Putting Software Testing Terminology to the Test

Samuel J. Crawford*, Spencer Smith*, Jacques Carette*
*Department of Computing and Software
McMaster University
Hamilton, Canada
{crawfs1, smiths, carette}@mcmaster.ca

Abstract—This document is a model and instructions for LaTeX. This and the IEEEtran.cls file define the components of your paper [title, text, heads, etc.]. *CRITICAL: Do Not Use Symbols, Special Characters, Footnotes, or Math in Paper Title or Abstract.

Index Terms—software testing, terminology, taxonomy, literature review, test approaches

I. Background

Testing software is complicated, expensive, and often expensive. Therefore, automating the software testing process is an area of interest. In particular, test case generation for Drasil, a framework for "generating all of the software artifacts for (well understood) research software" [1] was desired.

Before attempting to automate a knowledge-based process in ad hoc manner, it is important to understand the underlying domain. This allows the information itself to drive the scope, including both what is possible and what is needed. In this case, the goal was to uncover the various approaches towards testing, as well as which prerequisites (e.g., input files, oracles) existed for each. A search for a systematic, rigorous, and "complete" taxonomy revealed that existing software testing taxonomies are inadequate:

- [2] focuses on parts of the testing process (e.g., test goal, testable entity)
- [3] prioritizes organizing testing approaches over defining them
- [4] provides a foundation for classification but not how it applies to software testing terminology

The "solution" to this gap was to examine existing literature (II), which quickly reinforced the need for a taxonomy! Despite the amount of well understood and organized knowledge, there were still many discrepancies and ambiguities, either within the same source or between various sources (III).

II. Methodology

A. Source Order

0) Textbooks trusted at McMaster [5, 6, 7]
   - Ad hoc and arbitrary; not systematic
   - Colored maroon
1) Established standards (such as IEEE, ISO/IEC, and SWEBOK) [8, 9, 10, 11, 12, 13, 14, 15, 16, 17]
   - Standards organizations colored green
   - "Meta-level" commentaries or collections of terminology (often based on these standards), such as [18], colored blue
2) Other resources: less-formal classifications of terminology (such as [19]), sources investigated to "fill in" missing definitions (see Undefined Terms), and testing-related resources that emerged for unrelated reasons
   - Colored black, along with any "surface-level" analysis that followed trivially

By looking at a variety of sources, discrepancies both within and between them can be uncovered, and the various classifications of terminology can be analyzed.

B. Procedure

Except for some sources in Undefined Terms, all sources were looked through in their entirely to "extract" as much terminology as possible. Heuristics were used to guide this process, by investigating…

- …glossaries and lists of terms
- …testing-related terms
  e.g., terms that included "test(ing)", "validation", "verification", "review(s)", or "audit(s)"
- …terms that had emerged as part of already-discovered testing approaches, especially those that were ambiguous or prompted further discussion
  e.g., terms that included "performance", "recovery", "component", "bottom-up", "boundary", or "configuration"
- …terms that implied testing approaches[1] (see Derived Test Approaches)

If a term's definition had already been recorded, either the "new" one replaced it if the "old" one wasn't as clear/concise or parts of both were merged to paint a more complete picture. If any discrepancies or ambiguities arose, they were investigated to a reasonable extent and documented. If a testing approach was mentioned but not defined, it was still added to the glossary to indicate it should be investigated further (see Undefined Terms).

[1]Since these methods for deriving test approaches only arose as research progressed, some examples would have been missed during the first pass(es) of resources investigated earlier in the process. While reiterating over them would be ideal, this may not be possible due to time constraints.

Identify applicable funding agency here. If none, delete this.

A similar methodology was used for tracking software qualities, albeit in a separate document (see Derived Test Approaches).

During the first pass of data collection, all software-testing-focused terms were included. Some of them are less applicable to test case automation (such as static testing; see #39) or too broad (such as attacks; see Attacks, #55), so they will be omitted over the course of analysis.

C. Undefined Terms

This process also led to some testing approaches without definitions; [8] and [18] in particular introduced many. Once more "standard" sources had been exhausted, a strategy was proposed to look for sources that explicitly defined these terms, with the added benefit of uncovering more terms to explore, potentially in different domains (see #57). This also uncovered some out-of-scope testing approaches, including EMSEC testing, HTML testing, and aspects of loop testing and orthogonal array testing; since these are out of scope, relevant sources were not investigated fully.

The following terms (and their respective related terms) were explored, bringing the number of testing approaches from 432 to 515 and the number of undefined terms from 153 to 171 (the assumption can be made that about 78% of added terms also included a definition):

- Assertion Checking
- Loop Testing
- EMSEC Testing
- Asynchronous Testing
- Performance(-related) Testing
- Web Application Testing
  - HTML Testing
  - DOM Testing
- Sandwich Testing
- Orthogonal Array Testing
- Backup Testing

Different sources categorized software testing approaches in different ways; while it is useful to record and think about these categorizations, following one (or more) during the research stage could lead to bias and a prescriptive categorization, instead of letting one emerge descriptively during the analysis stage. Since these categorizations are not mutually exclusive, it also means that more than one could be useful (both in general and to this specific project); more careful thought should be given to which are "best", and this should happen during the analysis stage.

III. Observations

A. Categories of Testing Approaches

For classifying different kinds of tests, [8] provides some terminology (see Table I). Related testing approaches may be grouped into a "class" or "family" to group those with "commonalities and well-identified variabilities that can be instantiated", where "the commonalities are large and the variabilities smaller" (see #64). Examples of these are the classes of combinatorial [17, p. 15] and data flow testing [p. 3] and the family of performance-related testing [20, p. 1187][2], and may also be implied for security testing, a test type that consists of "a number of techniques[3]" [17, p. 40].

It also seems that these categories are orthogonal. For example, "a test type can be performed at a single test level or across several test levels" [8, p. 15], [17, p. 7]. Due to this, a specific test approach can be derived by combining test approaches from different categories; for some examples of this. However, the boundaries between items within a category may be unclear: "although each technique is defined independently of all others, in practice [sic] some can be used in combination with other techniques" [17, p. 8]. For example, "the test coverage items derived by applying equivalence partitioning can be used to identify the input parameters of test cases derived for scenario testing" [p. 8]. Even the categories themselves are not consistently defined, as some approaches are categorized differently by different sources; these differences will be tracked noted so that they can be analyzed more systematically (see #21). There are also several instances of inconsistencies between parent and child test approach categorizations (which may indicate they aren't necessarily the same, or that more thought must be given to classification/organization). Examples of discrepancies in test-approach categorization:

1) Experience-based testing is categorized as both a test design technique and a test practice on the same page [8, pp. 22, 34]!
   - These authors previously say "experience-based testing practices like exploratory testing … are not … techniques for designing test cases", although they "can use … test techniques" [17, p. viii]. This implies that "experience-based test design techniques" are techniques used by the practice of experience-based testing, not that experience-based testing is itself a test technique. If this is the case, it is not always clearly articulated [8, pp. 4, 22], [17, p. 4], [9, p. 5-13], [16] and is sometimes contradicted [18, p. 46]. However, this conflates the distinction between "practice" and "technique", making these terms less useful, so this may just be a mistake (see #64).
   - This also causes confusion about its children, such as error guessing and exploratory testing; again, on the same page, [8, p. 34] says error

[2]The original source describes "performance testing … as a family of performance-related testing techniques", but it makes more sense to consider "performance-related testing" as the "family" with "performance testing" being one of the variabilities.

[3]This may or may not be distinct from the notion of "test technique" described in IEEE Testing Terminology.

guessing is an "experience-based test design technique" and "experience-based test practices include … exploratory testing, tours, attacks, and checklist-based testing". Other sources also do not agree whether error guessing is a technique [8, pp. 20, 22], [17, p. viii] or a practice [9, p. 5-14].

2) The following are test approaches that are categorized as test techniques in [17, p. 38], followed by sources that categorize them as test types:
   a) Capacity testing [8, p. 22], [12, p. 2] (implied by [18, p. 53] and by its quality [15], [17, Tab. A.1])
   b) Endurance testing [12, p. 2] (implied by [18, p. 55])
   c) Load testing [8, pp. 5, 20, 22], [11, p. 253], [16] (implied by [18, p. 54])
   d) Performance testing [8, pp. 7, 22, 26-27], [17, p. 7] (implied by [18, p. 53])
   e) Stress testing [8, pp. 9, 22], [11, p. 442] (implied by [18, p. 54])
3) Model-based testing is categorized as both a test practice [8, p. 22], [17, p. viii] and a test technique [21, p. 4] (implied by [17, p. 7], [11, p. 469]).
4) Data-driven testing is categorized as both a test practice [8, p. 22] and a test technique [21, p. 43].

B. Derived Test Approaches

In addition to methods of categorizing test approaches, the literature also provides multiple methods to derive new ones. Since the field of software is ever-evolving, being able to adapt to new developments, as well as being able to talk about and understand them, is crucial.

1) Coverage-derived Techniques: Test techniques are able to "identify test coverage items … and derive corresponding test cases" [8, p. 11] (similar in [11, p. 467]) in a "systematic" way [11, p. 464]. This means that a given coverage metric implies a test approach aimed to maximize it; for example, "path testing" is testing that "aims to execute all entry-to-exit control flow paths in a SUT's control flow graph" [9, p. 5013], thus maximizing the path coverage (see also #63, [24, Fig. 1]).

2) Quality-driven Types: Since test types are "focused on specific quality characteristics" [8, p. 15], [17, p. 7], [11, p. 473], they can derived from software qualities: "capabilit[ies] of software product[s] to satisfy stated and implied needs when used under specified conditions" [11, p. 424]. This is supported by [25] which says that reliability and performance testing, both examples of test types [8, 17], are based on their underlying qualities [25, p. 18].

After discussing this further (see #21 and #23), it was decided that tracking software qualities, in addition to testing approaches, would be worthwhile (see #27). This was done by capturing their definitions and any rationale for why it might be useful to consider an explicitly separate "test type" in a separate document, so this information could be captured without introducing clutter. Over time, software qualities were "upgraded" to test types when mentioned (or implied) by a source.

3) Requirements-driven Approaches: While not as universally applicable, some types of requirements have associated types of testing (e.g., functional, non-functional, security). This may mean that categories of requirements also imply related testing approaches (such as "technical testing"). Even assuming this is the case, some types of requirements do not apply to the code itself, and as such are out of scope (see #43):

- Nontechnical Requirement: a "requirement affecting product and service acquisition or development that is not a property of the product or service" [11, p. 293]
- Physical Requirement: a "requirement that specifies a physical characteristic that a system or system component must possess" [11, p. 322]

4) Attacks: Since attacks are given as a test practice [8, p. 34], different kinds of software attacks, such as code injection and password cracking, can also be used as test approaches.

Acknowledgment

TABLE I
IEEE Testing Terminology

Term	Definition	Examples
Approach	A "high-level test implementation choice, typically made as part of the test strategy design activity" that includes "test level, test type, test technique, test practice and the form of static testing to be used" [8, p. 10]; described by a test strategy [11, p. 472] and is also used to "pick the particular test case values" [11, p. 465]	black or white box, minimum and maximum boundary value testing [11, p. 465]
(Design)[a] Technique	A "defined" and "systematic" [11, p. 464] "procedure used to create or select a test model, identify test coverage items, and derive corresponding test cases" [8, p. 11] (similar in [11, p. 467]) "that … generate evidence that test item requirements have been met or that defects are present in a test item" [17, p. vii]; "a variety … is typically required to suitably cover any system" [8, p. 33] and is "often selected based on team skills and familiarity, on the format of the test basis", and on expectations [8, p. 23]	equivalence partitioning, boundary value analysis, branch testing [8, p. 11]
Level[b] (sometimes "Phase"[c] or "Stage"[d])	A stage of testing "typically associated with the achievement of particular objectives and used to treat particular risks", each performed in sequence [8, p. 12], [17, p. 6] with their "own documentation and resources" [11, p. 469]; more generally, "designat[es] … the coverage and detail" [11, p. 249]	unit/component testing, integration testing, system testing [8, p. 12], [17, p. 6], [11, p. 467]
Practice	A "conceptual framework that can be applied to … [a] test process to facilitate testing" [8, p. 14], [11, p. 471]; more generally, a "specific type of activity that contributes to the execution of a process" [11, p. 331]	scripted testing, exploratory testing, automated testing [8, p. 20]
Type	"Testing that is focused on specific quality characteristics" [8, p. 15], [17, p. 7], [11, p. 473]	security testing, usability testing, performance testing [8, p. 15], [11, p. 473]

[a] "Design technique" is sometimes abbreviated to "technique" [8, p. 11], [16].
[b] "Test level" can also refer to the scope of a test process; for example, "across the whole organization" or only "to specific projects" [8, p. 24].
[c] "Test phase" can be a synonym for "test level" [11, p. 469], [12, p. 9] but can also refer to the "period of time in the software life cycle" when testing occurs [11, p. 470], usually after the implementation phase [11, pp. 420, 509], [22, p. 56].
[d] [9, pp. 5-6 to 5-7], [16], [23, pp. 9, 13].

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. [Online]. Available: https://github.com/JacquesCarette/Drasil/tree/v0.1-alpha

[2] G. Tebes, L. Olsina, D. Peppino, and P. Becker, "TestTDO: A Top-Domain Software Testing Ontology," Curitiba, Brazil, May 2020, pp. 364–377.

[3] E. Souza, R. Falbo, and N. Vijaykumar, "ROoST: Reference Ontology on Software Testing," Applied Ontology, vol. 12, pp. 1–32, Mar. 2017.

[4] M. Unterkalmsteiner, R. Feldt, and T. Gorschek, "A Taxonomy for Requirements Engineering and Software Test Alignment," ACM Transactions on Software Engineering and Methodology, vol. 23, no. 2, pp. 1–38, Mar. 2014, arXiv:2307.12477 [cs]. [Online]. Available: http://arxiv.org/abs/2307.12477

[5] R. Patton, Software Testing, 2nd ed. Indianapolis, IN, USA: Sams Publishing, 2006.

[6] J. Peters and W. Pedrycz, Software Engineering: An Engineering Approach, ser. Worldwide series in computer science. John Wiley & Sons, Ltd., 2000.

[7] H. van Vliet, Software Engineering: Principles and Practice, 2nd ed. Chichester, England: John Wiley & Sons, Ltd., 2000.

[8] 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.

[9] H. Washizaki, Ed., Guide to the Software Engineering Body of Knowledge, Version 4.0, Jan. 2024. [Online]. Available: https://waseda.app.box.com/v/SWEBOK4-book

[10] P. Bourque and R. E. Fairley, Eds., Guide to the Software Engineering Body of Knowledge, Version 3.0. Washington, DC, USA: IEEE Computer Society Press, 2014. [Online]. Available: www.swebok.org

[11] ISO/IEC and IEEE, "ISO/IEC/IEEE International Standard - Systems and software engineering–Vocabulary," ISO/IEC/IEEE 24765:2017(E), Sep. 2017.

[12] ——, "ISO/IEC/IEEE International Standard - Systems and software engineering –Software testing –Part 1: General concepts," ISO/IEC/IEEE 29119-1:2013, Sep. 2013.

[13] ISO/IEC, "ISO/IEC 25019:2023 - Systems and software engineering –Systems and software Quality Requirements and Evaluation (SQuaRE) –Quality-in-use model," ISO/IEC 25019:2023, Nov. 2023. [Online]. Available: https://www.iso.org/obp/ui/en/#iso:std:iso-iec:25019:ed-1:v1:en

[14] IEEE, "IEEE Standard for System and Software Verification and Validation," IEEE Std 1012-2012

(Revision of IEEE Std 1012-2004), 2012.
[15] ISO/IEC, "ISO/IEC 25010:2023 - Systems and software engineering –Systems and software Quality Requirements and Evaluation (SQuaRE) –Product quality model," ISO/IEC 25010:2023, Nov. 2023. [Online]. Available: https://www.iso.org/obp/ui/#iso:std:iso-iec:25010:ed-2:v1:en
[16] M. Hamburg and G. Mogyorodi, editors, "ISTQB Glossary, v4.3," 2024. [Online]. Available: https://glossary.istqb.org/en_US/search
[17] ISO/IEC and IEEE, "ISO/IEC/IEEE International Standard - Software and systems engineering –Software testing –Part 4: Test techniques," ISO/IEC/IEEE 29119-4:2021(E), Oct. 2021.
[18] D. G. Firesmith, "A Taxonomy of Testing Types," Pittsburgh, PA, USA, 2015. [Online]. Available: https://apps.dtic.mil/sti/pdfs/AD1147163.pdf
[19] I. Kuļešovs, V. Arnicane, G. Arnicans, and J. Borzovs, "Inventory of Testing Ideas and Structuring of Testing Terms," vol. 1, pp. 210–227, Jan. 2013.
[20] M. H. Moghadam, "Machine Learning-Assisted Performance Testing," in Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ser. ESEC/FSE 2019. New York, NY, USA: Association for Computing Machinery, 2019, pp. 1187–1189. [Online]. Available: https://doi.org/10.1145/3338906.3342484
[21] B. Kam, "Web Applications Testing," Queen's University, Kingston, ON, Canada, Technical Report 2008-550, Oct. 2008. [Online]. Available: https://research.cs.queensu.ca/TechReports/Reports/2008-550.pdf
[22] W. E. Perry, Effective Methods for Software Testing, 3rd ed. Indianapolis, IN, USA: Wiley Publishing, Inc., 2006.
[23] P. Gerrard, "Risk-based E-business Testing - Part 1: Risks and Test Strategy," Systeme Evolutif, London, UK, Tech. Rep., 2000. [Online]. Available: https://www.agileconnection.com/sites/default/files/article/file/2013/XUS129342file1_0.pdf
[24] S. Sharma, K. Panwar, and R. Garg, "Decision Making Approach for Ranking of Software Testing Techniques Using Euclidean Distance Based Approach," International Journal of Advanced Research in Engineering and Technology, vol. 12, no. 2, pp. 599–608, Feb. 2021. [Online]. Available: https://iaeme.com/Home/issue/IJARET?Volume=12&Issue=2
[25] N. E. Fenton and S. L. Pfleeger, Software Metrics: A Rigorous & Practical Approach, 2nd ed. Boston, MA, USA: PWS Publishing Company, 1997.