Putting Software Testing Terminology to the Test

M.A.Sc. Seminar

Samuel Crawford, B.Eng.

McMaster University
Department of Computing and Software

Fall 2024

Table of Contents

Table of Contents

The Need for A Knowledge Base (cont.)

- Before we can generate test cases, we need to "teach" Drasil how to build them and what information is needed to do so
- If knowledge about testing is to be "well understood", it needs to be documented clearly, consistently, and correctly

The Need for A Knowledge Base (cont.)

- Before we can generate test cases, we need to "teach" Drasil how to build them and what information is needed to do so
- If knowledge about testing is to be "well understood", it needs to be documented clearly, consistently, and correctly
- Independently of Drasil, if the field of software engineering holds code to a high standard in terms of clarity, consistency, and robustness, then the literature that supports code development should be held to this same standard!

The Lack of a Knowledge Base

"The Problem"

- Unfortunately, a search for a systematic, rigorous, and complete taxonomy for software testing revealed that the existing ones are inadequate and incomplete:
  - Tebes et al. (2020) focus on *parts* of the testing process (e.g., test goal, testable entity),
  - Souza et al. (2017) prioritize organizing testing approaches over defining them, and
  - Unterkalmsteiner et al. (2014) focus on the "information linkage or transfer" (p. A:6) between requirements engineering and software testing.

Table of Contents

The Problem with Testing Literature

Unstandardized Standards

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"!

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

References

Anthony Hunt, Peter Michalski, Dong Chen, Jason Balaci, and Spencer Smith. Drasil - Generate All the Things!, 2021. URL `https://jacquescarette.github.io/Drasil/`.

Erica Souza, Ricardo Falbo, and Nandamudi Vijaykumar. ROoST: Reference Ontology on Software Testing. *Applied Ontology*, 12:1–32, March 2017. doi: 10.3233/AO-170177.

Guido Tebes, Luis Olsina, Denis Peppino, and Pablo Becker. TestTDO: A Top-Domain Software Testing Ontology. pages 364–377, Curitiba, Brazil, May 2020. ISBN 978-1-71381-853-3.

Michael Unterkalmsteiner, Robert Feldt, and Tony Gorschek. A Taxonomy for Requirements Engineering and Software Test Alignment. *ACM Transactions on Software Engineering and Methodology*, 23(2):1–38, March 2014. ISSN 1049-331X, 1557-7392. doi: 10.1145/2523088. URL `http://arxiv.org/abs/2307.12477`. arXiv:2307.12477 [cs].