ^. symbol) (Atomic "tol"))

tolStrDisFac_eq :: Expr
tolStrDisFac_eq = log (log ((1) / ((1) - (C pb_tol)))
  * ((Grouping (((C plate_len) / (1000)) * ((C plate_wi
  ((C sflawParamM) - (1)) / ((C sflawParamK) *
  (Grouping (Grouping ((C mod_elas) * (1000)) *
  (square (Grouping ((C act_thick) / (1000))))
  )) :^ (C sflawParamM) * (C loadDF))))

tolStrDisFac :: QDefinition
tolStrDisFac = mkDataDef sdf_tol tolStrDisFac_eq
```

Figure 4: Drasil (Haskell) code for J_{tol} Knowledge

```
tolStrDisFac_eq :: Expr
tolStrDisFac_eq = log (log ((1) / ((1) - (C pb_tol)))
  * ((Grouping ((C plate_len) * (C plate_width))) :^
  ((C sflawParamM) - (1)) / ((C sflawParamK) *
  (Grouping ((C mod_elas) * (square (C act_thick)))) :^
```

Figure 5: Modified Drasil (Haskell) code for J_{tol}

Table 1: Constraints on quantities Used To Verify Inputs

Var	Constraints	Typical Value	Uncertainty
L	$L > 0$	1.5 m	10%
ρ_P	$\rho_P > 0$	1007 kg/m^3	10%

showing the table of contents for the SRS. Explain that it can be generated in tex (pdf) or html.

Part of SRS is automatically generated traceability information between definitions, assumptions, theories and instanced models.

1 Reference Material **3**
1.1 Table of Units 3
1.2 Table of Symbols 3
1.3 Abbreviations and Acronyms 4

2 Introduction **5**
2.1 Purpose of Document 5
2.2 Scope of Requirements 5
2.3 Characteristics of Intended Reader 6
2.4 Organization of Document 6

3 Stakeholders **6**
3.1 The Client 6
3.2 The Customer 6

4 General System Description **6**
4.1 User Characteristics 7
4.2 System Constraints 7

5 Scope of the Project **7**
5.1 Product Use Case Table 7
5.2 Individual Product Use Cases 7

6 Specific System Description **8**
6.1 Problem Description 8
6.1.1 Terminology and Definitions 8
6.1.2 Physical System Description 10
6.1.3 Goal Statements 10
6.2 Solution Characteristics Specification 10
6.2.1 Assumptions 10
6.2.2 Theoretical Models 12
6.2.3 General Definitions 13
6.2.4 Data Definitions 13
6.2.5 Instance Models 17
6.2.6 Data Constraints 18

7 Requirements **19**
7.1 Functional Requirements 19
7.2 Non-Functional Requirements 21

8 Likely Changes **21**

Figure 6: Table of Contents for Generated SRS for GlassBR

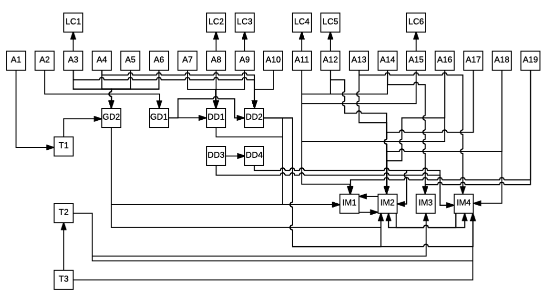

Figure 7: Traceability Graph

5 QUALITY IMPROVEMENTS

5.1 Certifiability

$$E_W = \int_0^t h_C A_C (T_C - T_W(t))dt - \int_0^t h_P A_P (T_W(t) - T_P(t))dt$$

- *If wrong, wrong everywhere*
- Sanity checks captured and reused
- Generate guards against invalid input
- Generate test cases
- Generate view suitable for inspection
- Traceability for verification of change

5.2 Reusability

- De-embed knowledge
- Reuse throughout document
  - Units
  - Symbols
  - Descriptions
  - Traceability information
- Reuse between documents
  - SRS
  - MIS
  - Code
  - Test cases
- Reuse between projects
  - Knowledge reuse
  - A family of related models, or reuse of pieces
  - Conservation of thermal energy
  - Interpolation
  - Etc.

5.3 Reproducibility

- Usual emphasis is on reproducing code execution
- However, [2] show reproducibility challenges due to undocumented:
  - Assumptions
  - Modifications
  - Hacks
- Shouldn't it be easier to independently replicate the work of others?
- Require theory, assumptions, equations, etc.
- Drasil can potentially check for completeness and consistency

6 FUTURE WORK

7 CONCLUDING REMARKS

8 ACKNOWLEDGMENTS

The assistance of McMaster University's Dr. Manuel Campidelli, Dr. Wael and Dr. Michael Tait with the GlassBR example is greatly appreciated.

REFERENCES

[1] Jeffrey C. Carver, Richard P. Kendall, Susan E. Squires, and Douglass E. Post. 2007. Software Development Environments for Scientific and Engineering Software: A Series of Case Studies. In *ICSE '07: Proceedings of the 29th International Conference on Software Engineering*. IEEE Computer Society, Washington, DC, USA, 550–559. https://doi.org/10.1109/ICSE.2007.77

[2] Cezar Ionescu and Patrik Jansson. 2012. Dependently-Typed Programming in Scientific Computing — Examples from Economic Modelling. In *Revised Selected Papers of the 24th International Symposium on Implementation and Application of Functional Languages (Lecture Notes in Computer Science)*, Vol. 8241. Springer International Publishing, 140–156. https://doi.org/10.1007/978-3-642-41582-1_9

[3] David Lorge Parnas. 2010. Precise Documentation: The Key to Better Software. In *The Future of Software Engineering*. 125–148. https://doi.org/10.1007/978-3-642-15187-3_8

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

[5] W. Spencer Smith, Thulasi Jegatheesan, and Diane F. Kelly. 2016. 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).* 8 pp.

[6] W. Spencer Smith and Nirmitha Koothoor. 2016. A Document-Driven Method for Certifying Scientific Computing Software for Use in Nuclear Safety Analysis. *Nuclear Engineering and Technology* 48, 2 (April 2016), 404–418. https://doi.org/10.1016/j.net.2015.11.008

[7] Daniel Szymczak, W. Spencer Smith, and Jacques Carette. 2016. Position Paper: A Knowledge-Based Approach to Scientific Software Development. In *Proceedings of SE4Science'16 in conjunction with the International Conference on Software Engineering (ICSE)*. In conjunction with ICSE 2016, Austin, Texas, United States. 4 pp.