Putting Software Testing Terminology to the Test

MASc Seminar

Samuel Crawford, B.Eng.

McMaster University
Department of Computing and Software

Fall 2024

Table of Contents

1. Introduction
2. Project
   - Drasil
   - The Common Drasil Workflow
   - Why Test Generated Code?
   - Next Steps
3. References

Table of Contents

1. Introduction

2. Project
   - Drasil
   - The Common Drasil Workflow
   - Why Test Generated Code?
   - Next Steps

3. References

About Me

- I am **Samuel "Sam" Crawford**

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)

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

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:

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

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

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

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)

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)

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

1. Introduction

2. Project
   - Drasil
   - The Common Drasil Workflow
   - Why Test Generated Code?
   - Next Steps

3. References

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]

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

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]!

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

Visualizing Drasil's Traceability

Problem Statement

- Currently, there is no way to verify Drasil's output

Problem Statement

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

Problem Statement

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

- 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

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?

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

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

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

The Common Drasil Workflow

Example: Projectile

1. Create a manual version of an artifact

The Common Drasil Workflow

Example: Projectile

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

The Common Drasil Workflow

Example: Projectile

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

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)

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)

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)

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)

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)

2. Understand the manual artifact (and its components) well

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`

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 [**?**]
     - 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?

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

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

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

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

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

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:

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?

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?

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?

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!

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]

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

Acknowledgment

- 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!

Acknowledgment

- 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, who also provided a great thesis template

Acknowledgment

- 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, who also provided a great thesis template
- The past and current Drasil team have created a truly amazing framework!

Thank you!

Questions?

Table of Contents

1. Introduction
2. Project
   - Drasil
   - The Common Drasil Workflow
   - Why Test Generated Code?
   - Next Steps
3. References

References


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