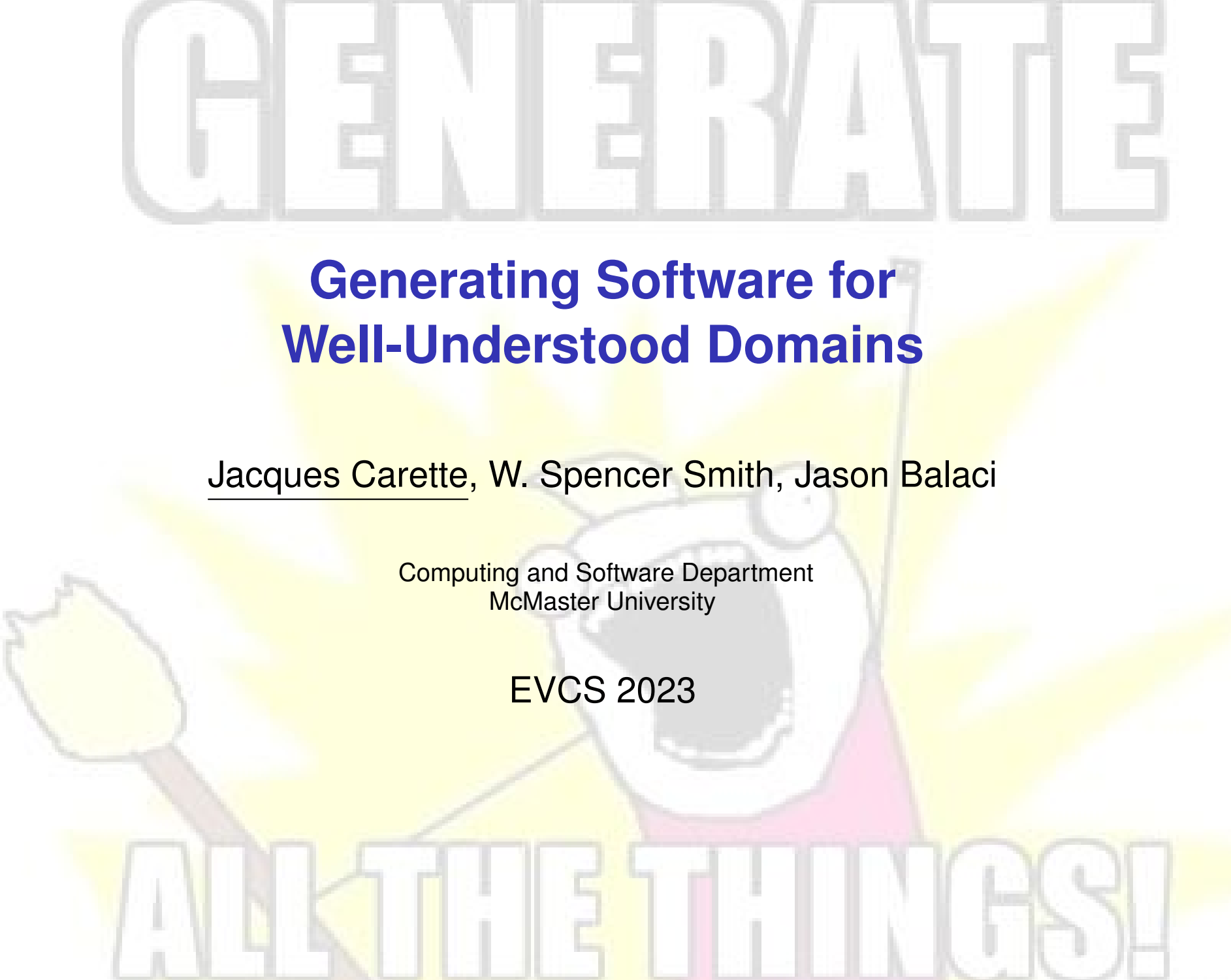


Generating Software for Well-Understood Domains

Jacques Carette, W. Spencer Smith, Jason Balaci

Computing and Software Department
McMaster University

EVCS 2023

Old Fashioned?

I enjoyed reading this short paper. It is a very classical paper, reminiscing of the DSL systems of the early days, like Neighbor's Draco and Baxter's DMS. Indeed, it is an old vision to represent domain knowledge first-class, to avoid duplication, and offer variation points to make engineering trade-offs.

– ***Referee 2***

As general feedback, the paper's message reads a bit "old-fashioned". The "well-understood" qualification is nice (because it qualifies what would otherwise be overly-general statements) but the actual benefit claimed is straight out of the playbook of the "automatic programming" community of the 80s and 90s. For instance, I was strongly reminded of Novak's GLisp...

– ***Referee 3***

Jacques Carette, W. Spencer Smith, Jason Balaci

Well Understood?

Given $F, Q, \kappa, \phi, \gamma$ calculate:

$$\mathbf{K} = \int_V \mathbf{B}^T \mathbf{D}^{vp} \mathbf{B} dV; \mathbf{F} = \mathbf{R}_i - \int_V \mathbf{B}^T \boldsymbol{\sigma}_i dV + \int_V \mathbf{B}^T \Delta \boldsymbol{\sigma}^{vp} dV \quad (1)$$

with

$$\mathbf{D}_{vp} = \mathbf{D}\left[\mathbf{I} - \Delta t C_1 \lambda' \frac{\partial Q}{\partial \boldsymbol{\sigma}} \left(\frac{\partial F}{\partial \boldsymbol{\sigma}}\right)^T \mathbf{D}\right], \lambda' = \frac{d\lambda}{dF} \quad (2)$$

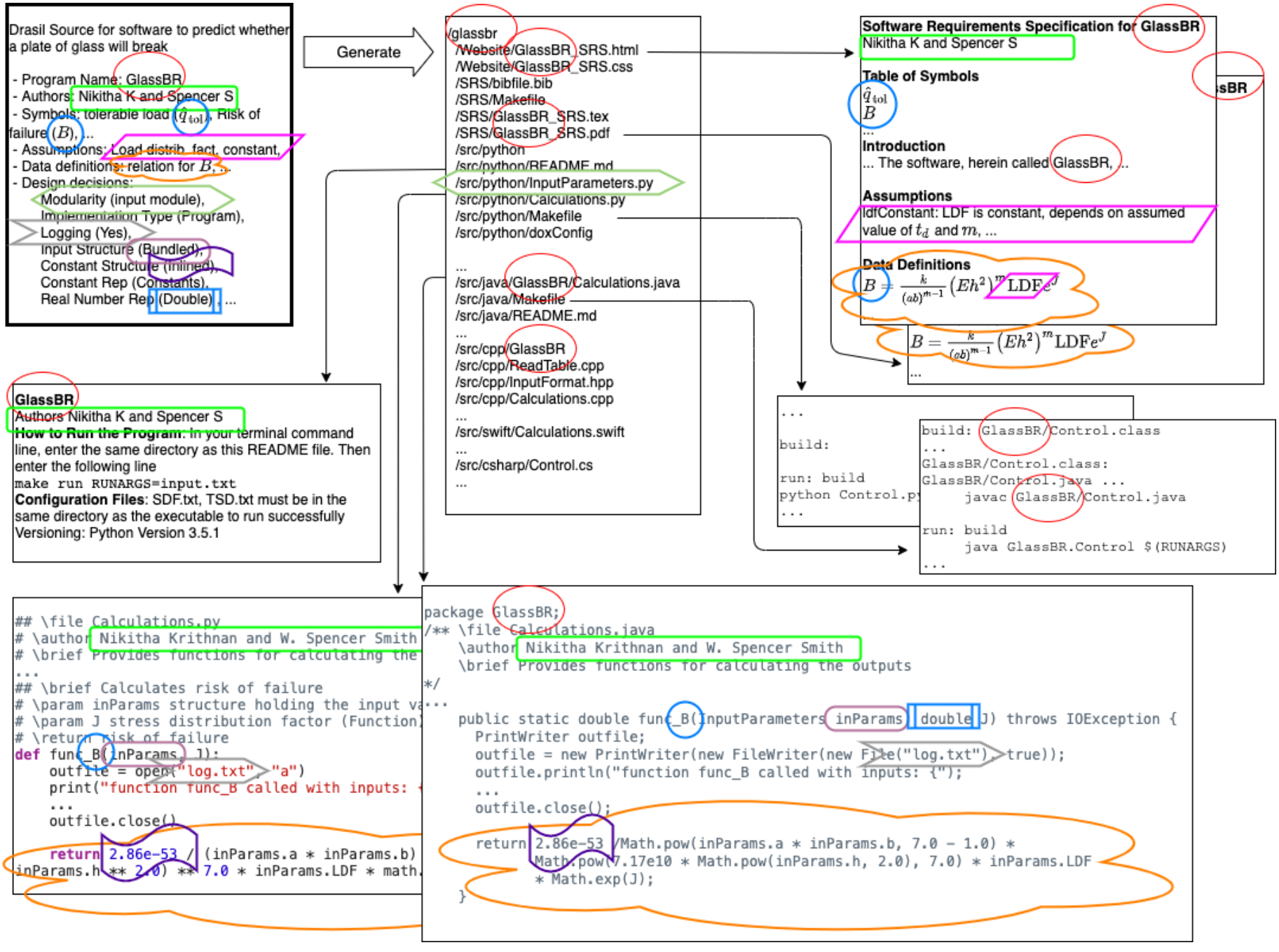
$$\Delta \boldsymbol{\sigma}^{vp} = \Delta t C_1 \lambda \mathbf{D} \frac{\partial Q}{\partial \boldsymbol{\sigma}} \quad (3)$$

$$C_1 = [1 + \lambda' \Delta t (H_e + H_p)]^{-1} \quad (4)$$

$$H_e = \left(\frac{\partial F}{\partial \boldsymbol{\sigma}}\right)^T \mathbf{D}\left(\frac{\partial Q}{\partial \boldsymbol{\sigma}}\right) \quad (5)$$

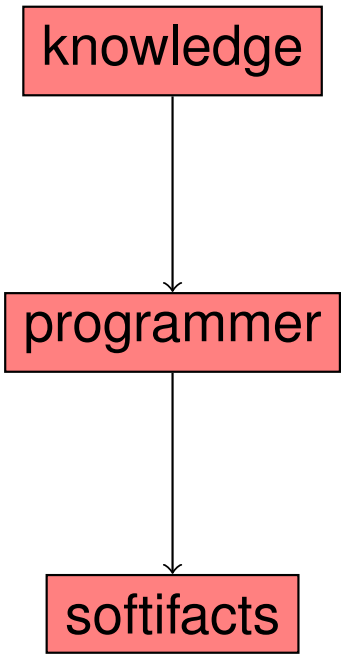
$$H_p = -\frac{\partial F}{\partial \kappa} \left(\frac{\partial \kappa}{\partial \boldsymbol{\epsilon}^{vp}}\right)^T \frac{\partial Q}{\partial \boldsymbol{\sigma}} \quad (6)$$

Well Understood!

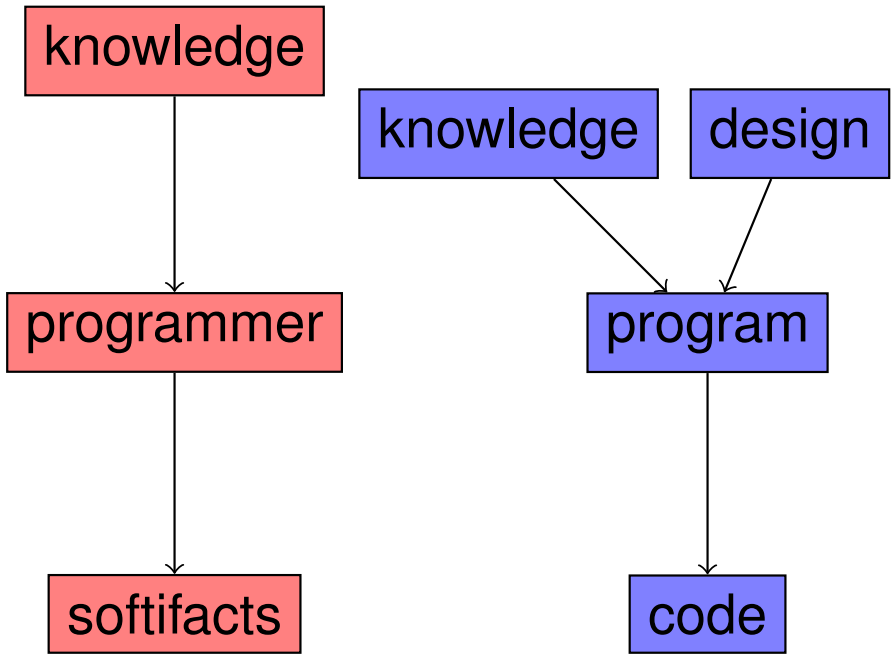

Drasil Source for software to predict whether a plate of glass will break
- Program Name: GlassBR
- Authors: Nikitha K and Spencer S
- Symbols: tolerable load ($\hat{q}_{tol}$), Risk of failure (B), ...
- Assumptions: Load distrib. fact. constant, ...
- Data definitions: relation for B, ...
- Design decisions:
Modularity (input module),
Implementation Type (Program),
Logging (Yes),
Input Structure (Bundled),
Constant Structure (Inlined),
Constant Rep (Constants),
Real Number Rep (Double), ...
Generate
/glassbr
/Website/GlassBR_SRS.html
/Website/GlassBR_SRS.css
/SRS/bibfile.bib
/SRS/Makefile
/SRS/GlassBR_SRS.tex
/SRS/GlassBR_SRS.pdf
/src/python
/src/python/README.md
/src/python/InputParameters.py
/src/python/Calculations.py
/src/python/Makefile
/src/python/doxConfig
...
/src/java/GlassBR/Calculations.java
/src/java/Makefile
/src/java/README.md
...
/src/cpp/GlassBR
/src/cpp/ReadTable.cpp
/src/cpp/InputFormat.hpp
/src/cpp/Calculations.cpp
...
/src/swift/Calculations.swift
...
/src/csharp/Control.cs
...
Software Requirements Specification for GlassBR
Nikitha K and Spencer S
Table of Symbols
$\hat{q}_{tol}$
B
...
Introduction
... The software, herein called GlassBR, ...
Assumptions
ldfConstant: LDF is constant, depends on assumed value of t_d and m, ...
Data Definitions
$B = \frac{k}{(ab)^{m-1}}(Eh^2)^m \text{LDF} e^J$
$B = \frac{k}{(ab)^{m-1}}(Eh^2)^m \text{LDF} e^J$
...
GlassBR
Authors Nikitha K and Spencer S
How to Run the Program: In your terminal command line, enter the same directory as this README file. Then enter the following line
make run RUNARGS=input.txt
Configuration Files: SDF.txt, TSD.txt must be in the same directory as the executable to run successfully
Versioning: Python Version 3.5.1
...
build:
run: build
python Control.py
...
build: GlassBR/Control.class
...
GlassBR/Control.class:
GlassBR/Control.java ...
javac GlassBR/Control.java
run: build
java GlassBR.Control $(RUNARGS)
...
\file Calculations.py
\author Nikitha Krithnan and W. Spencer Smith
\brief Provides functions for calculating the
...
\brief Calculates risk of failure
# \param inParams structure holding the input va
# \param J stress distribution factor (Function)
# \return risk of failure
def func_B(inParams, J):
outfile = open("log.txt", "a")
print("function func_B called with inputs: {
...
outfile.close()
return 2.86e-53 / (inParams.a * inParams.b)
inParams.h ** 2.0) ** 7.0 * inParams.LDF * math.
package GlassBR;
/** \file Calculations.java
\author Nikitha Krithnan and W. Spencer Smith
\brief Provides functions for calculating the outputs
*/
...
public static double func_B(InputParameters inParams, double J) throws IOException {
PrintWriter outfile;
outfile = new PrintWriter(new FileWriter(new File("log.txt"), true));
outfile.println("function func_B called with inputs: {");
...
outfile.close();
return 2.86e-53 /Math.pow(inParams.a * inParams.b, 7.0 - 1.0) *
Math.pow(7.17e10 * Math.pow(inParams.h, 2.0), 7.0) * inParams.LDF
* Math.exp(J);
}

Jacques Carette, W. Spencer Smith, Jason Balaci

Process

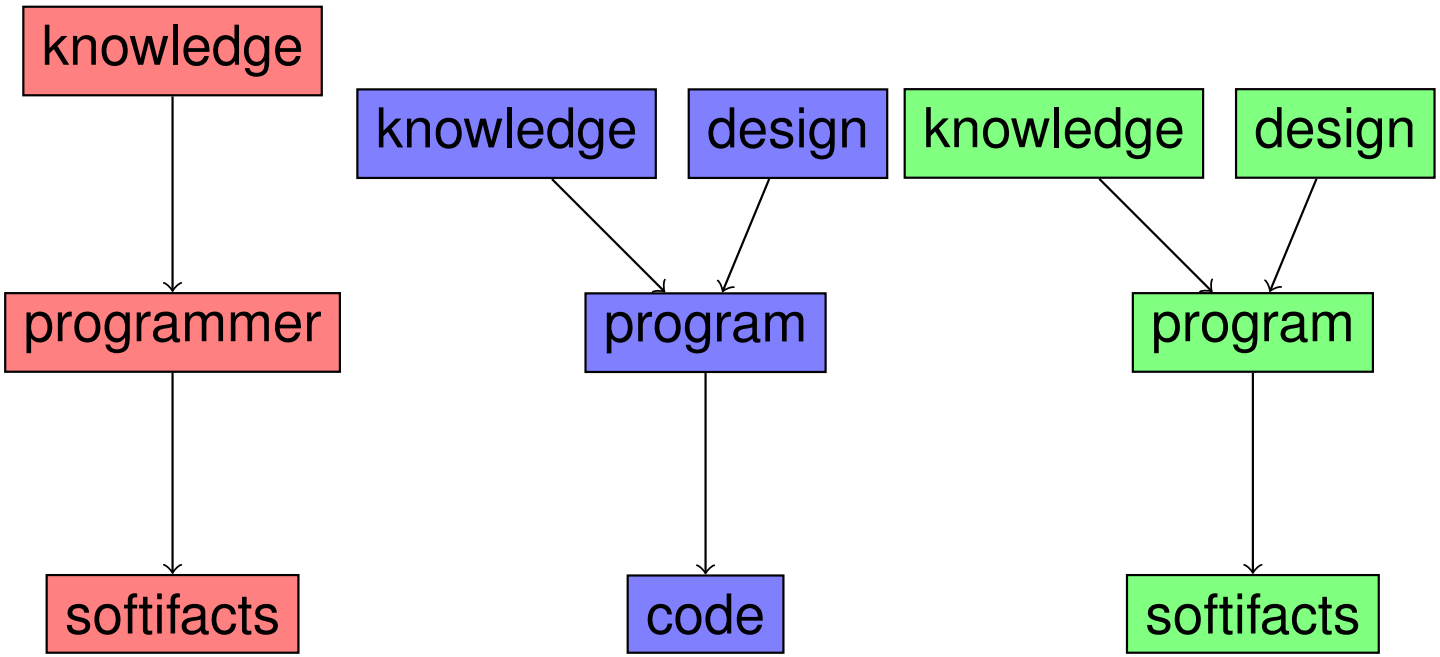


Process

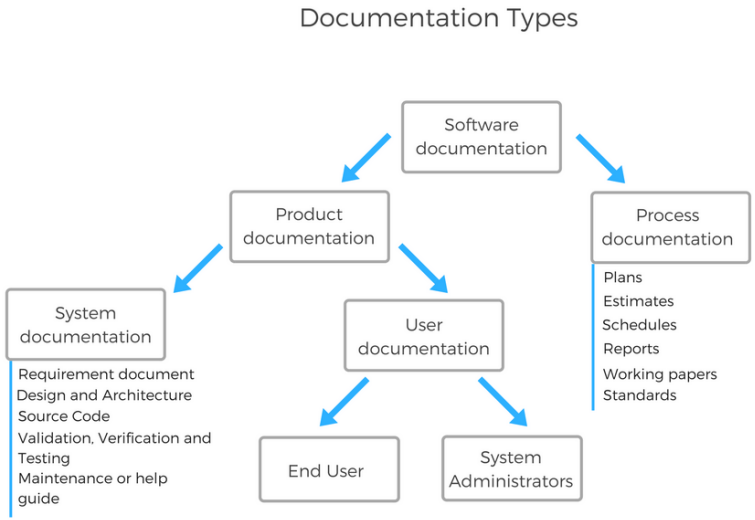

knowledge
knowledge
design
programmer
program
softifacts
code

Jacques Carette, W. Spencer Smith, Jason Balaci

Process



Key Philosophical Differences

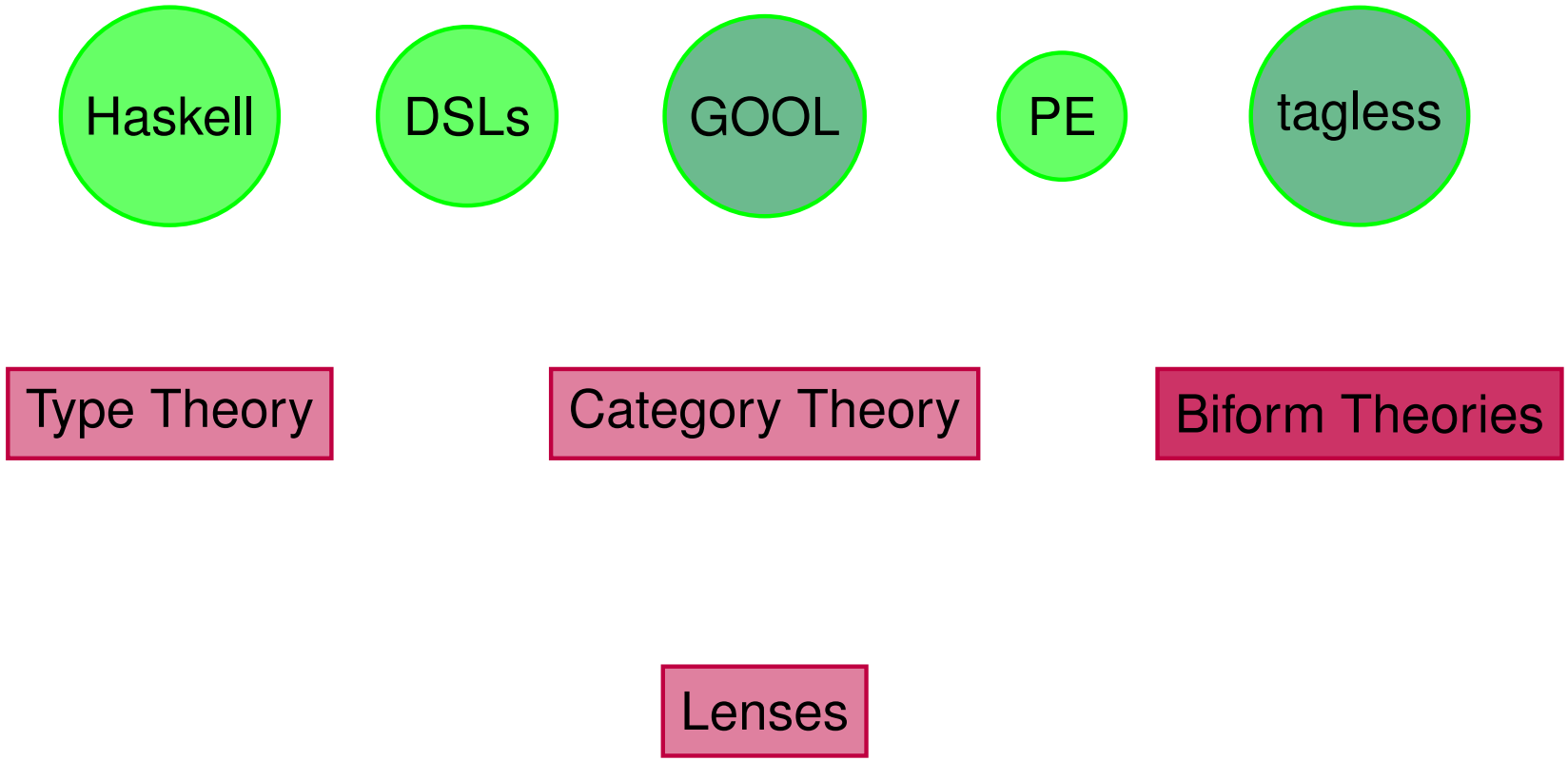


```
## \file Projectile.py
# \author Samuel J. Crawford, Brooks MacLachlan, and W. Spencer Smith
# \brief Contains the entire Projectile program
import math
import sys

## \brief Calculates flight duration: the time when the projectile lands (s)
# \param v_launch launch speed: the initial speed of the projectile when launched (m/s)
# \param theta launch angle: the angle between the launcher and a straight line from the launcher to
# \param g_vect gravitational acceleration (m/s^2)
# \return flight duration: the time when the projectile lands (s)
def func_t_flight(v_launch, theta, g_vect):
    return 2.0 * v_launch * math.sin(theta) / g_vect
```

Jacques Carette,
W. Spencer Smith,
Jason Balaci

Better Technology & Theory



What if we generated it all*?

Drasil

* when it makes sense do to so