Putting Software Testing Terminology to the Test

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

Abstract—Testing is a pervasive software development activity which is often complicated and expensive (if not simply overlooked). This is in part due to an unstable knowledge base: there is no standard, consistent, and "complete" taxonomy for software testing, inhibiting precise communication. Discrepancies and ambiguities are widespread throughout the literature, and sometimes exist between different parts of the same document! We systematically investigate the current state of software terminology. We 1) identify established standards and prominent testing resources, 2) capture relevant testing terms from these sources, along with their definitions and relationships (both explicit and implied), and 3) build graphs to visualize and analyze this data. Over five hundred test approaches were uncovered, as well as some methods for describing "implied" test approaches. We also build a tool to generate graphs of the relations between test approaches and track ambiguities captured by this tool and manually through the research process. Our results show ten terms are given as synonyms to two (or more) disjoint test approaches and fourteen pairs of test approaches may be synonyms and/or have a child-parent relationship. There is also confusion surrounding functional, recovery, scalability, and performance testing, along with over fifty more "minor" discrepancies. Overall, there is a need for testing terminology to be standardized to make the discussion, analysis, and implementation of various test approaches more coherent. We provide some preliminary advice on how to accomplish this.

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

I. Background

Testing software is complicated, expensive, and often overlooked. In our own project, we wanted to automate the generation of tests. As we did not want to do this in an ad hoc manner, we wanted to get a good grasp on the target domain, testing.

The goal was to uncover the various approaches towards testing, as well as which prerequisites (e.g., input files, oracles) are needed for each. A search for a systematic, rigorous, and "complete" taxonomy for software testing revealed that the existing ones are inadequate:

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

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

Thus we set about closing this gap. We first define the scope of what kinds of "software testing" are of interest (II) and examine the existing literature (III). This reinforced the need for a proper taxonomy! Despite the amount of well understood and organized knowledge (IV), there are still many discrepancies and ambiguities in the literature, either within the same source or between various sources (V). We provide some potential solutions covering some of these discrepancies (VI).

II. Scope

Since our motivation is restricted to testing of code, only the "testing" component of Verification and Validation (V&V) is considered. For example, design reviews (see [4, p. 132]) and documentation reviews (see [p. 132]) are out of scope, as they focus on the V&V of design and documentation respectively. Likewise, ergonomics testing and proximity-based testing (see [5]) are out of scope as they fundamentally involve hardware. Security audits that focus on "an organization's … processes and infrastructure" [5], 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" [6, 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 [7, p. 26] are focused on hardware, this might not be representative of all types (e.g., ML model testing seems domain-specific).

Furthermore, only some aspects of testing approaches are relevant. This mainly manifests as a testing approach that applies to 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" [4, 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 [8, 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" [9, 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" [8, 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 [8, p. 5-15].

A. Static Testing

Sometimes, the term "testing" excludes static testing [10, p. 222], [7, p. 13], restricting it to "dynamic validation" [8, p. 5-1] or "dynamic verification" "in which a system or component is executed" [4, p. 427]. Since "terminology is not uniform among different communities, and some use the term testing to refer to static techniques[1] as well" [8, p. 5-2], the scope of "testing" for the purpose of this project will include both "static testing" and "dynamic testing", as done by [9, p. 17], [11, pp. 8-9], and even a source that explicitly excluded static testing [4, p. 440]!

Static testing seems to be somewhat more ad hoc which makes it less relevant for own goals (automatic generation of tests). In particular, some techniques generate false positives which require human intervention. Nevertheless, 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! For the current purpose of this work, we keep these in-scope at this stage of the analysis.

III. Methodology

A. Source Order

As there is no single authoritative source on software testing terminology, we need to look at many. Unfortunately, this brings to light a variety of discrepancies. Starting from some set of sources, we then use "snowballing" to gather further sources.

0) Trusted textbooks [12, 13, 14]
   - Ad hoc and arbitrary; not systematic
   - Colored maroon,

1) Established standards (such as IEEE, ISO/IEC, and SWEBOK) [4, 5, 8, 9, 15, 16, 17, 18, 19, 20]
   - Standards organizations colored green
   - "Meta-level" commentaries or collections of terminology (often based on these standards), such as [7], colored blue,

2) Other resources: less-formal classifications of terminology (such as [21]), 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 straightforwardly.

B. Procedure

All sources were analyzed in their entirety, except for some in Undefined Terms, to systematically extract terminology. 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", and
- terms that implied testing approaches (see Derived Test Approaches).

When terms had multiple definitions, either the clearest and most concise version was kept, or they were merged to paint a more complete picture. If any discrepancies or ambiguities arose, they were reasonably investigated and always 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) or too broad (such as Attacks), so they will be omitted over the course of analysis.

C. Undefined Terms

This search process led to some testing approaches being mentioned without definition; [9] and [7] in particular introduced many. Once "standard" sources had been exhausted, we devised a strategy to look for sources that explicitly defined these terms, consistent with our snowballing approach. This uncovered new approaches, both in and out of scope (such as EMSEC testing, HTML testing, and aspects of loop testing and orthogonal array testing).

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

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

- 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 categorize software testing approaches in different ways. Reference [9] provides a classfication for different kinds of tests (see Table I). Since this seems to be widely used ("test level" and "test type" in particular) and is useful when focusing on a particular subset of testing, this terminology is used for now. A deeper rationale for a proposed classification will be given during the analysis stage.

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". Examples of these are the classes of combinatorial [20, p. 15] and data flow testing [p. 3] and the family of performance-related testing [23, p. 1187][2], and may also be implied for security testing, a test type that consists of "a number of techniques[3]" [20, 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" [9, p. 15], [20, p. 7]. Due to this, a specific test approach can be derived by combining test approaches from different categories; for some examples of this.

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-driven Techniques: Test techniques are able to "identify test coverage items … and derive corresponding test cases" [9, p. 11] (similar in [4, p. 467]) in a "systematic" way [4, 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" [8, p. 5013], thus maximizing the path coverage (see also [24, Fig. 1]).

[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.

2) Quality-driven Types: Since test types are "focused on specific quality characteristics" [9, p. 15], [20, p. 7], [4, 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" [4, p. 424]. This is supported by reliability and performance testing, which are both examples of test types [9, 20] that are based on their underlying qualities [25, p. 18].

After discussing this further, it was decided that tracking software qualities, in addition to testing approaches, would be worthwhile. 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:

- Nontechnical Requirement: a "requirement affecting product and service acquisition or development that is not a property of the product or service" [4, p. 293]
- Physical Requirement: a "requirement that specifies a physical characteristic that a system or system component must possess" [4, p. 322]

4) Attacks: Since attacks are given as a test practice [9, 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

After gathering all this data, we found many discrepancies and ambiguities. We first report on the more major issues, classified under the rubrics of Synonyms, Parent Relations, Functional Testing, Operational (Acceptance) Testing, Recovery Testing, and Scalability Testing. We then close with some Minor Discrepancies.

A. Synonyms

The same approach often has many names. For example, specification-based testing is also called:

1) Black-Box Testing [9, p. 9], [20, p. 8], [4, p. 431], [8, p. 5-10], [5], [7, p. 46] (without hyphen), [26, p. 344], [14, p. 399]
2) Closed-Box Testing [9, p. 9], [4, p. 431]
3) Functional Testing [4, p. 196], [27, p. 44], [14, p. 399] (implied by [20, p. 129], [4, p. 431])
4) Domain Testing [8, p. 5-10]
5) Input Domain-Based Testing (implied by [15, p. 4-8])

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" [9, p. 10]; described by a test strategy [4, p. 472] and is also used to "pick the particular test case values" [4, p. 465]	black or white box, minimum and maximum boundary value testing [4, p. 465]
(Design)[a] Technique	A "defined" and "systematic" [4, p. 464] "procedure used to create or select a test model, identify test coverage items, and derive corresponding test cases" [9, p. 11] (similar in [4, p. 467]) "that … generate evidence that test item requirements have been met or that defects are present in a test item" [20, p. vii]; "a variety … is typically required to suitably cover any system" [9, p. 33] and is "often selected based on team skills and familiarity, on the format of the test basis", and on expectations [9, p. 23]	equivalence partitioning, boundary value analysis, branch testing [9, 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 [9, p. 12], [20, p. 6] with their "own documentation and resources" [4, p. 469]; more generally, "designat[es] … the coverage and detail" [4, p. 249]	unit/component testing, integration testing, system testing [9, p. 12], [20, p. 6], [4, p. 467]
Practice	A "conceptual framework that can be applied to … [a] test process to facilitate testing" [9, p. 14], [4, p. 471]; more generally, a "specific type of activity that contributes to the execution of a process" [4, p. 331]	scripted testing, exploratory testing, automated testing [9, p. 20]
Type	"Testing that is focused on specific quality characteristics" [9, p. 15], [20, p. 7], [4, p. 473]	security testing, usability testing, performance testing [9, p. 15], [4, p. 473]

[a] "Design technique" is sometimes abbreviated to "technique" [9, p. 11], [5].
[b] "Test level" can also refer to the scope of a test process; for example, "across the whole organization" or only "to specific projects" [9, p. 24].
[c] "Test phase" can be a synonym for "test level" [4, p. 469], [16, p. 9] but can also refer to the "period of time in the software life cycle" when testing occurs [4, p. 470], usually after the implementation phase [4, pp. 420, 509], [22, p. 56].
[d] [8, pp. 5-6 to 5-7], [5], [11, pp. 9, 13].

While some of these synonyms may express mild variations, their core meaning is nevertheless the same. Here we use the terms "specification-based" and "structure-based testing" as 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. Thus, synonyms do not inherently signify a discrepancy.

However, there are cases 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 four out of ten examples that have arisen:

1) Invalid Testing:
   - Error Tolerance Testing [27, p. 45]
   - Negative Testing [5], implied by [20, p. 10]
2) Soak Testing:
   - Endurance Testing [20, p. 39]
   - Reliability Testing [6, Tab. 1, p. 26], [11, Tab. 2]

   Endurance testing is given as a kind of reliability testing by [7, p. 55], but the two are not described as synonyms.
3) User Scenario Testing:
   - Scenario Testing [5]
   - Use Case Testing [27, p. 48]

   "Scenario testing" and "use case testing" are given as synonyms by [5] and [27, pp. 47-49], but listed separately by [9, p. 22] and [20, p. 20]; the latter gives use case testing as a "common form of scenario testing". Since the actor in a use case "can be a user or another system" [20, p. 20], "use case testing" may instead be a child of "user scenario testing" (see Table II).
4) Link Testing:
   - Branch Testing (implied by [20, p. 24])
   - Component Integration Testing [27, p. 45]
   - Integration Testing (implied by [11, p. 13])

There are pairs of synonyms where one is described as a sub-approach of the other, abusing the meaning of "synonym" and causing confusion. The seven (out of fourteen) pairs where each relation has a concrete source are given in Table II.

B. Parent Relations

Parent relations are not immune to difficulties, including self-referential definitions. Performance and usability testing are both given as sub-approaches of themselves [11, Tab. 2], [6, Tab. 1], while performance testing is not described as a sub-approach of usability testing. This would have been more meaningful information to capture.

C. Categories of Testing Approaches

While the IEEE categorization of testing approaches is useful, it is not without its faults. The boundaries between items within a category may be unclear: "although

TABLE II
Pairs of test approaches with both child-parent and synonym relations.

"Child"	→	"Parent"	Child-Parent Source(s)	Synonym Source(s)
All Transitions Testing	→	State Transition Testing	[20, p. 19]	[27, p. 15]
Co-existence Testing	→	Compatibility Testing	[9, p. 3], [19], [20, Tab. A.1]	[20, p. 37]
Fault Tolerance Testing	→	Robustness Testing[a]	[7, p. 56]	[5]
Functional Testing	→	Specification-based Testing	[20, p. 38]	[4, p. 196], [14, p. 399], [27, p. 44]
Orthogonal Array Testing	→	Pairwise Testing	[28, p. 1055]	[8, p. 5-11], [29, p. 473]
Performance Testing	→	Performance-related Testing	[9, p. 22], [20, p. 38]	[23, p. 1187]
Use Case Testing	→	Scenario Testing	[20, p. 20]	[5], [27, pp. 47-49]

[a]Fault tolerance testing may also be a sub-approach of reliability testing [4, p. 375], [8, p. 7-10], which is distinct from robustness testing [7, p. 53].

each technique is defined independently of all others, in practice [sic] some can be used in combination with other techniques" [20, 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, and some approaches are categorized differently by different sources:

1) Experience-based testing is categorized as both a test design technique and a test practice on the same page—twice [9, Fig. 2, p. 34]!
2) The following test approaches are categorized as test techniques by [20, p. 38] and as test types by the sources provided:
   a) Capacity testing [9, p. 22], [16, p. 2],
   b) Endurance testing [16, p. 2],
   c) Load testing [4, p. 253], [5], [9, pp. 5, 20, 22],
   d) Performance testing [9, pp. 7, 22, 26-27], [20, p. 7], and
   e) Stress testing [4, p. 442], [9, pp. 9, 22].
3) "Installability testing" is given as a test type [9, p. 22], [20, p. 38] but is sometimes called a test level as "installation testing" [13, p. 445].
4) Model-based testing is categorized as both a test practice [9, p. 22], [20, p. viii] and a test technique [27, p. 4].
5) Data-driven testing is categorized as both a test practice [9, p. 22] and a test technique [27, p. 43].
6) Although ad hoc testing is sometimes classified as a "technique" [8, p. 5-14], it is one in which "no recognized test design technique is used" [27, p. 42].

There are also instances of inconsistencies between parent and child test approach categorizations. This may indicate they aren't necessarily the same, or that more thought must be given to this method of classification.

D. Functional Testing

"Functional testing" seems to be described in many ways, alongside other, likely related, terms:

- Specification-based Testing is defined as "testing in which the principal test basis is the external inputs and outputs of the test item" [9, 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" [4, p. 196]. Notably, [4] lists both as synonyms of "black-box testing" (pp. 431, 196, respectively). But 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) [5]; the latter references [4, 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 [9, p. 21] (testing "used to check the implementation of functional requirements"). Overall, specification-based testing [9, pp. 2-4, 6-9, 22] and black-box testing (8, p. 5-10; 2, p. 3) are test design techniques used to "derive corresponding test cases" [9, p. 11] (from given "selected inputs and execution conditions" [4, p. 196]).
- Correctness Testing [8, 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" (9, p. 21; 4, p. 196) but groups it with "correctness testing". Since "correctness" is a software quality (4, p. 104; 8, p. 3-13) which is what defines a "test type" [9, p. 15], it seems consistent to label "functional testing" as a "test type" [9, pp. 15, 20, 22]. This is listed separately from "functionality testing" by [7, p. 53].
- Conformance Testing [8, p. 5-7] insures "that the functional specifications are correctly implemented", and can be called "conformance testing" or "functional testing". "Conformance testing" is later defined as used "to verify that the SUT conforms to

standards, rules, specifications, requirements, design, processes, or practices" [8, p. 5-7]. This definition seems to be a superset of testing methods mentioned earlier as the latter includes "standards", "rules", "requirements", "design", "processes", and "practices" as well as "specifications"!
A complicating factor is that "compliance testing" is also (plausibly!) given as a synonym of "conformance testing" [27, p. 43]. 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" [4, p. 93], which seems to describe something different.

- Functional Suitability Testing: Procedure testing is called a "type of functional suitability testing" [9, p. 7], but no definition of that term 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" [19]. 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") [19]. Notably, "functional correctness", which includes precision and accuracy (19; [5]), 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" [4, p. 196] and is said to be a synonym of "functional suitability" [5], 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 [5] and made distinct from "functional testing"; interestingly, security is described as a sub-approach of both non-functional and functionality testing [11, Tab. 2]. This is listed separately from "correctness testing" by [7, p. 53].

E. Operational (Acceptance) Testing

Some sources refer to "operational acceptance testing" (9, p. 22; [5]) while some refer to "operational testing" (8, p. 6-9, in the context of software engineering operations; 30; 4, p. 303; 15, pp. 4-6, 4-9). A distinction is sometimes made [7, 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 [31, 32]) as a type of acceptance testing (9, p. 22; [5]) that focuses on "non-functional" attributes of the system [31].

A summary of 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" [30] or "evaluate a system or component in its operational environment" [4, p. 303], particularly "to determine if operations and/or systems administration staff can accept [it]" [5].

F. Recovery Testing

"Recovery testing" is "testing … aimed at verifying software restart capabilities after a system crash or other disaster" [8, p. 5-9] including "recover[ing] the data directly affected and re-establish[ing] the desired state of the system" [19] (similar in [8, p. 7-10]) so that the system "can perform required functions" [4, p. 370]. It is also called "recoverability testing" [27, p. 47] and potentially "restart & recovery (testing)" [11, Fig. 5]. The following terms, along with "recovery testing" itself [9, p. 22] are all classified as test types, and the relations between them can be found in Figure 1.

- Recoverability Testing: Testing "how well a system or software can recover data during an interruption or failure" [8, p. 7-10] (similar in [19]) and "re-establish the desired state of the system" [19]. Synonym for "recovery testing" in [27, p. 47].
- Disaster/Recovery Testing serves to evaluate if a system can "return to normal operation after a hardware or software failure" [4, 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" [20, 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 "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" [16, 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 [4, 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" [20, p. 37]. This seems to correspond to the definition of "disaster/recovery testing" in [20, 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" [20, 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" [8, p. 5-9] by entering a "backup operational mode in which [these responsibilities] … are assumed by a secondary system" [5]. While not explicitly related to recovery, "failover/recovery testing" also describes the idea of "failover", and [7, p. 56] uses the term "failover and recovery testing", which could be a synonym of both of these terms.

G. Scalability Testing

There were three ambiguities around the term "scalability testing". The relations between these test approaches (and other relevant ones) are shown in Figure 3.

1) Reference [20, p. 39] gives "scalability testing" as a synonym of "capacity testing" while other sources differentiate between the two [7, p. 53], [33, pp. 22-23]
2) Reference [20, p. 39] includes the external modification of the system as part of "scalability", while [19] implies that it is limited to the system itself
3) SWEBOK V4's definition of "scalability testing" [8, p. 5-9] is really a definition of usability testing!

H. Minor Discrepancies

We now outline "minor" discrepancies/ambiguities found in the literature, grouped by the "categories" of sources outlined in Source Order. These discrepancies can then be grouped into degrees of severity:

1) High: Semantic differences between test approaches
2) Medium: Differences in related information about test approaches (such as synonyms or supporting information)
3) Low: Typos, redundant information, or referencing issues

The numbers of these groups of discrepancies are shown in Table III, where a given row corresponds to the number of discrepancies either within/between one or more sources within that category and/or between a source of that category and one of a "more trusted" source (i.e., a source from a category higher up in the table).

1) In Standards [4, 9, 16, 17, 18, 19, 20, 30, 34, 35]:
   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)" [9, 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 [9, 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 ([5]; 4, pp. 73, 237), the definition of "interoperability testing" from [4, p. 238], and the decomposition of "compatibility" into "co-existence" and "interoperability" by [19]!
      - The "interoperability" element of "compatibility testing" is explicitly excluded by [20, p. 37], (incorrectly) implying that "compatibility testing" and "co-existence testing" are synonyms.
      - The definition of "compatibility testing" in [27, 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" [9, p. 5].
   3) Integration, system, and system integration testing are all listed as "common test levels" (9, p. 12; 20, 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 [5] and of system testing by [7, p. 23].
   4) Similarly, component, integration, and component integration testing are all listed in [4], but "component integration testing" is only defined as "testing of groups of related components" [4, 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 [5].
   5) Retesting and regression testing seem to be separated from the rest of the testing approaches [9, p. 23], but it is not clearly detailed why; [36, p. 3] consider regression testing to be a test level.
   6) A component is an "entity with discrete structure … within a system considered at a particular level of analysis" [17] and "the terms module, component, and unit [sic] are often used interchangeably or defined to be subelements of one another in different

TABLE III
Minor Discrepancies

Category \ Severity	High	Medium	Low	Total
Established Standards	2	6	3	11
"Meta-level" Collections	9	5	8	22
Trusted Textbooks	1	0	0	1
Other	6	3	6	15
Total	18	14	17	49

ways depending upon the context" with no standardized relationship [4, p. 82]. This means unit/component/module testing can refer to the testing of both a module and a specific function in a module. However, "component" is sometimes defined differently than "module": "components differ from classical modules for being re-used in different contexts independently of their development" [37, p. 107], so distinguishing the two may be necessary.

7) A typo in [20, Fig. 2] means that "specification-based techniques" is listed twice, when the latter should be "structure-based techniques".
8) (author?) define an "extended entry (decision) table" both as a decision table where the "conditions consist of multiple values rather than simple Booleans" [20, p. 18] and one where "the conditions and actions are generally described but are incomplete" [4, p. 175].
9) (author?) provide a definition for "inspections and audits" [4, 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.
10) (author?) say that "test level" and "test phase" are synonyms, both meaning a "specific instantiation of [a] test sub-process" (4, pp. 469, 470; 16, 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" [9, p. 24], while "test phase" can also refer to the "period of time in the software life cycle" when testing occurs [4, p. 470], usually after the implementation phase [4, pp. 420, 509], [22, p. 56].
11) (author?) use the same definition for "partial correctness" [4, p. 314] and "total correctness" (p. 480).

2) In "Meta-Level" Sources (SWEBOK [8, 15], ISTQB [5], or [7]): resume

1) SWEBOK V4 defines "privacy testing" as testing that "assess[es] the security and privacy of users' personal data to prevent local attacks" [8, 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 [9, 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" [4, p. 17], by "a small, selected group of potential users" [8, p. 5-8], or "in the developer's test environment by roles outside the development organization" [5].
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" [5].
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" [5]. The use of "performance" (or "correctness") in these definitions is 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" [5] seems to focus on how the system is used as opposed to the system itself.
6) The definition of "math testing" given by [5] 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" [5].
8) Thirdly, "par sheet testing" from [5] 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" [38].
9) [5] 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 [39, p. 153]; is this a testing approach or a tool that supports testing?
10) The source cited for the definition of "test type" from [5] 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 [9, p. 34] or "organized around a special focus" [5].
14) While model testing is said to test the object under test, it seems to describe testing the models themselves [7, 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 [7, p. 23].
16) "Hardware-" and "human-in-the-loop testing" have the same acronym: "HIL"[4] [7, p. 23].
17) The same is true for "customer" and "contract(ual) acceptance testing" ("CAT") [7, p. 30].
18) The acronym "SoS" is used but not defined by [7, p. 23].
19) It is ambiguous whether "tool/environment testing"

[4] "HiL" is used for the former by [40, p. 2].

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 [7, p. 25] seems to imply the former.

20) (author?) say that performance and security testing are subtypes of reliability testing [19], but these are all listed separately by [7, p. 53].
21) The distinctions between development testing [4, p. 136], developmental testing [7, p. 30], and developer testing (7, p. 39; 11, p. 11) are unclear and seem miniscule.

3) In Textbooks [12, 13, 14]: resume

1) "Load testing" is defined as using loads "usually between anticipated conditions of low, typical, and peak usage" [9, p. 5], while (author?) says the loads should as large as possible [12, p. 86].

4) In Other Sources: resume

1) (author?) lists "three [backup] location categories: local, offsite and cloud based [sic]" [33, p. 16], but does not define or discuss "offsite backups" (pp. 16-17).
2) (author?) claim that [26] defines "prime path coverage" [41, p. 184], but it doesn't.
3) "State-based" is misspelled by (author?) as "state-base" [27, pp. 13, 15] and "stated-base" (Tab. 1).
4) (author?)'s definition of "boundary value testing" says "See boundary value analysis," but this definition is not present [27].
5) "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" [27, p. 43].
6) (author?) seems to imply that "mutation testing" is a synonym of "back-to-back testing" [27, p. 46], but these are two quite distinct techniques.
7) (author?) also says that the goal of negative testing is "showing that a component or system does not work" [27, 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" [5] (i.e., negative testing) is one way to help test this!
8) "Program testing" is given as a synonym of "component testing" [27, p. 46], although it probably should be a synonym of "system testing" instead.
9) (author?) give "white-", "grey-", and "black-box testing" as synonyms for "module", "integration", and "system testing", respectively [42, 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.
10) Availability testing isn't assigned to a test priority [11, Tab. 2], despite the claim that "the test types[5] have been allocated a slot against the four test priorities" (p. 13); I think usability and/or performance would have made sense.
11) "Visual browser validation" is described as both static and dynamic in the same table [11, Tab. 2], even though they are implied to be orthogonal classifications: "test types can be static or dynamic" (p. 12, emphasis added).
12) (author?) makes a distinction between "transaction verification" and "transaction testing" [11, Tab. 2] and also uses the phrase "transaction flows" (Fig. 5) but doesn't explain them.
13) The phrase "continuous automated testing" [11, p. 11] is redundant since continuous testing is a subcategory of automated testing (9, p. 35, [5]).
14) End-to-end functionality testing is not indicated to be functionality testing [11, Tab. 2].
15) (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" [6, p. 28].

VI. Recommendations

We provide different recommendations for resolving various discrepancies (see Discrepancies and Ambiguities). This was done with the goal of organizing them more logically and making them:

1) Atomic (e.g., disaster/recovery testing seems to have two disjoint definitions)
2) Straightforward (e.g., backup and recovery testing's definition implies the idea of performance, but its name does not; failover/recovery testing, failover and recovery testing, and failover testing are all given separately)
3) Consistent (e.g., backup/recovery testing and failover/recovery testing explicitly exclude an aspect included in its parent disaster/recovery testing)

A. Recovery Testing

The following terms should be used in place of the current terminology to more clearly distinguish between different recovery-related test approaches:

- Recoverability Testing: "Testing … aimed at verifying software restart capabilities after a system crash or other disaster" [8, p. 5-9] including "recover[ing] the data directly affected and re-establish[ing] the desired state of the system" [19] (similar in [8, p. 7-10]) so that the system "can perform required functions" [4, p. 370]. "Recovery testing" will be a synonym, as in [27, p. 47], since it is the more prevalent term throughout various sources, although "recoverability testing" is preferred to indicate that this explicitly

[5] "Each type of test addresses a different risk area" [11, p. 12], which is distinct from the notion of "test type" described in IEEE Testing Terminology.

focuses on the ability to recover, not the performance of recovering.

- Failover Testing: Testing that "validates the SUT's ability to manage heavy loads or unexpected failure to continue typical operations" [8, p. 5-9] by entering a "backup operational mode in which [these responsibilities] … are assumed by a secondary system" [5]. This will replace "failover/recovery testing", since it is more clear, and since this is one way that a system can recover from failure, it will be a subset of "recovery testing".
- Transfer Recovery Testing: Testing to evaluate if, in the case of a failure, "operation of the test item can be transferred to a different operating site and … be transferred back again once the failure has been resolved" [20, p. 37]. This replaces the second definition of "disaster/recovery testing", since the first is just a description of "recovery testing", and could potentially be considered as a kind of failover testing. This may not be intrinsic to the hardware/software (e.g., may be the responsibility of humans/processes).
- Backup Recovery Testing: Testing that determines the ability "to restor[e] from back-up memory in the event of failure" [20, p. 37]. The qualification that this occurs "without transfer[ing] to a different operating site or back-up system" [p. 37] could be made explicit, but this is implied since it is separate from transfer recovery testing and failover testing, respectively.
- Recovery Performance Testing: Testing "how well a system or software can recover … [from] an interruption or failure" [8, p. 7-10] (similar in [19]) "within specified parameters of time, cost, completeness, and accuracy" [16, p. 2]. The distinction between the performance-related elements of recovery testing seemed to be meaningful, but was not captured consistently by the literature. This will be a subset of "performance-related testing" as "recovery testing" is in [9, p. 22]. This could also be extended into testing the performance of specific elements of recovery (e.g., failover performance testing), but this be too fine-grained and may better be captured as an orthogonally derived test approach.

B. Scalability Testing

The ambiguity around scalability testing found in the literature is resolved and/or explained by other sources! [20, p. 39] gives "scalability testing" as a synonym of "capacity testing", defined as the testing of a system's ability to "perform under conditions that may need to be supported in the future" which "may include assessing what level of additional resources (e.g. memory, disk capacity, network bandwidth) will be required to support anticipated future loads". This focus on "the future" is supported by [5], which defines "scalability" as "the degree to which a component or system can be adjusted for changing capacity". In contrast, capacity testing focuses

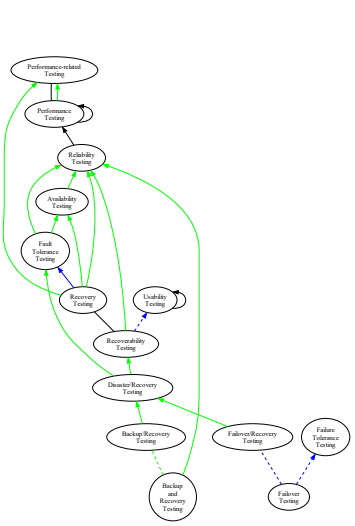

Fig. 1. Relations between "recovery testing" terms.

Fig. 2. Proposed relations between rationalized "recovery testing" terms.

on the system's present state, evaluating the "capability of a product to meet requirements for the maximum limits of a product parameter", such as the number of concurrent users, transaction throughput, or database size [19]. Because of this nuance, it makes more sense to consider these terms separate and not synonyms, as done by [7, p. 53] and [33, pp. 22-23].

Unfortunately, only focusing on future capacity requirements still leaves room for ambiguity. While the previous definition of "scalability testing" includes the external modification of the system, [19] describes it as testing the "capability of a product to handle growing or shrinking workloads or to adapt its capacity to handle variability", implying that this is done by the system itself. The potential reason for this is implied by SWEBOK V4's claim that one objective of elasticity testing is "to evaluate scalability" [8, p. 5-9]: [19]'s notion of "scalability" likely refers more accurately to "elasticity"! This also makes sense in the context of other definitions provided by SWEBOK V4 [8]:

- Scalability: "the software's ability to increase and scale up on its nonfunctional requirements, such as load, number of transactions, and volume of data" [p. 5-5]. Based on this definition, scalability testing is then a subtype of load testing and volume testing, as well as potentially transaction flow testing.
- Elasticity Testing[6]: testing that "assesses the ability of the SUT … to rapidly expand or shrink compute, memory, and storage resources without compromising the capacity to meet peak utilization" [p. 5-9]. Based

[6]While this definition seems correct, it only cites a single source that doesn't contain the words "elasticity" or "elastic"!

on this definition, elasticity testing is then a subtype of memory management testing (with both being a subtype of resource utilization testing) and stress testing.

This distinction is also consistent with how the terms are used in industry: [43] says that scalability is the ability to "increase … performance or efficiency as demand increases over time", while elasticity allows a system to "tackle changes in the workload [that] occur for a short period".

To make things even more confusing, SWEBOK V4 says "scalability testing evaluates the capability to use and learn the system and the user documentation" and "focuses on the system's effectiveness in supporting user tasks and the ability to recover from user errors" [8, p. 5-9]. This seems to define "usability testing" with elements of functional and recovery testing, which is completely separate from the definitions of "scalability", "capacity", and "elasticity testing"! This definition should simply be disregarded, since it is inconsistent with the rest of the literature. The removal of the previous two synonym relations is demonstrated in Figures 3 and 4.

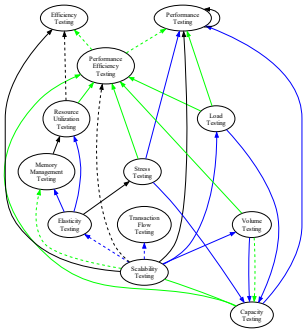

Fig. 3. The current state of the relations between "scalability testing" terms.

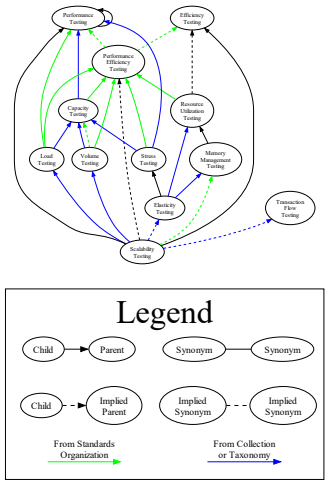

Fig. 4. The proposed relations between "scalability testing" terms.

C. Performance(-related) Testing

"Performance testing" is defined as testing "conducted to evaluate the degree to which a test item accomplishes its designated functions" [4, p. 320], [9, p. 7] (similar in [20, pp. 38-39], [23, p. 1187]). It does this by "measuring the performance metrics" [23, p. 1187] (similar in [5]) (such as the "system's capacity for growth" [6, 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 [8, p. 5-9]). It is performed either …

1) … "within given constraints of time and other resources" [4, p. 320], [9, p. 7] (similar in [23, p. 1187]), or
2) … "under a 'typical' load" [20, 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'" [20, p. 38], along with other approaches like load and capacity testing [9, p. 22]. In contrast, [8, 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 sub-approaches to performance testing instead. This is consistent with how some sources treat "performance testing" and "performance-related testing" as synonyms [8, p. 5-9], [23, p. 1187], as noted in Synonyms. 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" [4, p. 320], [9, p. 7] (similar in [23, p. 1187]), and "performance testing" is a sub-approach of this performed "under a 'typical' load" [20, 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" [4, p. 253], [5], [9, p. 5], [20, p. 39], so it is a child of "performance-related testing" and a parent of "performance testing".

Finally, the "self-loops" mentioned in Parent Relations provide no new information and can be removed. These changes (along with those from Recovery Testing and Scalability Testing made implicitly) result in the relations shown in Figure 5.

Acknowledgment

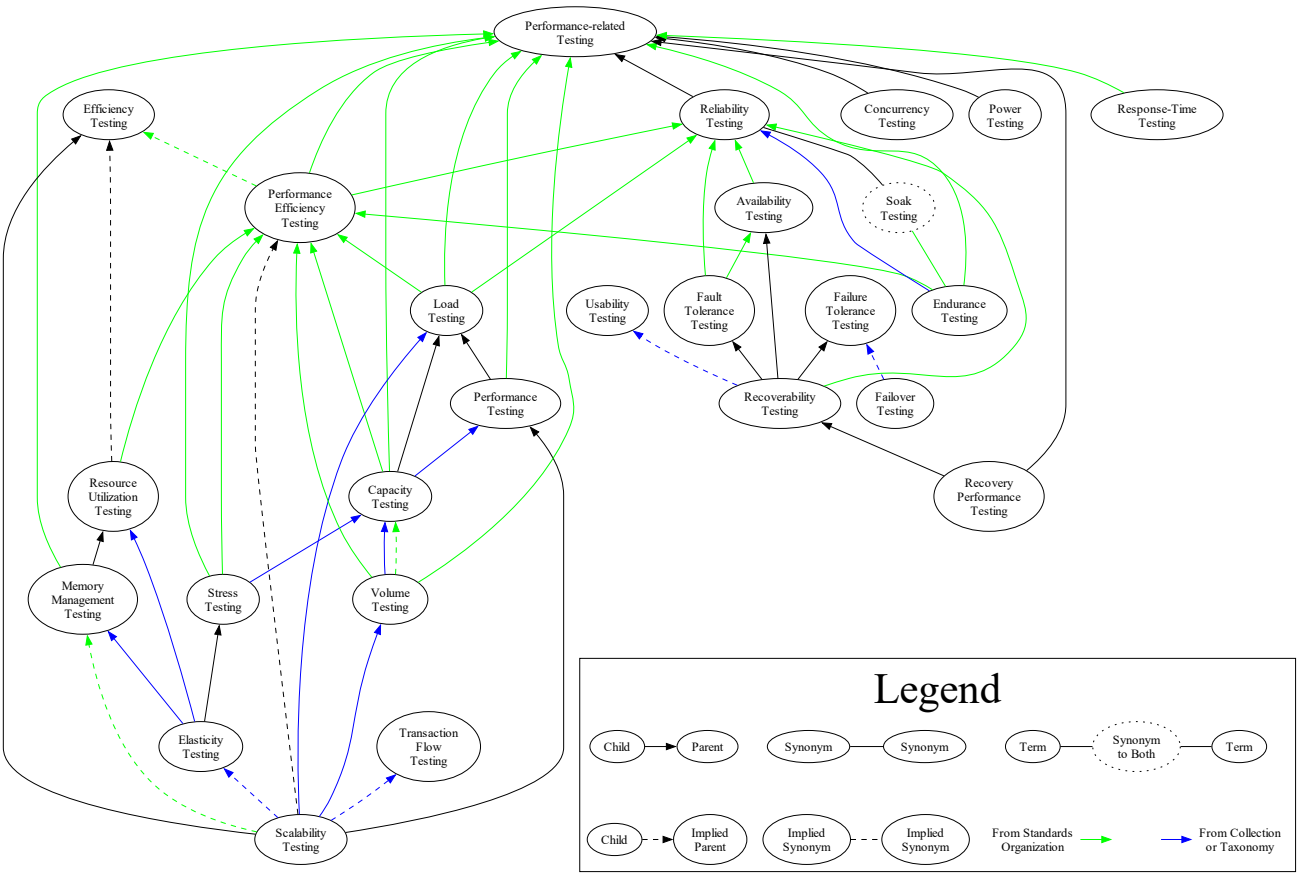

Fig. 5. The proposed relations between "performance-related testing" terms.

References

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

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

[3] 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

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

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

[6] 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

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

[8] 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

[9] 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.

[10] 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

[11] 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
[12] R. Patton, Software Testing, 2nd ed. Indianapolis, IN, USA: Sams Publishing, 2006.
[13] J. Peters and W. Pedrycz, Software Engineering: An Engineering Approach, ser. Worldwide series in computer science. John Wiley & Sons, Ltd., 2000.
[14] H. van Vliet, Software Engineering: Principles and Practice, 2nd ed. Chichester, England: John Wiley & Sons, Ltd., 2000.
[15] 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
[16] 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.
[17] 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
[18] IEEE, "IEEE Standard for System and Software Verification and Validation," IEEE Std 1012-2012 (Revision of IEEE Std 1012-2004), 2012.
[19] 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
[20] 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.
[21] 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.
[22] W. E. Perry, Effective Methods for Software Testing, 3rd ed. Indianapolis, IN, USA: Wiley Publishing, Inc., 2006.
[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] 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.
[26] 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
[27] 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
[28] 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
[29] 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
[30] 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
[31] LambdaTest, "What is Operational Testing: Quick Guide With Examples," 2024. [Online]. Available: https://www.lambdatest.com/learning-hub/operational-testing
[32] 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
[33] 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

[34] 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

[35] ——, "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

[36] 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.

[37] 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

[38] M. Bluejay, "Slot Machine PAR Sheets," May 2024. [Online]. Available: https://easy.vegas/games/slots/par-sheets

[39] 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

[40] 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

[41] 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

[42] 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

[43] P. Pandey, "Scalability vs Elasticity," Feb. 2023. [Online]. Available: https://www.linkedin.com/pulse/scalability-vs-elasticity-pranav-pandey/