Drasil: From generating code to generating software

Jacques Carette, Spencer Smith, Dan Szymczak and Steven Palmer

McMaster University

WG 2.11, August 2016 Meeting
GENERATE

ALL THE THINGS!
Context

software certification
Context

software (re)certification
software (re)certification

- All software artefacts as evidence:
  - requirements, software specification, software design, code, tests, “theory manual”, user manual, ...
software (re)certification

- All software artefacts as evidence:
  - requirements, software specification, software design, code, tests, “theory manual”, user manual, ...

- Massive amounts of knowledge duplication
  - Implies that either
    - non-code artefacts do not get maintained well enough, OR
    - are felt to be an expensive nuisance
  - duplication harms traceability
Example SRS/LP

(see document)
Literate Programming

What can we learn from it?
What can we learn from it?

1. Code in most languages is not well organized for human understanding.
Literate Programming

What can we learn from it?

1. Code in most languages is not well organized for human understanding.
2. Code in some languages can not efficiently be broken down into very small pieces.
Literate Programming

What can we learn from it?

1. Code in most languages is not well organized for human understanding.
2. Code in some languages can not efficiently be broken down into very small pieces.
3. Chunk labels add convenient traceability information.
Ideas behind our prototype:

1. no information duplication
Ideas behind our prototype:

1. no information duplication
2. Recipes used to weave together information into documents / software artefacts.
Drasil

Ideas behind our prototype:

1. no information duplication
2. Recipes used to weave together information into documents / software artefacts.

Implies:
- Bug in one place, bugs everywhere!
Drasil

Ideas behind our prototype:

1. no information duplication
2. Recipes used to weave together information into documents / software artefacts.

Implies:

- Bug in one place, bugs everywhere!
- Huge up-front investment.
Drasil

Ideas behind our prototype:

1. no information duplication
2. Recipes used to weave together information into documents / software artefacts.

Implies:
- Bug in one place, bugs everywhere!
- Huge up-front investment.
- Doesn’t work if you have no theory.
Example (high level)

- **SRS (verbose)**
- **SRS (LaTeX)**
- **SRS (html)**
- **MG**
- **MIS**
- **Test cases**
- **C Code) (checks)**
- **Makefile**
- **Matlab (no checks)**
- **Recipes**

\[-\nabla \cdot \mathbf{q} + q''' = \rho C \frac{\partial T}{\partial t}\]

- Uncertainty, typical values etc.

\[W = J/s = \text{kg m}^2 \text{s}^{-3}\]

\[L > 0\]

\[h_c\] is the heat transfer coeff between clad and coolant.
Sanity checks

<table>
<thead>
<tr>
<th>Var</th>
<th>Constraints</th>
<th>Typical Value</th>
<th>Uncertainty</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L$</td>
<td>$L &gt; 0$</td>
<td>1.5 m</td>
<td>10%</td>
</tr>
<tr>
<td>$D$</td>
<td>$D &gt; 0$</td>
<td>0.412 m</td>
<td>10%</td>
</tr>
<tr>
<td>$V_P$</td>
<td>$V_P &gt; 0$</td>
<td>0.05 m$^3$</td>
<td>10%</td>
</tr>
<tr>
<td>$A_P$</td>
<td>$A_P &gt; 0$</td>
<td>1.2 m$^2$</td>
<td>10%</td>
</tr>
<tr>
<td>$\rho_P$</td>
<td>$\rho_P &gt; 0$</td>
<td>1007 kg/m$^3$</td>
<td>10%</td>
</tr>
</tbody>
</table>

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

- Sanity checks captured and reused
- Generate guards against invalid input
- Generate test cases
<table>
<thead>
<tr>
<th>Ref</th>
<th>T1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label</td>
<td><strong>Conservation of energy</strong></td>
</tr>
<tr>
<td>Eq</td>
<td>[-\nabla \cdot \mathbf{q} + q'''' = \rho C \frac{\partial T}{\partial t}]</td>
</tr>
<tr>
<td>Desc.</td>
<td>Conservation of energy for time varying heat transfer in a material of specific heat capacity $C$ and density $\rho$, where $\mathbf{q}$ is the thermal flux vector, $q''''$ is the volumetric heat generation, $T$ is the temperature, $\nabla$ is the del operator and $t$ is time.</td>
</tr>
</tbody>
</table>
Basic Drasil Design

- Chunk *(name)*
- Concept *(description)*
- Quantity *(symbol)*
- Unital
- RelationChunk *(relation)*
- DefEqChunk *(equation)*
- Unit *(unit)*
Example Recipe

vars :: [EqChunk]
vars = [h_g, h_c]

s1, s2, s3, s4 :: LayoutObj
s1=table_of_units si_units
s2=table_of_symbols vars
s3=Section 0 (S "Data Definitions") $ map (Definition.Data) vars
s4=Section 0 (S "Code") $ map (CodeBlock.toCode CLang Calc) [h_c]

srs :: Quantity s => [s] -> String -> [LayoutObj] -> Document
srs ls author body =
  Document ((S "SRS for ") :+: (foldr1 (:+:) (intersperse (S " and ")
    (map (\x -> U $ x ^. symbol) ls )))
  (S author) body

srsBody :: Document
srsBody = srs vars "Spencer Smith" [s1, s2, s3, s4]
Example Recipe

table_of_symbols :: (Unit s, Quantity s) => [s] -> LayoutObj
table_of_symbols ls=Section 0 (S "Table of Sym") [intro, table ls]

table :: (Unit s, Quantity s) => [s] -> LayoutObj
table ls=Table [S "Symbol", S "Description", S "Units"] (mkTable
  [(
    \ch -> U (ch ^. symbol)),
  (\ch -> ch ^. descr),
  (\ch -> Sy $ ch ^. unit)] ls)
  (S "Table of Symbols") False
Example Recipe

table_of_symbols :: (Unit s, Quantity s) => [s] -> LayoutObj
table_of_symbols ls = Section 0 (S "Table of Sym") [intro, table ls]

table :: (Unit s, Quantity s) => [s] -> LayoutObj
table ls = Table [S "Symbol", S "Description", S "Units"] (mkTable
    [(
        \ch -> U (ch ^. symbol)),
        (\ch -> ch ^. descr),
        (\ch -> Sy $ ch ^. unit)] ls)
    (S "Table of Symbols") False

Classy Optics

class Chunk c where
    name :: Simple Lens c String
class Chunk c => Concept c where
    descr :: Simple Lens c Sentence
Units Recipe

fundamentals :: [FundUnit]
fundamentals = [metre, kilogram, second, kelvin, mole, ampere, candela]

derived :: [DerUChunk]
derived = [centigrade, joule, watt, calorie, kilowatt]

si_units :: [UnitDefn]
si_units = map

Fundamental SI Units

fund :: String -> String -> String -> FundUnit
fund nam desc sym = UD (CC nam (S desc)) (UName $ Atomic sym)

metre, kilogram, second, kelvin, mole, ampere, candela :: FundUnit
metre = fund "Metre" "length" "m"
kilogram = fund "Kilogram" "mass" "kg"
second = fund "Second" "time" "s"
kelvin = fund "Kelvin" "temperature" "K"
mole = fund "Mole" "amount of substance" "mol"
ampere = fund "Ampere" "electric current" "A"
candela = fund "Candela" "luminous intensity" "cd"
The $h_c$ Chunk

\[
    h_c = \frac{2k_c h_b}{2k_c + \tau_c h_b}
\]

\[
    \text{heat\_transfer} :: \text{DerUChunk}
\]

\[
    \text{heat\_transfer} = \text{DUC (UD ht\_con ht\_symb)} \text{heat\_transfer\_eqn}
\]

ht\_con :: ConceptChunk

ht\_con = \text{makeCC "Heat transfer" "Heat transfer"}

ht\_symb :: USymb

ht\_symb = \text{from\_udefn heat\_transfer\_eqn}

heat\_transfer\_eqn = \text{USynonym (UProd}

[\text{kilogram} ^ . \text{unit}, \text{UPow (second} ^ . \text{unit} (-3),

\text{UPow (centigrade} ^ . \text{unit} (-1))]

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 = \text{fromEqn "h\_c" (S "convective heat transfer ...")}

(IH 'sub' IC) heat\_transfer h\_c\_eq
Approach

- Case studies
  - Solar water heating tank
  - Slope stability analysis
  - Glass safety analysis
  - Game physics engine
  - (medium-sized industrial code)
- Small chunks of knowledge
- Aggressively look for patterns and capture
- Currently working on capturing design decisions
Approach

- Case studies
  - Solar water heating tank
  - Slope stability analysis
  - Glass safety analysis
  - Game physics engine
  - (medium-sized industrial code)
- Small chunks of knowledge
- Aggressively look for patterns and capture
- Currently working on capturing design decisions