

A New Taxonomy of Software Testing Approaches

Seeking More Standardized Standards

Samuel Joseph Crawford, Jacques Carette, & Spencer Smith {crawfs1, carette, smiths}@mcmaster.ca

Department of Computing and Software, McMaster University

April 22, 2023



Goal

We want a systematic, rigorous, and "complete" taxonomy of software testing approaches.

- This will help us automatically generate test cases in our research framework Drasil [1]
- We need to understand the underlying domain to determine which kinds of testing can be generated and how to do so (e.g., what knowledge is required?)

Problem

Existing software testing taxonomies are inadequate; for example:

- Tebes et al. (2020) mainly focus on parts of the testing process (e.g., test goal, testable entity)
- \blacksquare ROoST, by Souza et al. (2017), is an ontology, and as such, prioritizes organizing testing approaches over defining them

Methodology

Since a taxonomy doesn't already exist, we should create one!

- Relevant information (currently 190 testing approaches, 85 software qualities, and their
- We will iterate this process until we encounter diminishing returns, implying that something approaching a **complete taxonomy** has emerged!
- Since there are many standardized documents about software testing (or software in general), this should be trivial, no?

Levels of testing

Integration testing System testing System integration testing Acceptance testing

Unit testing

- User acceptance testing Operational acceptance
- Factory acceptance testing
- Alpha testing Beta testing
- Production verification
- testing

Test practices

Model-based testing Scripted testing **Exploratory testing** Experience-based testing Manual testing A/B testing Back-to-back testing Mathematical-based testing Fuzz testing Keyword-driven testing Automated testing Capture-replay driven Data-driven

Functional testing Accessibility testing Compatibility testing Conversion testing Disaster/recovery testing Installability testing Interoperability testing Localization testing Maintainability testing

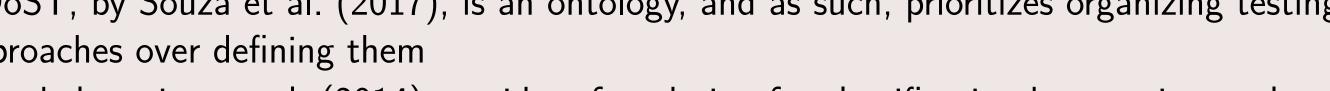
— Load

Types of testing

- Performance-related
- testing Performance
- Stress — Capacity
- Recovery Portability testing Procedure testing Reliability testing Security testing Usability testing

- Branch testing
- Branch condition
- combination testing
- MC/DC testing Data flow testing
- All-definitions testing
- All-C-uses testing — All-P-uses testing
- All-uses testing All-DU-paths testing

Experience-based: Error guessing



■ Unterkalmsteiner et al. (2014) provide a foundation for classification but not its results

- Started from **established standards and resources**, such as IEEE [2, 3, 4] and SWEBOK [5]
- definitions) is then collected and organized into spreadsheets

In Our Experience...

Test design techniques/ measures

Specification-based: Equivalence partitioning Classification tree method

- Boundary value analysis Syntax testing Combinatorial testing — All combinations
- Pairwise Each choice
- Base choice Decision table testing
- Cause-effect graphing State transition testing
- Scenario testing Use case testing Random testing Metamorphic testing
- Requirements-based testing
- Structure-based: Statement testing
- Decision testing Branch condition testing
 - - testing and combination

Figure 1: Classification of some "test approach choices" [2, p. 22].

Static testing

Reviews (ISO/IEC 20246)

Static analysis

Model verification

Information often appears logical, but this often breaks down. For example, the classification of test approaches in Figure 1 reveals the following ambiguities:

- Experience-based testing is both a test design technique and a test practice
- What distinguishes the following pairs is unclear:
 - Disaster/recovery testing and recovery testing
 - Branch condition branch condition testing

More Examples

Despite [2] being a standard for general concepts related to software testing, it leaves much unstandardized. As shown in Figure 2, most (55 out of 99) testing approaches it mentions do not have a definition! Eight of these were (at the very least) described in the previous version of this standard [4], and nine were present in the same way in another IEEE standard [3] that would have been available upon publication of this one. However, the presence of a definition does not guarantee that it is useful! See Figure 3 for some good (bad?) examples.

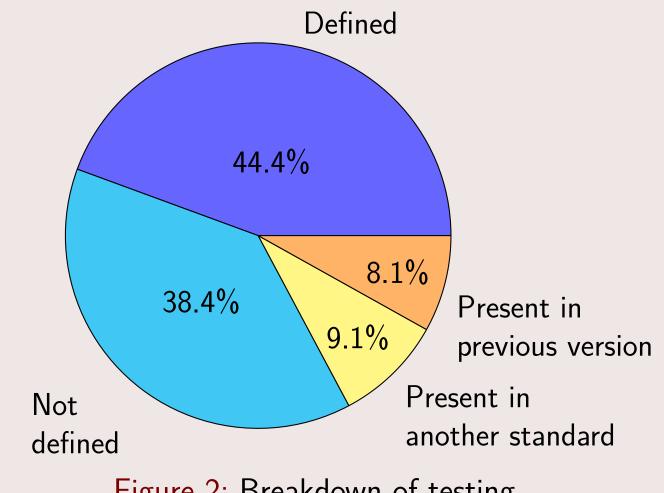


Figure 2: Breakdown of testing approach definitions in [2].

software element 1. system element that is software

cf. system element, software/system element

1. state of

event sequence analysis operable **1.** per

1. mechanism or piece of equipment designed to serve a purpose or perform a function

Figure 3: Less-than-helpful definitions [3, pp. 421, 170, 136, 301 (counterclockwise from top)]. Note: "equipment" is not defined, and "mechanism" is only defined as how "a function ...transform[s] input into output" [p. 270].

This problem extends to definitions of testing approaches. For example, SWEBOK V4 says "scalability testing evaluates the capability to use and learn the system and the user documentation." It also focuses on the system's effectiveness in supporting user tasks and the ability to recover from user errors" [5, p. 5-9]. This definition seems to be an amalgamation of the definitions of usability, recovery, and potentially functional testing. What's more, SWEBOK's definition of elasticity testing cites a single source [5, p. 5-9] that doesn't contain the words "elasticity" or "elastic"!

Even when the general idea behind an approach is understood, discrepancies can still arise. While alpha testing is quite common and understood, there is disagreement on who performs it:

- "users within the organization developing the software" [3, p. 17],
- "a small, selected group of potential users" [5, p. 5-8], or
- "roles outside the development organization" [6].

Conclusions & Future Work

- Current software testing taxonomies are incomplete, inconsistent, and/or incorrect
- For one to be useful, it needs to be built systematically from a large body of established sources
- We will continue investigating how the literature defines and categorizes software testing approaches to analyze any discrepancies and structure these ideas coherently
- Hopefully, this leads to a **centralized**, **consistent taxonomy** that can grow alongside the literature as the field of testing advances

References

- [1] J. Carette, S. Smith, J. Balaci, T.-Y. Wu, S. Crawford, D. Chen, D. Szymczak, B. MacLachlan, D. Scime, and M. Niazi, "Drasil," Feb. 2021.
- [2] ISO/IEC and IEEE, "ISO/IEC/IEEE International Standard Systems and software engineering –Software testing –Part 1: General concepts," ISO/IEC/IEEE 29119-1:2022(E), Jan. 2022.
- [3] ISO/IEC and IEEE, "ISO/IEC/IEEE International Standard Systems and software engineering-Vocabulary," ISO/IEC/IEEE 24765:2017(E), Sept. 2017.
- [4] ISO/IEC and IEEE, "ISO/IEC/IEEE International Standard Systems and software engineering –Software testing –Part 1: General concepts," ISO/IEC/IEEE 29119-1:2013, Sept. 2013.
- Jan. 2024.

[6] M. Hamburg and G. Mogyorodi, editors, "ISTQB Glossary, v4.3," 2024.

[5] H. Washizaki, ed., Guide to the Software Engineering Body of Knowledge, Version 4.0.

Acknowledgments

We thank the Government of Ontario for OGS funding.