# 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—Testing software is complicated, expensive, and often overlooked. Therefore, automating the software testing process is an area of interest, and understanding the underlying domain is an important prerequisite. Since existing software testing taxonomies are inadequate, the literature was investigated to attempt to build one that is systematic, rigorous, and "complete". Through this process, over five hundred test approaches were uncovered, as well as a (relatively) consistent method for classifying them and some methods for describing "implied" test approaches. In addition, a tool was built to automatically generate graphs based on the relations between them and track their ambiguities, in addition to those captured manually through the research process. As of now, ten terms were given as synonyms to two (or more) disjoint test approaches and fourteen pairs of test approaches may be synonyms and/or child and parent. There is also notable confusion around the terms "functional testing", "recovery testing", "scalability testing", and "performance testing", as well as over fifty more "minor" discrepancies. Overall, there is a need for testing terminology to be standardized to allow for consistency when discussing, analyzing, and/or implementing software testing.

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

## I. Background

Testing software is complicated, expensive, and often overlooked. 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

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

The "solution" to this gap was to define the scope of what kinds of "software testing" were of interest (II), then examine the existing literature for this subsection (III); this quickly reinforced the need for a taxonomy! Despite the amount of well understood and organized knowledge (IV), there are still many discrepancies and ambiguities, either within the same source or between various sources (V).

## II. Scope

Since the motivation for this project is the generation of test cases for code, only the "testing" component of Verification and Validation (V&V) is considered (see #22). For example, design reviews (see [5, p. 132]) and documentation reviews (see [p. 132]) are out of scope, since they focus on the V&V of the design and documentation of the code, respectively, and not on the code itself. Likewise, ergonomics testing and proximity-based testing (see [6]) are out of scope since they are for testing hardware systems. Security audits that focus on "an organization's ... processes and infrastructure" [6], are also out of scope, but security audits that "aim to ensure that all of the products installed on a site are secure when checked against the known vulnerabilities for those products" [7, p. 28] are not.

Sometimes, wider decisions must be made on whether a whole category of testing is in scope or not. For example, while all the examples of domain-specific testing given by [8, p. 26] are focused on hardware, this might not be representative of all types (e.g., ML model testing seems domain-specific).

This also means that only some aspects of some testing approaches are relevant. This mainly manifests as a testing approach that can verify both the V&V itself and the code. For example:

1) Error seeding is the "process of intentionally adding known faults to those already in a computer program", done to both "monitor[] the rate of detection and removal", which is a part of V&V of the V&V itself, "and estimat[e] the number of faults remaining" [5, p. 165], which helps verify the actual code.
2) Fault injection testing, where "faults are artificially introduced into the SUT", can be used to evaluate the effectiveness of a test suite [9, p. 5-18], which

is a part of V&V of the V&V itself, or "to test the robustness of the system in the event of internal and external failures" [10, p. 42], which helps verify the actual code.

3) "Mutation [t]esting was originally conceived as a technique to evaluate test suites in which a mutant is a slightly modified version of the SUT" [9, p. 5-15], which is in the realm of V&V of the V&V itself. However, it "can also be categorized as a structure-based technique" and can be used to assist fuzz and metamorphic testing [9, p. 5-15].

### A. Static Testing

Sometimes, the term "testing" excludes static testing [11, p. 222], [8, p. 13], restricting it to "dynamic validation" [9, p. 5-1] or "dynamic verification" "in which a system or component is executed" [5, p. 427]. Since "terminology is not uniform among different communities, and some use the term testing to refer to static techniques[1] as well" [9, p. 5-2], the scope of "testing" for the purpose of this project will include both "static testing" and "dynamic testing", as done by [10, p. 17], [12, pp. 8-9], and even a source that explicitly excluded static testing [5, p. 440]!

Static testing tends to be less systematic/consistent and often requires human intervention, which makes it less relevant to this project's end goal: generating test cases automatically. However, understanding the breadth of testing approaches provides a more complete picture of how software can be tested, how the various approaches are related to one another, and potentially how even parts of these "out-of-scope" approaches may be generated in the future! Therefore, these "out-of-scope" approaches are within the scope of this research (at least in the preliminary phase) and will be identified systematically and excluded before analysis. Even some dynamic methods, such as demonstrations and dynamic analysis, which fall under the realm of "evaluation" as opposed to "testing" [8, p. 13] may be ineligible for automatic generation, due to their reliance on human intervention.

## III. Methodology

### A. Source Order

0) Textbooks trusted at McMaster [13, 14, 15]
   - Ad hoc and arbitrary; not systematic
   - Colored maroon
1) Established standards (such as IEEE, ISO/IEC, and SWEBOK) [5, 6, 9, 10, 16, 17, 18, 19, 20, 21]
   - Standards organizations colored green
   - "Meta-level" commentaries or collections of terminology (often based on these standards), such as [8], colored blue
2) Other resources: less-formal classifications of terminology (such as [22]), 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[2] (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). 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 Static Testing, #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; [10] and [8] 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

[1] Not formally defined, but distinct from the notion of "test technique" described in IEEE Testing Terminology.

[2] 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.

(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

## IV. Observations

### A. Categories of Testing Approaches

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.

For classifying different kinds of tests, [10] 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 [21, p. 15] and data flow testing [p. 3] and the family of performance-related testing [23, p. 1187][3], and may also be implied for security testing, a test type that consists of "a number of techniques[4]" [21, 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" [10, p. 15], [21, 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" [21, 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 [10, 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" [21, 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 [10, pp. 4, 22], [21, p. 4], [9, p. 5-13], [6] and is sometimes contradicted [8, 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, [10, p. 34] says error 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 [10, pp. 20, 22], [21, p. viii] or a practice [9, p. 5-14].
2) The following are test approaches that are categorized as test techniques in [21, p. 38], followed by sources that categorize them as test types:
   a) Capacity testing [10, p. 22], [17, p. 2] (implied by [8, p. 53] and by its quality [20], [21, Tab. A.1])
   b) Endurance testing [17, p. 2] (implied by [8, p. 55])
   c) Load testing [10, pp. 5, 20, 22], [5, p. 253], [6]

[3]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.

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

(implied by [8, p. 54])
   d) Performance testing [10, pp. 7, 22, 26-27], [21, p. 7] (implied by [8, p. 53])
   e) Stress testing [10, pp. 9, 22], [5, p. 442] (implied by [8, p. 54])
3) Model-based testing is categorized as both a test practice [10, p. 22], [21, p. viii] and a test technique [24, p. 4] (implied by [21, p. 7], [5, p. 469]).
4) Data-driven testing is categorized as both a test practice [10, p. 22] and a test technique [24, 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" [10, p. 11] (similar in [5, p. 467]) in a "systematic" way [5, 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, [26, Fig. 1]).

2) Quality-driven Types: Since test types are "focused on specific quality characteristics" [10, p. 15], [21, p. 7], [5, 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" [5, p. 424]. This is supported by [27] which says that reliability and performance testing, both examples of test types [10, 21], are based on their underlying qualities [27, 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" [5, p. 293]
- Physical Requirement: a "requirement that specifies a physical characteristic that a system or system component must possess" [5, p. 322]

4) Attacks: Since attacks are given as a test practice [10, p. 34], different kinds of software attacks, such as code injection and password cracking, can also be used as test approaches.

## V. Discrepancies and Ambiguities

Many subsets of the current state of testing literature contain discrepancies and ambiguities. The following sections outline some of the larger ones, followed by more Minor Discrepancies, both among sources and within individual ones, that may be areas for further investigation.

### A. Synonyms

There exist many test approaches with multiple synonyms. For example, synonyms for specification-based testing include:

1) Black-Box Testing [10, p. 9], [21, p. 8], [5, p. 431], [9, p. 5-10], [6], [8, p. 46] (without hyphen), [28, p. 344], [15, p. 399]
2) Closed-Box Testing [10, p. 9], [5, p. 431]
3) Functional Testing [5, p. 196], [24, p. 44], [15, p. 399] (implied by [21, p. 129], [5, p. 431])
4) Domain Testing [9, p. 5-10]
5) Input Domain-Based Testing (implied by [16, p. 4-8])

While some of these synonyms may be incorrect, the legitimate ones carry different connotations. I use the terms "specification-based" and "structure-based testing" since they articulate the source of the information for designing test cases, but a team or project also using gray-box testing may prefer the terms "black-box" and "white-box testing" for consistency. Because of this, the existence of synonyms is not inherently a discrepancy.

However, there are instances in which a term is given a synonym to two (or more) disjoint, unrelated terms, which would be a source of ambiguity to teams using these terms. The following are examples that have arisen (synonyms in italics have at least one of their synonyms implied):

1) Invalid Testing:
   - Error Tolerance Testing [24, p. 45]
   - Negative Testing ([6]; implied by 21, p. 10)
2) Soak Testing:
   - Endurance Testing [21, p. 39]
   - Reliability Testing (12, Tab. 2; 7, Tab. 1, p. 26)
3) User Scenario Testing:
   - Scenario Testing [6]
   - Use Case Testing (24, p. 48; although "an actor can be a user or another system" [21, p. 20])
4) (Multiple) Condition Testing:
   - Branch Condition Combination Testing ([6]; 13, p. 120)

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" [10, p. 10]; described by a test strategy [5, p. 472] and is also used to "pick the particular test case values" [5, p. 465] | black or white box, minimum and maximum boundary value testing [5, p. 465] |
| (Design)[a] Technique | A "defined" and "systematic" [5, p. 464] "procedure used to create or select a test model, identify test coverage items, and derive corresponding test cases" [10, p. 11] (similar in [5, p. 467]) "that … generate evidence that test item requirements have been met or that defects are present in a test item" [21, p. vii]; "a variety … is typically required to suitably cover any system" [10, p. 33] and is "often selected based on team skills and familiarity, on the format of the test basis", and on expectations [10, p. 23] | equivalence partitioning, boundary value analysis, branch testing [10, 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 [10, p. 12], [21, p. 6] with their "own documentation and resources" [5, p. 469]; more generally, "designat[es] … the coverage and detail" [5, p. 249] | unit/component testing, integration testing, system testing [10, p. 12], [21, p. 6], [5, p. 467] |
| Practice | A "conceptual framework that can be applied to … [a] test process to facilitate testing" [10, p. 14], [5, p. 471]; more generally, a "specific type of activity that contributes to the execution of a process" [5, p. 331] | scripted testing, exploratory testing, automated testing [10, p. 20] |
| Type | "Testing that is focused on specific quality characteristics" [10, p. 15], [21, p. 7], [5, p. 473] | security testing, usability testing, performance testing [10, p. 15], [5, p. 473] |

[a] "Design technique" is sometimes abbreviated to "technique" [10, p. 11], [6].
[b] "Test level" can also refer to the scope of a test process; for example, "across the whole organization" or only "to specific projects" [10, p. 24].
[c] "Test phase" can be a synonym for "test level" [5, p. 469], [17, p. 9] but can also refer to the "period of time in the software life cycle" when testing occurs [5, p. 470], usually after the implementation phase [5, pp. 420, 509], [25, p. 56].
[d] [9, pp. 5-6 to 5-7], [6], [12, pp. 9, 13].

- Branch Condition Testing [13, p. 120]

5) Extended Branch Coverage:
   - Branch Condition Combination Testing [15, p. 422]
   - Branch Condition Testing [15, p. 422]
6) Link Testing:
   - Branch Testing (implied by 21, p. 24)
   - Component Integration Testing [24, p. 45]
   - Integration Testing (implied by 12, p. 13)
7) State-based Testing:
   - State Transition Testing [8, p. 47]
   - State-based Web Browser Testing [29, p. 193]
8) Static Verification:
   - Static Assertion Checking [30, p. 343]
   - Static Testing (implied by 30, p. 343)
9) Production Acceptance Testing:
   - Operational Testing [6]
   - Production Verification Testing
10) Operational Testing:
   - Field Testing
   - Qualification Testing

Some interesting notes about these synonyms:

- The terms are not synonyms, although endurance testing is given as a kind of reliability testing by [8, p. 55].
- "Scenario testing" and "use case testing" are given as synonyms by [6] and [24, pp. 47-49], but listed separately by [10, p. 22], which also gives "use case testing" as a "common form of scenario testing" [21, p. 20].
- [30, p. 343] lists Runtime Assertion Checking (RAC) and Software Verification (SV) as "two complementary forms of assertion checking"; based on how the term "static assertion checking" is used by [31, p. 345], it seems like this should be the complement to RAC instead.
- "Operational" and "production acceptance testing" are treated as synonyms by [6], but listed separately by [8, p. 30].
- "Production acceptance testing" [8, p. 30] seems to be the same as "production verification testing" [10, p. 22], but neither are defined.

There are also some pairs of synonyms where one is sometimes described as a sub-approach of the other, making the relationship between them unclear. As above, some examples are given below, with as many sources given as possible (any relationships without a source provided are inferred from their definitions):

1) "All transitions testing" is called a sub-approach of "state transition testing" in [21, p. 19], but the two are called synonyms in [24, p. 15].
2) "Co-existence testing" is called a sub-approach of

"compatibility testing" in (20; 10, p. 3; 21, Tab. A.1), but the two are called synonyms in [21, p. 37].

3) "Fault tolerance testing" is called a sub-approach of "robustness testing" in [8, p. 56], but the two are called synonyms in [6].
4) "Functional testing" is called a sub-approach of "specification-based testing" in [21, p. 38], but the two are called synonyms in (5, p. 196; 24, p. 44; 15, p. 399; implied by 21, p. 129; 5, p. 431).
5) "Orthogonal array testing" is called a sub-approach of "pairwise testing" in [32, p. 1055], but the two are called synonyms in (9, p. 5-11; 33, p. 473).
6) "Performance testing" is called a sub-approach of "performance-related testing" in (10, p. 22; 21, p. 38), but the two are called synonyms in [23, p. 1187].
7) "Use case testing" is called a sub-approach of "scenario testing" in (21, p. 20 ), but the two are called synonyms in ([6]; 24, pp. 47-49).
8) "Condition testing" is implied to be a sub-approach of "decision testing" in [6], but the two are called synonyms in (9, p. 5-13).
9) "Dynamic analysis" is implied by its static counterpart to be a sub-approach of "dynamic testing" in (10, pp. 9, 17, 25, 28; [6]), but the two are called synonyms in [5, p. 149].
10) "Beta testing" is implied to be a sub-approach of "user testing" in [8, p. 39], but the two are implied to be synonyms in [8, p. 39].
11) "Functionality testing" may be a sub-approach of "functional suitability testing", but the two are called synonyms in [6].
12) "Programmer testing" is called a sub-approach of "developer testing" in [8, p. 39], but the two may be synonyms.
13) "Structured walkthroughs" may be a sub-approach of "walkthroughs", but the two are called synonyms in [6].
14) "Functionality testing" may be a sub-approach of "functional suitability testing", but the two are implied to be synonyms in [6].

The relationships between the following pairs of approaches aren't given in any investigated sources and are also ambiguous, based on their definitions. In each pair, the first may be a sub-approach of the second, or they may be synonyms.

- Field testing and operational testing
- Organization-based testing and role-based testing
- System qualification testing and system testing

Some interesting notes about these two lists of discrepancies:

- Fault tolerance testing may also be a subtype[5] of reliability testing [5, p. 375; 9, p. 7-10], which is distinct from reliability testing [8, p. 53].
- The distinction between organization- and role-based testing in [8, pp. 17, 37, 39] seems arbitrary, but further investigation may prove it to be meaningful (see #59).

[5]Not formally defined, but distinct from the notion of "test type" described in IEEE Testing Terminology.

### B. Parent Relations

There are also some issues with how parent relations are defined in the literature, the most trivial being self-cycles, where a source describes a test approach as a subset of itself:

1) Performance Testing (12, Tab. 2; 7, Tab. 1)
2) Usability Testing (12, Tab. 2; 7, Tab. 1)

Interestingly, [12, Tab. 2] and [7, Tab. 1] do not describe performance testing as a sub-category of usability testing, which would have been more meaningful.

### C. Functional Testing

Throughout the literature, "functional testing" seems to be described in many ways, alongside other, potentially related, terms:

- Specification-based Testing: This is defined as "testing in which the principal test basis is the external inputs and outputs of the test item" [10, p. 9], which agrees with a definition of "functional testing": "testing that … focuses solely on the outputs generated in response to selected inputs and execution conditions" [5, p. 196]. Notably, [5] lists both as synonyms of "black-box testing" (pp. 431, 196, respectively). However, they are sometimes defined as separate terms: "specification-based testing" as "testing based on an analysis of the specification of the component or system" (including "black-box testing" as a synonym) and "functional testing" as "testing performed to evaluate if a component or system satisfies functional requirements" (specifying no synonyms) [6]. This definition of "functional testing" references [5, p. 196] ("testing conducted to evaluate the compliance of a system or component with specified functional requirements") which has "black-box testing" as a synonym, and mirrors [10, p. 21] (testing "used to check the implementation of functional requirements"). Overall, specification-based testing [10, pp. 2-4, 6-9, 22] and black-box testing (9, p. 5-10; 3, p. 3) are test design techniques used to "derive corresponding test cases" [10, p. 11] (from given "selected inputs and execution conditions" [5, p. 196]).
- Correctness Testing: [9, p. 5-7] says "test cases can be designed to check that the functional specifications are correctly implemented, which is variously referred to in the literature as conformance testing, correctness testing or functional testing"; this mirrors previous definitions of "functional testing" (10, p. 21; 5, p. 196) but groups it with "correctness testing". Since "correctness" is a software quality (5, p. 104;

9, p. 3-13) which is what defines a "test type" [10, p. 15], it seems consistent to label "functional testing" as a "test type" [10, pp. 15, 20, 22]. This is listed separately from "functionality testing" by [8, p. 53].

- Conformance Testing: As mentioned above, [9, p. 5-7] says testing "that the functional specifications are correctly implemented" can be called "conformance testing" or "functional testing". The definition of "conformance testing" is later given as testing used "to verify that the SUT conforms to standards, rules, specifications, requirements, design, processes, or practices" [9, p. 5-7]. This definition seems to be a superset of the testing mentioned earlier; the former only lists "specifications" while the latter also includes "standards", "rules", "requirements", "design", "processes", and "practices"!
  Another complicating factor is that "compliance testing" is also given as a synonym of "conformance testing" [24, p. 43], which seems plausible. However, "conformance testing" can also be defined as testing that evaluates the degree to which "results … fall within the limits that define acceptable variation for a quality requirement" [5, p. 93] , which seems to describe something different. Perhaps this second definition of "conformance testing" should be used, and the previous definition of "compliance testing" be used for describing compliance with external standards, rules, etc. to keep them distinct.
- Functional Suitability Testing: Procedure testing is called a "type of functional suitability testing" [10, p. 7], but no definition of "functional suitability testing" is given. "Functional suitability" is the "capability of a product to provide functions that meet stated and implied needs of intended users when it is used under specified conditions", including meeting "the functional specification" [20]. This seems to align with the definition of "functional testing" as related to "black-box/ specification-based testing". "Functional suitability" has three child terms: "functional completeness" (the "capability of a product to provide a set of functions that covers all the specified tasks and intended users' objectives"), "functional correctness" (the "capability of a product to provide accurate results when used by intended users"), and "functional appropriateness" (the "capability of a product to provide functions that facilitate the accomplishment of specified tasks and objectives") [20]. Notably, "functional correctness", which includes precision and accuracy (20; [6]), seems to align with the quality/ies that would be tested by "correctness" testing.
- Functionality Testing: "Functionality" is defined as the "capabilities of the various … features provided by a product" [5, p. 196] and is said to be a synonym of "functional suitability" [6], although it seems like it should really be its "parent". Its associated test type is implied to be a sub-approach of build verification testing [6] and made distinct from "functional testing"; interestingly, security is described as a sub-approach of both non-functional and functionality testing [12, Tab. 2]. This is listed separately from "correctness testing" by [8, p. 53].

### D. Operational (Acceptance) Testing

Some sources refer to "operational acceptance testing" (10, p. 22; [6]) while some refer to "operational testing" (9, p. 6-9, in the context of software engineering operations; 34; 5, p. 303; 16, pp. 4-6, 4-9). A distinction is sometimes made [8, p. 30] but without accompanying definitions, it is hard to evaluate its merit. Since this terminology is not standardized, I propose that the two terms are treated as synonyms (as done by other sources [35, 36]) as a type of acceptance testing (10, p. 22; [6]) that focuses on "non-functional" attributes of the system [35] .

A summary of given definitions of "operational (acceptance) testing" is that it is "test[ing] to determine the correct installation, configuration and operation of a module and that it operates securely in the operational environment" [34] or "evaluate a system or component in its operational environment" [5, p. 303], particularly "to determine if operations and/or systems administration staff can accept [it]" [6].

### E. Recovery Testing

There are many terms related to "recovery testing", which is "testing … aimed at verifying software restart capabilities after a system crash or other disaster" [9, p. 5-9] including "recover[ing] the data directly affected and re-establish[ing] the desired state of the system" (20; similar in 9, p. 7-10) so that the system "can perform required functions" [5, p. 370]. It is also called "recoverability testing" [24, p. 47] and potentially "restart & recovery (testing)" [12, Fig. 5]. The following terms, along with "recovery testing" itself [10, p. 22] are all classified as test types, and the relations between them can be found in Figure ??.

- Recoverability Testing: Testing "how well a system or software can recover data during an interruption or failure" (9, p. 7-10; similar in 20) and "re-establish the desired state of the system" [20]. Given as a synonym for "recovery testing" by [24, p. 47].
- Disaster/Recovery Testing: Testing to evaluate if a system can "return to normal operation after a hardware or software failure" [5, p. 140] or if "operation of the test item can be transferred to a different operating site and … be transferred back again once the failure has been resolved" [21, p. 37]. These two definitions seem to describe different aspects of the system, where the first is intrinsic to the hardware/software and the second might not be.
- Backup and Recovery Testing: Testing "that measures the degree to which system state can be restored

from backup within specified parameters of time, cost, completeness, and accuracy in the event of failure" [17, p. 2]. This may be what is meant by "recovery testing" in the context of performance-related testing and seems to correspond to the definition of "disaster/recovery testing" in [5, p. 140].

- Backup/Recovery Testing: Testing that determines the ability "to restor[e] from back-up memory in the event of failure, without transfer[ing] to a different operating site or back-up system" [21, p. 37]. This seems to correspond to the definition of "disaster/recovery testing" in [21, p. 37]. It is also given as a sub-type of "disaster/recovery testing", even though that tests if "operation of the test item can be transferred to a different operating site" (p. 37).
- Failover/Recovery Testing: Testing that determines the ability "to mov[e] to a back-up system in the event of failure, without transfer[ing] to a different operating site" [21, p. 37]. This is given as a sub-type of "disaster/recovery testing", even though that tests if "operation of the test item can be transferred to a different operating site" (p. 37).
- Failover Testing: Testing that "validates the SUT's ability to manage heavy loads or unexpected failure to continue typical operations" [9, p. 5-9] by entering a "backup operational mode in which [these responsibilities] … are assumed by a secondary system" [6]. While not explicitly related to recovery, "failover/recovery testing" also describes the idea of "failover", and (author?) uses the term "failover and recovery testing" [8, p. 56], which could be a synonym of both of these terms.

### F. Scalability Testing

1) Reference [21, p. 39] gives "scalability testing" as a synonym of "capacity testing" while other sources differentiate between the two [8, p. 53], [37, pp. 22-23]
2) Reference [21, p. 39] includes the external modification of the system as part of "scalability", while [20] implies that it is limited to the system itself
3) SWEBOK V4's definition of "scalability testing" [9, p. 5-9] is really a definition of usability testing!

### G. Performance Testing

"Performance testing" is defined as testing "conducted to evaluate the degree to which a test item accomplishes its designated functions" (10, p. 7; 5, p. 320; similar in 21, pp. 38-39; 23, p. 1187). It does this by "measuring the performance metrics" (23, p. 1187; similar in [6]) (such as the "system's capacity for growth" [7, p. 23]), "detecting the functional problems appearing under certain execution conditions" [23, p. 1187], and "detecting violations of non-functional requirements under expected and stress conditions" (23, p. 1187; similar in [9, p. 5-9]). It is performed either …

1) … "within given constraints of time and other resources" (10, p. 7; 5, p. 320; similar in 23, p. 1187), or
2) … "under a 'typical' load" [21, p. 39].

It is listed as a subset of performance-related testing, which is defined as testing "to determine whether a test item performs as required when it is placed under various types and sizes of 'load'" [21, p. 38], along with other approaches like load and capacity testing [10, p. 22]. However, [9, p. 5-9] gives "capacity and response time" as examples of "performance characteristics" that performance testing would seek to "assess", which seems to imply that these are subapproaches to performance testing instead.

It seems that "performance testing" and "performance-related testing" are treated as synonyms by some sources (9, p. 5-9; 23, p. 1187). This makes sense because of how general the concept of "performance" is; most definitions of "performance testing" seem to treat it as a category of tests. However, it seems more consistent to infer that the definition of "performance-related testing" is the more general one often assigned to "performance testing" performed "within given constraints of time and other resources" (10, p. 7; 5, p. 320; similar in 23, p. 1187), and "performance testing" is a subapproach of this performed "under a 'typical' load" [21, p. 39]. This has other implications for relations between these types of testing; for example, "load testing" usually occurs "between anticipated conditions of low, typical, and peak usage" (10, p. 5; 21, p. 39; 5, p. 253 ; [6]), so it is a child of "performance-related testing" and a parent of "performance testing".

Similarly, "performance" and "performance efficiency" are both given as software qualities by (author?), with the latter defined as the "performance relative to the amount of resources used under stated conditions" [5, p. 319] or the "capability of a product to perform its functions within specified time and throughput parameters and be efficient in the use of resources under specified conditions" [20]. Initially, there didn't seem to be any meaningful distinction between the two, although the term "performance testing" is defined [5, p. 320] and used by (author?) and the term "performance efficiency testing" is also used by (author?) (but not defined explicitly). Further discussion (see #43) brought us to the conclusion that "performance efficiency testing" is a subset of "performance testing", and the difference of "relative to the amount of resources used" or "be efficient in the use of resources" between the two is meaningful.

This results in the relations between performance-related testing approaches (with the changes from Recovery Testing and Scalability Testing made implicitly) shown in Figure 1.

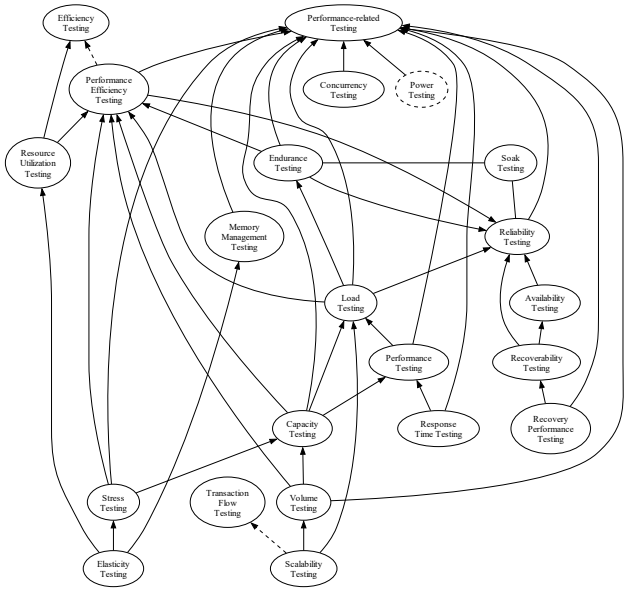


Fig. 1. The relations between "performance-related testing" terms.

## H. Minor Discrepancies

The following sections outline more "minor" discrepancies/ambiguities found in the literature, group by the "categories" of sources outlined in Methodology.

1) Discrepancies in Standards: The following discrepancies are from [5, 10, 17, 18, 19, 20, 21, 34, 38, 39]:

1) "Compatibility testing" is defined as "testing that measures the degree to which a test item can function satisfactorily alongside other independent products in a shared environment (co-existence), and where necessary, exchanges information with other systems or components (interoperability)" [10, p. 3]. This definition is nonatomic as it combines the ideas of "co-existence" and "interoperability". The term "interoperability testing" is not defined, but is used three times [10, pp. 22, 43] (although the third usage seems like it should be "portability testing"). This implies that "co-existence testing" and "interoperability testing" should be defined as their own terms, which is supported by definitions of "co-existence" and "interoperability" often being separate ([6]; 5, pp. 73, 237), the definition of "interoperability testing" from [5, p. 238], and the decomposition of "compatibility" into "co-existence" and "interoperability" by [20]!
   - The "interoperability" element of "compatibility testing" is explicitly excluded by [21, p. 37], (incorrectly) implying that "compatibility testing" and "co-existence testing" are synonyms.
   - The definition of "compatibility testing" in [24, p. 43] unhelpfully says "See interoperability testing", adding another layer of confusion to the direction of their relationship.
2) "Fuzz testing" is "tagged" (?) as "artificial intelligence" [10, p. 5], although I don't think this is a set-in-stone requirement.
3) Integration, system, and system integration testing are all listed as "common test levels" (10, p. 12; 21, p. 6), but no definitions are given for the latter two, making it unclear what "system integration testing" is; it is a combination of the two? somewhere on the spectrum between them? It is listed as a child of integration testing by [6] and of system testing by [8, p. 23].
4) Similarly, component, integration, and component integration testing are all listed in [5], but "component integration testing" is only defined as "testing of groups of related components" [5, p. 82]; it is a combination of the two? somewhere on the spectrum between them? Likewise, it is listed as a child of integration testing by [6].
5) "Installability testing" is given as a test type (10, p. 22; 21, p. 38) but is sometimes called a test level as "installation testing" [14, p. 445].
6) Retesting and regression testing seem to be separated from the rest of the testing approaches [10, p. 23], but it is not clearly detailed why; [40, p. 3] consider regression testing to be a test level.
7) A component is an "entity with discrete structure … within a system considered at a particular level of analysis" [18] and "the terms module, component, and unit [sic] are often used interchangeably or defined to be subelements of one another in different ways depending upon the context" with no standardized relationship [5, p. 82]. This means unit/component/module testing can refer to the testing of both a module and a specific function in a module (see #14). However, "component" is sometimes defined differently than "module": "components differ from classical modules for being reused in different contexts independently of their development" [41, p. 107], so distinguishing the two may be necessary.
8) A typo in [21, Fig. 2] means that "specification-based techniques" is listed twice, when the latter should be "structure-based techniques".
9) (author?) define an "extended entry (decision) table" both as a decision table where the "conditions consist of multiple values rather than simple Booleans" [21, p. 18] and one where "the conditions and actions are generally described but are incomplete" [5, p. 175] .
10) The subcategories of "performance-related testing" that are categorized as test types by other sources [5, 10, 17] are categorized as techniques in [21, p. 39].
11) (author?) provide a definition for "inspections and audits" [5, p. 228], despite also giving definitions for

"inspection" (p. 227) and "audit" (p. 36); while the first term could be considered a superset of the latter two, this distinction doesn't seem useful.

12) (author?) say that "test level" and "test phase" are synonyms, both meaning a "specific instantiation of [a] test sub-process" (5, pp. 469, 470; 17, p. 9), but there are also alternative definitions for them. "Test level" can also refer to the scope of a test process; for example, "across the whole organization" or only "to specific projects" [10, p. 24], while "test phase" can also refer to the "period of time in the software life cycle" when testing occurs [5, p. 470], usually after the implementation phase [5, pp. 420, 509], [25, p. 56].
13) (author?) use the same definition for "partial correctness" [5, p. 314] and "total correctness" (p. 480).

2) Discrepancies in "Meta-Level" Sources: The following discrepancies are either intrinsic to SWEBOK [9, 16], ISTQB [6], or another collection of testing terminology [so far only 8] (as described in Methodology), or between one of these and a source from a previous "category":

resume

1) SWEBOK V4 defines "privacy testing" as testing that "assess[es] the security and privacy of users' personal data to prevent local attacks" [9, p. 5-10]; this seems to overlap with (author?)'s definition of "security testing", which is "conducted to evaluate the degree to which a test item, and associated data and information, are protected so that" only "authorized persons or systems" can use them as intended [10, p. 9], both in scope and name.
2) Various sources say that alpha testing is performed by different people, including "only by users within the organization developing the software" [5, p. 17], by "a small, selected group of potential users" [9, p. 5-8], or "in the developer's test environment by roles outside the development organization" [6].
3) While correct, ISTQB's definition of "specification-based testing" is not helpful: "testing based on an analysis of the specification of the component or system" [6].
4) "ML model testing" and "ML functional performance" are defined in terms of "ML functional performance criteria", which is defined in terms of "ML functional performance metrics", which is defined as "a set of measures that relate to the functional correctness of an ML system" [6]. The use of "performance" (or "correctness") in these definitions are at best ambiguous and at worst incorrect.
5) While ergonomics testing is out of scope (as it tests hardware, not software), its definition of "testing to determine whether a component or system and its input devices are being used properly with correct posture" [6] seems to focus on how the system is used as opposed to the system itself.
6) The definition of "math testing" given by [6] is too specific to be useful, likely taken from an example instead of a general definition: "testing to determine the correctness of the pay table implementation, the random number generator results, and the return to player computations".
7) A similar issue exists with multiplayer testing, where its definition specifies "the casino game world" [6].
8) Thirdly, "par sheet testing" from [6] seems to refer to this specific example and does not seem more widely applicable, since a "PAR sheet" is "a list of all the symbols on each reel of a slot machine" [42].
9) [6] describe the term "software in the loop" as a kind of testing, while the source it references seems to describe "Software-in-the-Loop-Simulation" as a "simulation environment" that may support software integration testing [43, p. 153]; is this a testing approach or a tool that supports testing?
10) The source cited for the definition of "test type" from [6] does not seem to provide a definition itself.
11) The same is true for "visual testing".
12) The same is true for "security attack".
13) There is disagreement on the structure of tours; they can either be quite general [10, p. 34] or "organized around a special focus" [6].
14) While model testing is said to test the object under test, it seems to describe testing the models themselves [8, p. 20]; using the models to test the object under test seems to be called "driver-based testing" (p. 33).
15) "System testing" is listed as a subtype of "system testing" by [8, p. 23].
16) "Hardware-" and "human-in-the-loop testing" have the same acronym: "HIL"[6] [8, p. 23].
17) The same is true for "customer" and "contract(ual) acceptance testing" ("CAT") [8, p. 30].
18) The acronym "SoS" is used but not defined by [8, p. 23].
19) It is ambiguous whether "tool/environment testing" refers to testing the tools/environment themselves or using them to test the object under test; the latter is implied, but the wording of its subtypes [8, p. 25] seems to imply the former.
20) (author?) say that performance and security testing are subtypes of reliability testing [20], but these are all listed separately by [8, p. 53].
21) The distinctions between development testing [5, p. 136], developmental testing [8, p. 30], and developer testing (8, p. 39; 12, p. 11) are unclear and seem miniscule.

3) Discrepancies in McMaster-Trusted Sources: The following discrepancies are either intrinsic to a textbook trusted at McMaster [13, 14, 15], between these textbooks,

[6] "HiL" is used for the former by [44, p. 2].

or between one of them and a source from a previous "category":

resume

1) "Load testing" is defined as using loads "usually between anticipated conditions of low, typical, and peak usage" [10, p. 5], while (author?) says the loads should as large as possible [13, p. 86].

4) Discrepancies in Other Sources: The following discrepancies are either intrinsic to a source not in one of the above "categories", between two of these sources, or between one of these and a source from a previous "category":

resume

1) (author?) lists "three [backup] location categories: local, offsite and cloud based [sic]" [37, p. 16], but does not define or discuss "offsite backups" (pp. 16-17).
2) (author?) claim that [28] defines "prime path coverage" [29, p. 184], but it doesn't.
3) "State-based" is misspelled by (author?) as "state-base" [24, pp. 13, 15] and "stated-base" (Tab. 1).
4) Although ad hoc testing is sometimes classified as a "technique" [9, p. 5-14], it is one in which "no recognized test design technique is used" [24, p. 42].
5) (author?)'s definition of "boundary value testing" says "See boundary value analysis," but this definition is not present [24].
6) "Conformance testing" is implied to be a synonym of "compliance testing" by (author?), which only makes sense because of the vague definition of "compliance testing": "testing to determine the compliance of the component or system" [24, p. 43].
7) (author?) seems to imply that "mutation testing" is a synonym of "back-to-back testing" [24, p. 46], but these are two quite distinct techniques.
8) (author?) also says that the goal of negative testing is "showing that a component or system does not work" [24, p. 46] , which is not true; if robustness is an important quality for the system, then testing the system "in a way for which it was not intended to be used" [6] (i.e., negative testing) is one way to help test this!
9) "Program testing" is given as a synonym of "component testing" [24, p. 46], although it probably should be a synonym of "system testing" instead.
10) (author?) give "white-", "grey-", and "black-box testing" as synonyms for "module", "integration", and "system testing", respectively [45, p. 18] , but this mapping is incorrect; black-box testing can be performed on a module, for example . This makes the claim that "red-box testing" is a synonym for "acceptance testing" (p. 18) lose credibility.
11) Availability testing isn't assigned to a test priority [12, Tab. 2], despite the claim that "the test types[7] have been allocated a slot against the four test priorities" (p. 13); I think usability and/or performance would have made sense.
12) "Visual browser validation" is described as both static and dynamic in the same table [12, Tab. 2], even though they are implied to be orthogonal classifications: "test types can be static or dynamic" (p. 12, emphasis added).
13) (author?) makes a distinction between "transaction verification" and "transaction testing" [12, Tab. 2] and also uses the phrase "transaction flows" (Fig. 5) but doesn't explain them.
14) The phrase "continuous automated testing" [12, p. 11] is redundant since continuous testing is a subcategory of automated testing (10, p. 35, [6]).
15) End-to-end functionality testing is not indicated to be functionality testing [12, Tab. 2].
16) (author?)'s definition for "security audits" seems too specific, only applying to "the products installed on a site" and "the known vulnerabilities for those products" [7, p. 28].

Acknowledgment

[7] "Each type of test addresses a different risk area" [12, p. 12], which is distinct from the notion of "test type" described in IEEE Testing Terminology.

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

[6] M. Hamburg and G. Mogyorodi, editors, "ISTQB Glossary, v4.3," 2024. [Online]. Available: https://glossary.istqb.org/en_US/search

[7] P. Gerrard, "Risk-based E-business Testing - Part 2: Test Techniques and Tools," Systeme Evolutif, London, UK, Tech. Rep., 2000. [Online]. Available: wenku.uml.com.cn/document/test/EBTestingPart2.pdf

[8] D. G. Firesmith, "A Taxonomy of Testing Types," Pittsburgh, PA, USA, 2015. [Online]. Available: https://apps.dtic.mil/sti/pdfs/AD1147163.pdf

[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] 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.

[11] P. Ammann and J. Offutt, Introduction to Software Testing, 2nd ed. Cambridge, United Kingdom: Cambridge University Press, 2017. [Online]. Available: https://eopcw.com/find/downloadFiles/11

[12] 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

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

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

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

[16] 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

[17] 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, Sep. 2013.

[18] 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

[19] IEEE, "IEEE Standard for System and Software Verification and Validation," IEEE Std 1012-2012 (Revision of IEEE Std 1012-2004), 2012.

[20] 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

[21] 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.

[22] 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.

[23] 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

[24] 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

[25] W. E. Perry, Effective Methods for Software Testing, 3rd ed. Indianapolis, IN, USA: Wiley Publishing, Inc., 2006.

[26] 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
[27] N. E. Fenton and S. L. Pfleeger, Software Metrics: A Rigorous & Practical Approach, 2nd ed. Boston, MA, USA: PWS Publishing Company, 1997.
[28] K. Sakamoto, K. Tomohiro, D. Hamura, H. Washizaki, and Y. Fukazawa, "POGen: A Test Code Generator Based on Template Variable Coverage in Gray-Box Integration Testing for Web Applications," in Fundamental Approaches to Software Engineering, V. Cortellessa and D. Varró, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, Mar. 2013, pp. 343–358. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-642-37057-1_25
[29] S. Doğan, A. Betin-Can, and V. Garousi, "Web application testing: A systematic literature review," Journal of Systems and Software, vol. 91, pp. 174–201, 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121214000223
[30] P. Chalin, J. R. Kiniry, G. T. Leavens, and E. Poll, "Beyond Assertions: Advanced Specification and Verification with JML and ESC/Java2," in Formal Methods for Components and Objects, F. S. de Boer, M. M. Bonsangue, S. Graf, and W.-P. de Roever, Eds. Berlin, Heidelberg: Springer, 2006, pp. 342–363.
[31] S. K. Lahiri, K. L. McMillan, R. Sharma, and C. Hawblitzel, "Differential Assertion Checking," in Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, ser. ESEC/FSE 2013. New York, NY, USA: Association for Computing Machinery, Aug. 2013, pp. 345–355. [Online]. Available: https://dl.acm.org/doi/10.1145/2491411.2491452
[32] R. Mandl, "Orthogonal Latin squares: an application of experiment design to compiler testing," Communications of the ACM, vol. 28, no. 10, pp. 1054–1058, Oct. 1985. [Online]. Available: https://doi.org/10.1145/4372.4375
[33] P. Valcheva, "Orthogonal Arrays and Software Testing," in 3rd International Conference on Application of Information and Communication Technology and Statistics in Economy and Education, D. G. Velev, Ed., vol. 200. Sofia, Bulgaria: University of National and World Economy, Dec. 2013, pp. 467–473. [Online]. Available: https://icaictsee-2013.unwe.bg/proceedings/ICAICTSEE-2013.pdf
[34] ISO/IEC, "ISO/IEC TS 20540:2018 - Information technology – Security techniques –Testing cryptographic modules in their operational environment," ISO/IEC TS 20540:2018, May 2018. [Online]. Available: https://www.iso.org/obp/ui#iso:std:iso-iec:ts:20540:ed-1:v1:en
[35] LambdaTest, "What is Operational Testing: Quick Guide With Examples," 2024. [Online]. Available: https://www.lambdatest.com/learning-hub/operational-testing
[36] C. Bocchino and W. Hamilton, "Eastern Range Titan IV/Centaur-TDRSS Operational Compatibility Testing," in International Telemetering Conference Proceedings. San Diego, CA, USA: International Foundation for Telemetering, Oct. 1996. [Online]. Available: https://repository.arizona.edu/bitstream/handle/10150/607608/ITC_1996_96-01-4.pdf?sequence=1&isAllowed=y
[37] M. Bas, "Data Backup and Archiving," Bachelor Thesis, Czech University of Life Sciences Prague, Praha-Suchdol, Czechia, Mar. 2024. [Online]. Available: https://theses.cz/id/60licg/zaverecna_prace_Archive.pdf
[38] ISO, "ISO 21384-2:2021 - Unmanned aircraft systems –Part 2: UAS components," ISO 21384-2:2021, Dec. 2021. [Online]. Available: https://www.iso.org/obp/ui#iso:std:iso:21384:-2:ed-1:v1:en
[39] ——, "ISO 13849-1:2015 - Safety of machinery –Safety-related parts of control systems –Part 1: General principles for design," ISO 13849-1:2015, Dec. 2015. [Online]. Available: https://www.iso.org/obp/ui#iso:std:iso:13849:-1:ed-3:v1:en
[40] E. F. Barbosa, E. Y. Nakagawa, and J. C. Maldonado, "Towards the Establishment of an Ontology of Software Testing," vol. 6, San Francisco, CA, USA, Jan. 2006, pp. 522–525.
[41] L. Baresi and M. Pezzè, "An Introduction to Software Testing," Electronic Notes in Theoretical Computer Science, vol. 148, no. 1, pp. 89–111, Feb. 2006. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1571066106000442
[42] M. Bluejay, "Slot Machine PAR Sheets," May 2024. [Online]. Available: https://easy.vegas/games/slots/par-sheets
[43] Knüvener Mackert GmbH, Knüvener Mackert SPICE Guide, 7th ed. Reutlingen, Germany: Knüvener Mackert GmbH, 2022. [Online]. Available: https://knuevenermackert.com/wp-content/uploads/2021/06/SPICE-BOOKLET-2022-05.pdf
[44] S. Preuße, H.-C. Lapp, and H.-M. Hanisch, "Closed-loop System Modeling, Validation, and Verification," in Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies & Factory Automation (ETFA 2012). Krakow, Poland: IEEE, 2012, pp. 1–8. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6489679
[45] H. Sneed and S. Göschl, "A Case Study of Testing a Distributed Internet-System," Software Focus, vol. 1, pp. 15–22, Sep. 2000. [Online]. Available: https://www.researchgate.net/publication/220116945_Testing_software_for_Internet_application