First Committee Meeting

Progress Report

Samuel Crawford

McMaster University

June 19, 2023

Table of Contents

Table of Contents

About Me

- I am **Samuel "Sam" Crawford**
- Graduated from McMaster University (2022)
  - Bachelor of Engineering (B.Eng.) in Software Engineering
  - Worked on Drasil as an Undergraduate Summer Research Assistant (during the summers of 2018 and 2019)
- Currently pursuing a Master of Applied Science (M.A.Sc.) in Software Engineering under the supervision of **Dr. Jacques Carette** and **Dr. Spencer Smith**

Overview of Progression Towards M.A.Sc.

Course-related progression

- I'm required to complete:
  - Two "Software" courses ✓ ✓
  - One "Theory" course ✓
  - One "Systems" course ✓
- I've completed:
  - CAS 735: (Micro)service-oriented architectures - Fall 2022
  - CAS 761: Logic for Practical Use - Fall 2022
  - CAS 741: Development of Scientific Computing Software - Winter 2023
  - CAS 781: Advanced Topics in Computing and Software (High-Performance Scientific Computing) - Winter 2023

Overview of Progression Towards M.A.Sc.

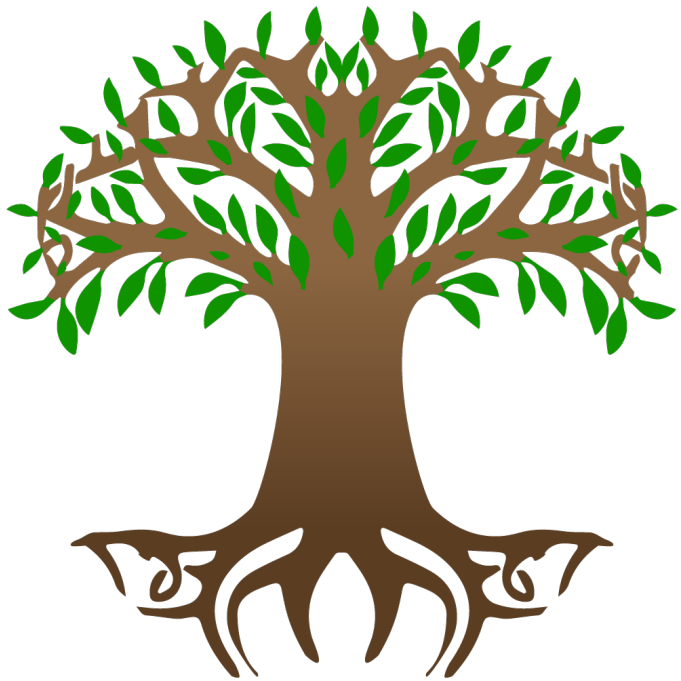
Thesis/research-related Progression

- Conducted "part-time research" while taking courses (Fall 2022/Winter 2023)
- Pivoted to "full-time research" for Spring 2023 (and beyond)
- Formed my supervisory committee; we are currently having our first supervisory committee meeting!

Table of Contents

Drasil is "a framework for generating all of the software artifacts from a stable knowledge base, focusing currently on scientific software" [Hunt et al., 2021]

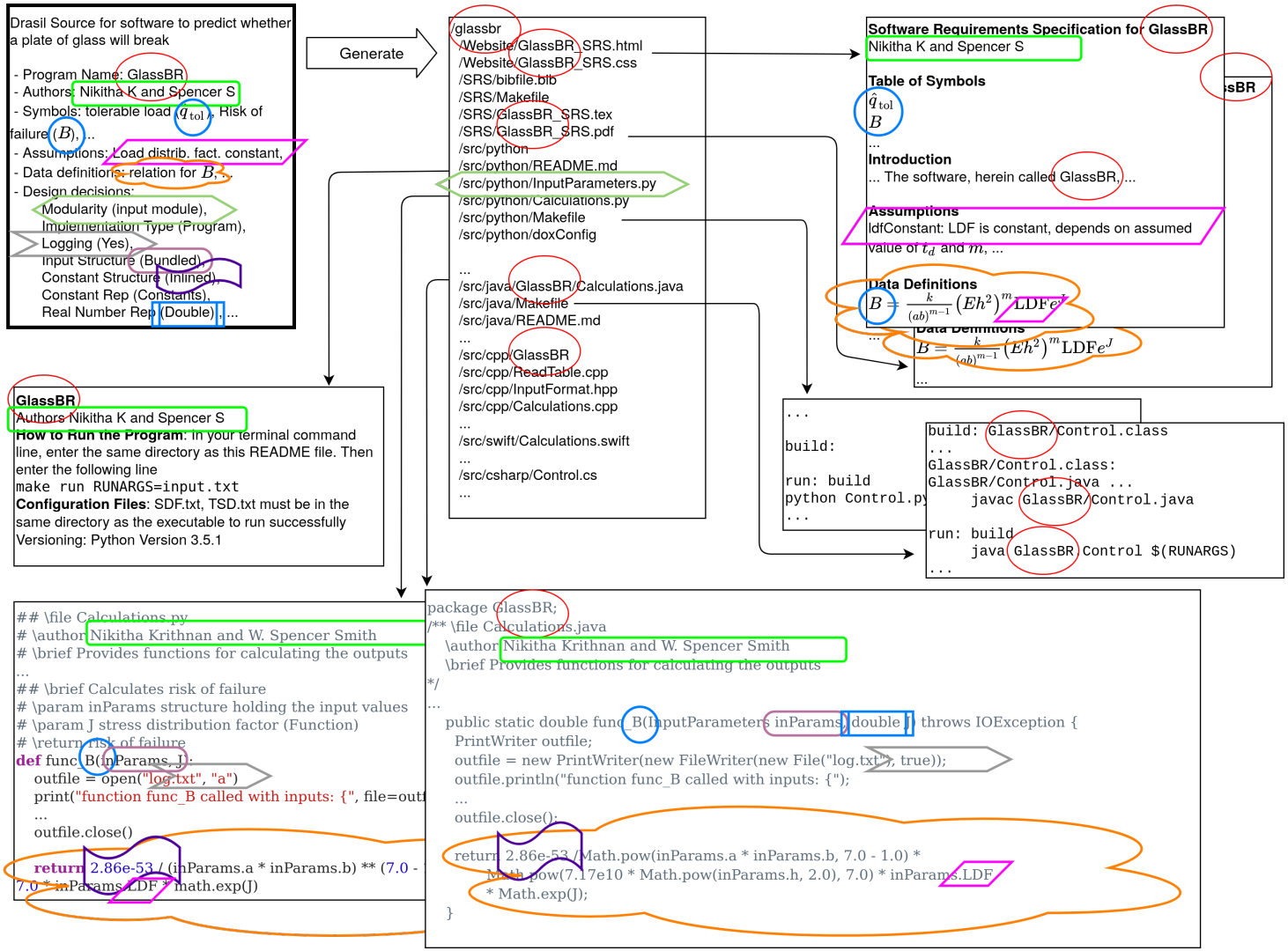
- This knowledge, using recipes, is used to generate software artifacts, including:
  - SRS (HTML, PDF, Jupyter)
  - Code (Python, Java, C#, C++, Swift)
  - READMEs
  - Makefiles
  - Its own website[1]!



Drasil's Logo [Carette et al., 2021]

[1] https://jacquescarette.github.io/Drasil/

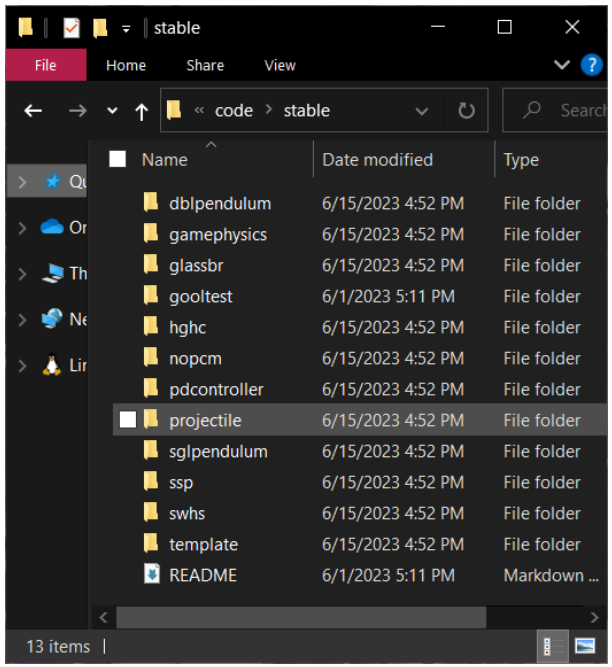
Visualizing Drasil's Traceability



Knowledge flow from knowledge base to artifacts; by Dr. Spencer Smith

Problem Statement

- Currently, there is no way to verify Drasil's output
- Drasil is "tested" by comparing generated artifacts to `stable`



Contents of `stable`

```
diff --strip-trailing-cr -r -X ../.gitignore
      stable/projectile/SRS/HTML/Projectile_SRS.html
build/projectile/SRS/HTML/Projectile_SRS.html
5c5,7
< <title>Software Requirements Specification for
      Projectile</title>
---
> <title>
> Software Requirements Specification for Projectile
      Motion
> </title>
24c26
< <h1>Software Requirements Specification for
      Projectile</h1>
---
> <h1>Software Requirements Specification for
      Projectile Motion</h1>
...
```

An example log

- This does not actually say anything about Drasil's output!

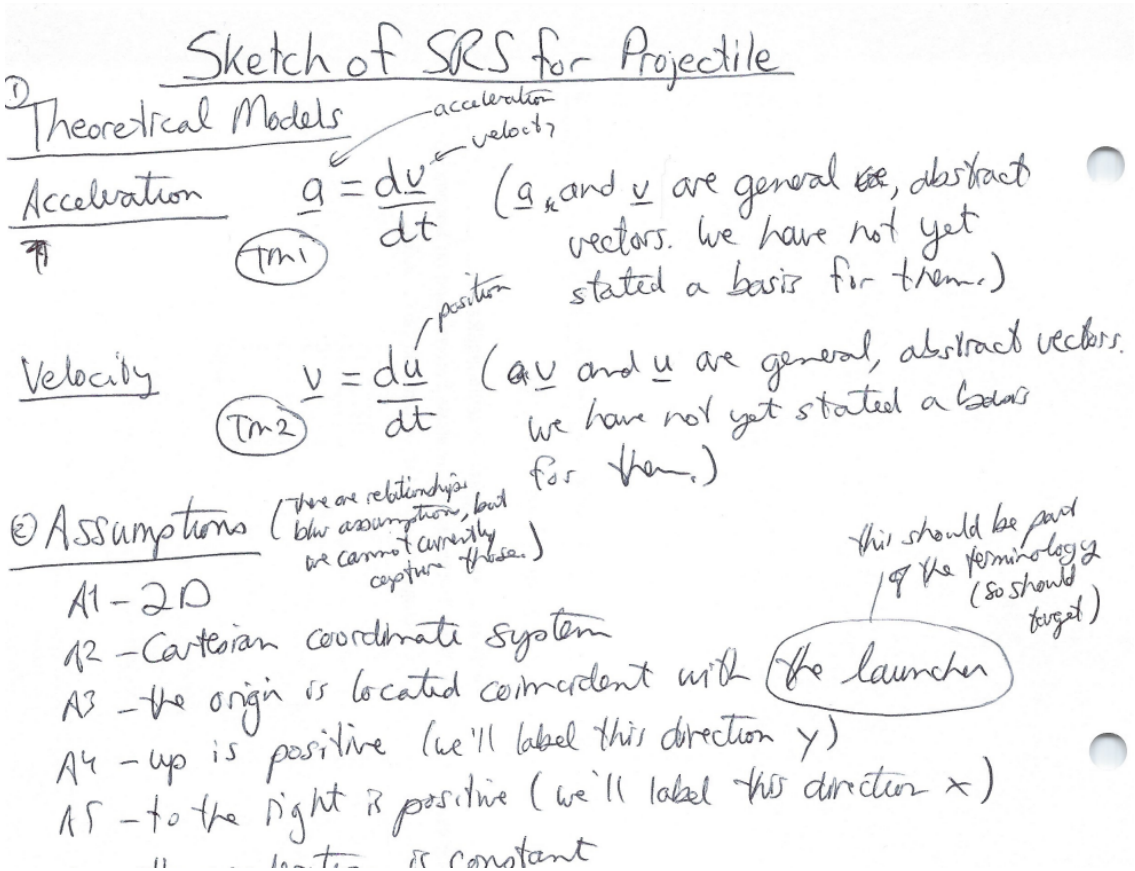
Purpose Statement

- The purpose of this research is to implement test case generation to verify generated code
- These test cases will be generated from information within Drasil
- Why use test cases for verification as opposed to, say, consistency/correctness checks?
  1. A more well-defined, Master's level scope
  2. Targets a more complex artifact that is harder to verify
  3. Gives Drasil another "bragging point"!

The Common Drasil Workflow

Example: Projectile

1. Create a manual version of an artifact



Sketch of Projectile SRS [Smith, 2019]

The Common Drasil Workflow

Example: Projectile

1. Create a manual version of an artifact
2. Understand it (and its components) well

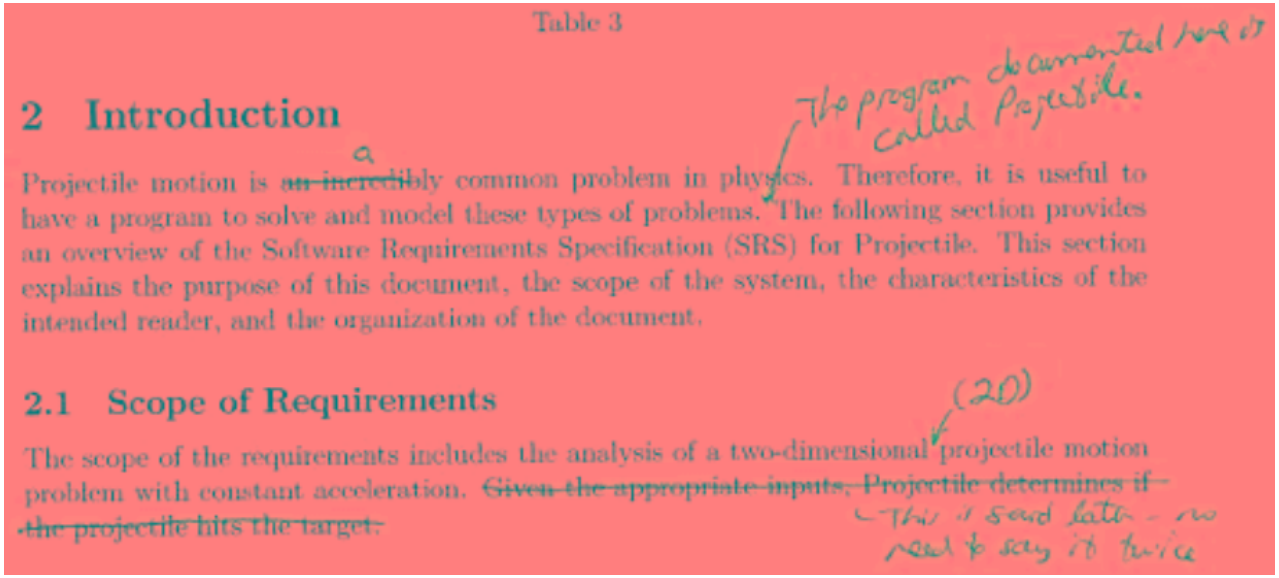


Table 3

2 Introduction

Projectile motion is ~~an incredibly~~ a common problem in physics. Therefore, it is useful to have a program to solve and model these types of problems. The following section provides an overview of the Software Requirements Specification (SRS) for Projectile. This section explains the purpose of this document, the scope of the system, the characteristics of the intended reader, and the organization of the document.

The program documented here is called Projectile.

2.1 Scope of Requirements

The scope of the requirements includes the analysis of a two-dimensional projectile motion problem with constant acceleration. ~~Given the appropriate inputs, Projectile determines if the projectile hits the target.~~

(20)

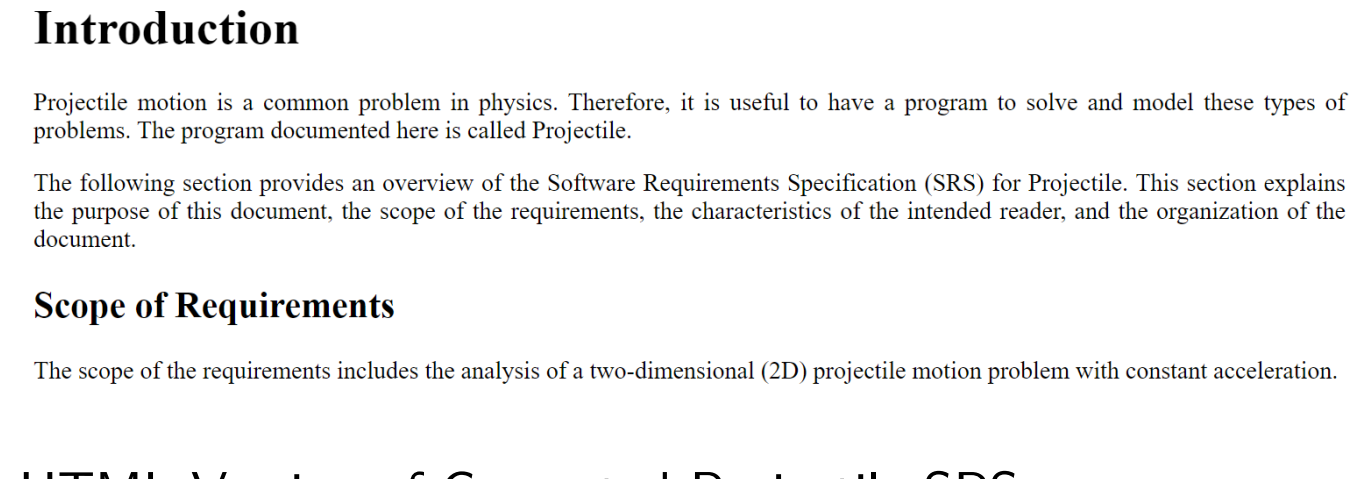
This is said later - no need to say it twice

Review of Manual Projectile SRS [Smith and Crawford, 2019]

The Common Drasil Workflow

Example: Projectile

1. Create a manual version of an artifact
2. Understand it (and its components) well
3. Generate it!



Introduction

Projectile motion is a common problem in physics. Therefore, it is useful to have a program to solve and model these types of problems. The program documented here is called Projectile.

The following section provides an overview of the Software Requirements Specification (SRS) for Projectile. This section explains the purpose of this document, the scope of the requirements, the characteristics of the intended reader, and the organization of the document.

Scope of Requirements

The scope of the requirements includes the analysis of a two-dimensional (2D) projectile motion problem with constant acceleration.

HTML Version of Generated Projectile SRS [Crawford et al., 2023]

The Common Drasil Workflow

Applied to Testing

1. Create a manual version of an artifact
   - Manual unit tests (26 pass, 18 fail with known reason)

```
## \brief Tests reading valid input
@mark.parametrize("filename,v_launch,theta,p_target",
                  get_expected("v_launch", "theta", "p_target"))
def test_get_input_valid(filename, v_launch, theta, p_target):
    assert isclose(conftest.inParams[filename].v_launch, v_launch)
    assert isclose(conftest.inParams[filename].theta, theta)
    assert isclose(conftest.inParams[filename].p_target, p_target)
```

Sample from `InputParameters_test.py`

The Common Drasil Workflow

Applied to Testing

1. Create a manual version of an artifact
   - Manual unit tests (26 pass, 18 fail with known reason)

```
def build_mocks(*attrs):
    mocks = []
    defaults = ["v_launch", "theta", "p_target"]
    for d in get_expected(*(defaults + list(attrs))):
        mock_attrs = dict()
        for i, attr in enumerate(defaults + list(attrs), start=1):
            mock_attrs[attr] = d[i]
        mock = Mock()
        mock.configure_mock(**mock_attrs)
        mocks.append(mock)
    return mocks

## \brief Tests calculation of t_flight with valid input
@mark.parametrize("mock", build_mocks("t_flight"))
def test_func_t_flight_valid(mock):
    assert isclose(Calculations.func_t_flight(mock, valid_g), mock.t_flight)
```

Sample from `Calculations_test.py`

The Common Drasil Workflow

Applied to Testing

1. Create a manual version of an artifact
   - Manual unit tests (26 pass, 18 fail with known reason)

```
## \brief Tests writing valid input
@mark.parametrize("s, d_offset, t_flight", get_expected("d_offset", "t_flight"))
def test_get_input_valid(s, d_offset, t_flight):
    OutputFormat.write_output(s, d_offset, t_flight)
    with open("output.txt") as f:
        assert f.readlines() == [f"s = {s}\n",
                                 f"d_offset = {d_offset}\n",
                                 f"t_flight = {t_flight}\n",]
```

Sample from `OutputFormat_test.py`

The Common Drasil Workflow

Applied to Testing

1. Create a manual version of an artifact
   - Manual unit tests (26 pass, 18 fail with known reason)
   - Manual system tests (3 pass, 4 fail with known reason)

```
## \brief Tests main with valid input file
@mark.parametrize("filename", get_expected())
def test_main_valid(monkeypatch, filename):
    with monkeypatch.context() as m:
        m.setattr(sys, 'argv', ['Control.py', str(Path("test/test_input") /
            f"{filename}.txt")])
        Control.main()
    assert read_file(output_filename) == read_file(str(Path("test/test_output") /
        f"{filename}.txt"))
```

Sample from `Control_test.py`

2. Understand the manual artifact (and its components) well
   - Changes made to "stable" to faciliate testing
     - The inclusion of `__init__.py` files to improve `import` statements
     - Wrapping `Control.py`'s functionality in a `main` function
     - Changing how command line parameters are passed to `Control.py`
   - Changes to be made to generated code to improve correctness
     - Invalid values should stop the calculations [Crawford et al., 2023]
     - Assumptions, such as values of constants, should be verified

Why Test Generated Code?

If the code is being generated from a stable knowledge base, then it should be correct. Why waste effort testing it?

1. The knowledge base is not actually "stable" yet
2. There are plenty of places for a mistake to be introduced
3. Testing provides a greater degree of confidence in Drasil's capabilities
4. Generating code for testing allows for it to be done "properly" instead of taking shortcuts commonly taken by humans

Next Steps

2. Understand the manual artifact (and its components) well
   - Understanding the problem domain lets one develop a solution that:
     - Makes use of all areas of the domain
     - Follows domain standards, including quality and terminology
   - There are specific areas of testing that need to be understood:
     - **Research Question #1:** What information is necessary for different types of testing?
     - **Research Question #2:** How can test cases be generated from information that currently exists within Drasil?
     - **Research Question #3:** How can new information be added to facilitate the generation of more types of testing?

"The information you have should be just as useful for generating tests as it should be for manually running them." — Dr. Jacques Carette

Next Steps

3. Generate it!
   - Test cases will then be written for:
     - Other variabilities of Projectile's Python implementation
     - Projectile's implementation in other languages
     - Other examples where code is generated: GlassBR, NoPCM, DblPendulum, PD Controller [Hunt et al., 2021]
   - These test cases will also be added to Drasil's CI/CD to ensure that future changes preserve the code's functionality

Acknowledgements

- Dr. Smith and Dr. Carette have been great supervisors in the past and have, both then and now, provided me with valuable guidance and feedback
  - They have helped me refine the scope of this project
  - The project itself was originally posed by Dr. Smith back in 2020!
- The format of this presentation was *heavily* based on a previous presentation by Jason Balaci
- Dr. Smith created the knowledge flow figure shown earlier
- The past and current Drasil team have created a truly amazing framework!

Thank you!

Questions?

Table of Contents

References


Carette, J., Smith, S., Balaci, J., Hunt, A., Wu, T.-Y., Crawford, S., Chen, D., Szymczak, D., MacLachlan, B., Scime, D., and Niazi, M. (2021).
Drasil.


Crawford, S., MacLachlan, B., and Smith, S. (2023).
Feedback on Projectile.
https://jacquescarette.github.io/Drasil/examples/projectile/SRS/srs/Projectile_SRS.html.


Hunt, A., Michalski, P., Chen, D., Balaci, J., and Smith, S. (2021).
Drasil - Generate All the Things!


Smith, S. (2019).
Sketch of SRS for Projectile.


Smith, S. and Crawford, S. (2019).
Feedback on Projectile.