Sustainable Software via Generation

Spencer Smith, Jacques Carette
Computing and Software Department, McMaster University, Canada
Email: smiths@mcmaster.ca

Keywords—software documentation, generative programming, sustainable software

Sustainable software projects satisfy, for a reasonable amount of developer effort, the requirements for the present, while also being maintainable, reusable and reproducible to address future needs. Achieving sustainability requires documentation (requirements specification, design documents, test plans, test reports, etc). Developers recognize the value of documentation, but often do not feel that they have the time/resources needed to write and maintain it in the face of inevitable software changes [1]. A solution to this problem is to provide a process that emphasizes documentation from the start, along with tools that facilitate change.

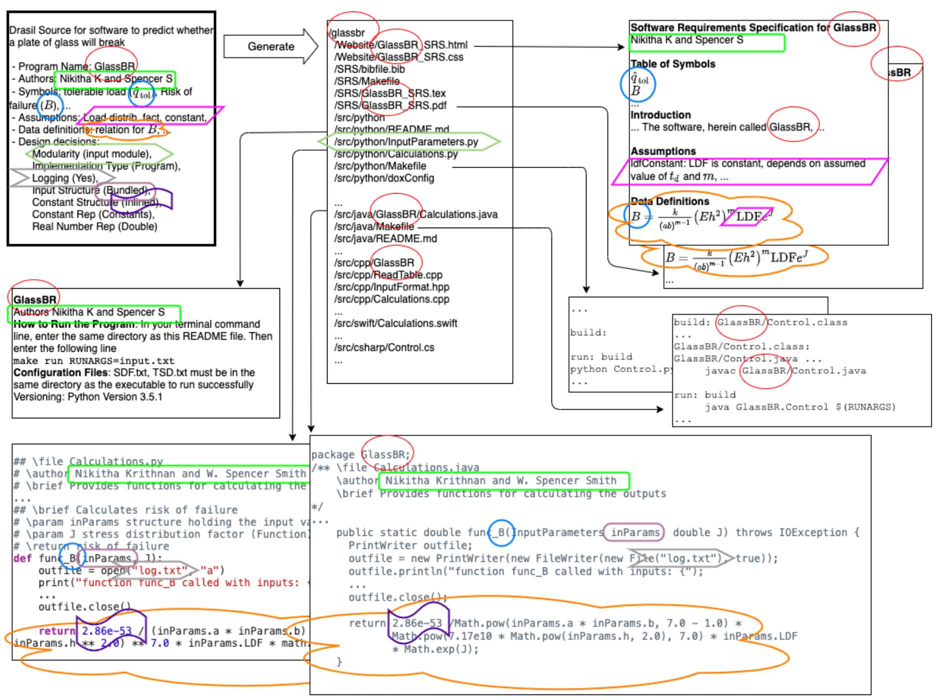
Drasil provides such a process and tools. Drasil is a framework for generating all of the software artifacts (code, documentation, scripts, etc.). Drasil takes advantage of the inherent knowledge duplication by capturing the knowledge once and providing means to transform that information into all of the views needed within a given project.

Figure 1 shows an example of the transformation of captured knowledge (as shown in the darker bordered box at the top left). The software predicts whether a given plane of glass is likely to break when subjected to an explosion. An important property of the software is its name. In this case the name is GlassBR. In the generated artifacts the name GlassBR appears more than 80 times – in the folder structure, in the requirements specification, in the README file, in the Makefile, and in the source code. Although the software name is a likely change, in a conventional software project changing the name across all artifacts is surprisingly difficult. With Drasil a change like this is made by one change to the source knowledge and regenerating. By capturing domain knowledge, Drasil facilitates more than just renaming. For instance, if the assumption of a constant Load Distribution Factor (LDF) changes, the regenerated software will have LDF as an input variable. Drasil also captures design decisions, like whether to log all calculations, whether to in-line constants rather than show them symbolically, etc. Drasil becomes even more powerful with the revelation that the same knowledge can be reused in different projects.

Developers can now generate sustainable software, with documentation and code that are consistent by construction.

References

[1] W. S. Smith, T. Jegatheesan, and D. F. Kelly, "Advantages, disadvantages and misunderstandings about document driven design for scientific software," in *Proceedings of the SE-HPCCE*, November 2016, 8 pp.



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 ($\hat{q}_{\text{tol}}$), 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)

```
/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
$\hat{q}_{\text{tol}}$
B
...

Introduction
... The software, herein called GlassBR, ...

Assumptions
ldfConstant: LDF is constant, depends on assumed value of t_d and m, ...

Data Definitions
$B = \frac{k}{(ab)^{m-1}}(Eh^2)^m \text{LDF} e^J$
...

sBR

$B = \frac{k}{(ab)^{m-1}}(Eh^2)^m \text{LDF} e^J$
...

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);
    }
```

Fig. 1. Mapping between changes in Drasil source code to the generated artifacts. The different colours and shapes show the connection between the source knowledge and the generated artifacts.