A New Taxonomy of Software Testing Approaches

Seeking More Standardized Standards

Samuel Joseph Crawford
crawfs1@mcmaster.ca
Department of Computing and Software, McMaster University

April 22, 2023

Background

The first step to any formal process is understanding the underlying domain well. Therefore, a systematic and rigorous understanding of software testing approaches is needed to develop formal tools to test software. In my specific case, my motivation was seeing which kinds of testing could be generated automatically by Drasil, "a framework for generating all of the software artifacts for (well understood) research software" (Carette et al., 2021).

Most software testing ontologies seem to focus on the high-level testing process rather than the testing techniques themselves. For example, the terms and definitions (Tebes et al., 2020b) from TestTDO (Tebes et al., 2020a) mainly focus on parts of the testing process (e.g., test goal, test plan, testing role, testable entity) and how they relate to one another. Unterkalmsteiner et al. (2014) provide a foundation to allow one "to classify and characterize alignment research and solutions that focus on the boundary between [requirements engineering and software testing]" but do not "do[] not aim at providing a systematic and exhaustive state-of-the-art survey of [either domain]" (p. A:2).

Background: The Elm Architecture

- Elm is a pure, strictly-typed functional programming language
- Every Elm program follows a rigid structure known as *The Elm Architecture* (TEA)

Theory: Extending The Elm Architecture to Multiple Clients

- Theorem 1: Two clients running Elm programs having an identical initial model, identical update functions, and processing an identical sequence of messages will end up with an identical model.
- Proof: This can be shown easily by using Elm's pureness property.
- Corollary: Any number of Elm clients having an identical initial model, identical update functions, and processing an identical sequence of messages will end up with an identical model.
- Observation: By ensuring every client receives the same messages in the same order, we effectively have a multi-user application, without writing any application-specific server code.

TEASync Framework Architectural Overview

Example Applications

The following is a simple application which allows multiple users to increment and decrement an integer. Figure **??** shows an example UI for this application. Figure **??** shows an example Pong game made using this framework.

```
type GlobalMsg = Increment | Decrement

type alias GlobalModel = { count : Int }

globalUpdate : GlobalMsg -> GlobalModel -> GlobalModel
globalUpdate msg globalModel =
case msg of
  Increment -> { globalModel | count = globalModel.count + 1 }
  Decrement -> { globalModel | count = globalModel.count - 1 }
```

Conclusions & Future Work

Leveraging the strictly-typed nature of Elm and its model-view-update architecture, we were able to create a simplified multi-user framework, requiring the programmer to write no server-side code. In addition to the upcoming pedagogical study, future work includes a data modelling extension allowing persistent, structured data, an authentication/authorization scheme, a binary data format to reduce network communication, and curriculum development for a TEASync-based summer camp.

References

Jacques Carette, Spencer Smith, Jason Balaci, Ting-Yu Wu, Samuel Crawford, Dong Chen, Dan Szymczak, Brooks MacLachlan, Dan Scime, and Maryyam Niazi. Drasil, February 2021. URL https://github.com/JacquesCarette/Drasil/tree/v0.1-alpha.

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

Guido Tebes, Luis Olsina, Denis Peppino, and Pablo Becker. TestTDO_terms_definitions_vfinal.pdf, February 2020b. URL https://drive.google.com/file/d/19TWHd50HF04K6PPyVixQzR6c7HjW2kED/view.

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].

Acknowledgments

We thank NSERC for CGS-M funding and the Government of Ontario for OGS funding.