Position Paper: A Knowledge-Based Approach to Scientific Software Development

Dan Szymczak, **Spencer Smith** and Jacques Carette

Computing and Software Department
Faculty of Engineering
McMaster University

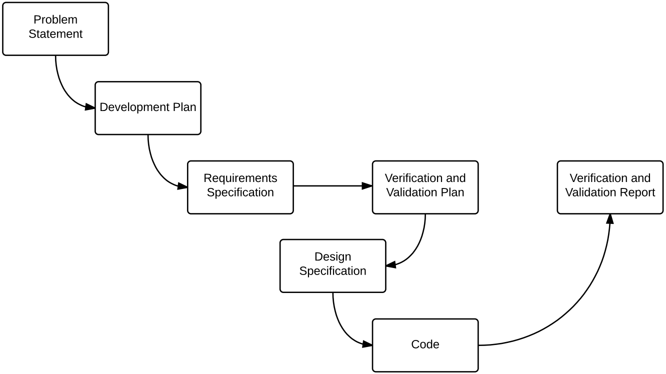
SE4Science, May. 16, 2016

Knowledge-Based Doc Driven Design (DDD)

1. Position
2. DDD Benefits
3. Challenges for DDD
4. Solution – Knowledge Based Approach (KBA)
   - Addresses Challenges
   - Benefits
5. Feasibility (Introducing Drasil)
6. Future Work
7. Conclusions

Knowledge-Based DDD

- DDD leads to high quality SCS
- Knowledge Based Approach
  - Facilitates DDD
  - Provides benefits



Problem Statement

Development Plan

Requirements Specification

Verification and Validation Plan

Verification and Validation Report

Design Specification

Code

Benefits of DDD

- Improve qualities
  - Verifiability
  - Maintainability
  - Reusability
  - Reproducibility
- Better communication
- How and Why to Fake It (Parnas and Clements, 1996)

Reasons “Manual” DDD is Unpopular

- Up front requirements are challenging
- Rapid change for numerical algorithms
- Information duplication
- Synchronization headaches between artifacts
- Perceived over-emphasis on non-executable artifacts

Knowledge Based Approach

- Capture knowledge
- From one “source” recipes to generate artifacts
- Automated
- Inspired by Knuth’s Literate Programming

How Addresses Challenges

- Supports changing requirements and design
  - Generation
  - Automated traceability
- Supports duplication
  - Knowledge is entered once, generated/transformed
  - Eases maintenance
  - If incorrect, incorrect everywhere
- Non-executable artifacts are generated

Verifiability

Var	Constraints	Typical Value	Uncertainty
L	$L > 0$	1.5 m	10%
D	$D > 0$	0.412 m	10%
V_P	$V_P > 0$	0.05 m^3	10%
A_P	$A_P > 0$	1.2 m^2	10%
ρ_P	$\rho_P > 0$	1007 kg/m^3	10%

- Sanity checks captured and reused
- Generate guards against invalid input
- Generate test cases

Reusability

Number	**T1**
Label	**Conservation of energy**
Equation	$-\nabla \cdot \mathbf{q} + q''' = \rho C \frac{\partial T}{\partial t}$
Description	The above equation gives the conservation of energy for time varying heat transfer in a material of specific heat capacity C and density ρ, where $\mathbf{q}$ is the thermal flux vector, q''' is the volumetric heat generation, T is the temperature, ∇ is the del operator and t is the time.

Usability

- As simple as possible, but not simpler (Einstein)
- Usability challenges for general purpose SCS
  - Complex, confusing
  - Generic symbols and terminology
- Generate apps suited to specific scientific and engineering needs
- Finite element software example

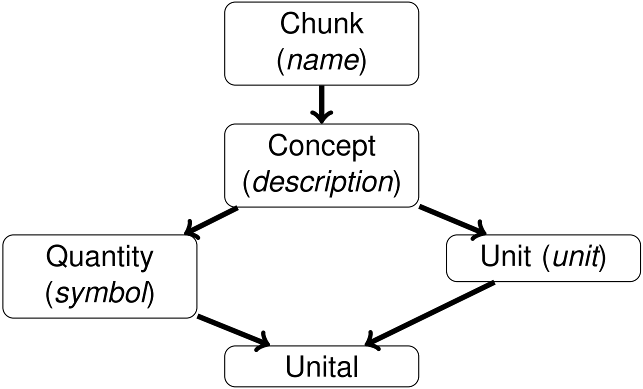
Reproducibility

- Knowledge is explicitly stored for the future
- Recipes can be use to regenerate any artifacts
- Recipes include build instructions

Software Certification

- Recertification can be expensive and time consuming
- Change propagates through documentation
- Recipes help with changing documentation standards

Drasil Framework Design



Example Recipe

```
srsBody = srs [h_g, h_c] "Spencer Smith" [s1,s2]

s1 = Section (S "Table of Units") [intro, table]

table = Table
 [S "Symbol", S "Description"] (mkTable
   [(\x -> Sy (x ^. unit)),
    (\x -> S (x ^. descr)) ] si_units)

intro = Paragraph (S "Throughout this ...")
```

Reusable Chunks

```
metre, second, kelvin :: FundUnit
metre  = fund ”Metre”  ”length (metre)”           ”m”
second = fund ”Second” ”time (second)”            ”s”
kelvin = fund ”Kelvin” ”temperature (kelvin)”     ”K”
```

The h_c Chunk

```
h_c_eq :: Expr
h_c_eq = 2*(C k_c)*(C h_b) /
  (2*(C k_c) + (C tau_c)*(C h_b))

h_c :: EqChunk
h_c = fromEqn "h_c"
 "convective heat transfer coefficient between
    clad and coolant"
   (sub h c) heat_transfer h_c_eq
```

Next Steps

- Generate more artifact types
- Generate different document views
- More types of information in chunks
- Use constraints to generate test cases
- Implement larger examples

Conclusions

- SCS has the opportunity to lead other software fields by leveraging its solid existing knowledge base
- DDD is feasible with a knowledge-based approach
- Documentation for QA and software certification does not have to be painful, expensive or time consuming
- Drasil will be developed via practical case studies