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 is a pervasive software development activity that is often complicated and expensive (if not simply overlooked), partly due to the lack of a standardized and consistent taxonomy for software testing. This hinders precise communication, leading to discrepancies and ambiguities across the literature and even within individual documents! In this paper, we systematically examine the current state of software testing 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 implicit—and 3) construct graphs to visualize and analyze this data. Our research uncovered 526 test approaches and four in-scope methods for describing "implied" test approaches. We also build a tool for generating graphs that illustrate relations between test approaches and track ambiguities captured by this tool and manually through the research process. Our results reveal 151 discrepancies or ambiguities, including ten terms used as synonyms to two (or more) disjoint test approaches and 11 pairs of test approaches may either be synonyms or have a parent-child relationship. They also reveal notable confusion surrounding functional, operational acceptance, recovery, and scalability testing. These findings make clear the urgent need for improved testing terminology so that the discussion, analysis and implementation of various test approaches can be more coherent. We provide some preliminary advice on how to achieve this standardization.

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

I. Introduction

Testing software is complicated, expensive, and often overlooked. Improving the productivity of testing and testing research requires a standard language for communication. For example, "complete testing" could mean that the tester has "completed the discovery of every bug in the product…[,] the agreed-upon tests…[, or] the time period assigned to testing", which can lead to miscommunication and, ultimately, the tester getting "blamed for not doing … [their] job" [1, p. 7]. Unfortunately, a search for a systematic, rigorous, and "complete" taxonomy for software testing revealed that the existing ones are inadequate:

- Tebes et al. [2] focus on parts of the testing process (e.g., test goal, testable entity),
- Souza et al. [3] prioritize organizing testing approaches over defining them, and
- Unterkalmsteiner et al. [4] provide a foundation for classification but not how it applies to software testing terminology.

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

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 and documentation reviews (see [5, pp. 132, 144], respectively) are out of scope, as they focus on the V&V of design and documentation, respectively. Likewise, ergonomics testing and proximity-based testing (see [6]) are out of scope as they fundamentally involve hardware.

Furthermore, only some aspects of some testing approaches are relevant. This mainly manifests as a testing approach that applies to both the V&V itself and to 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 (out of scope), "and estimat[e] the number of faults remaining" [5, p. 165], which helps verify the actual code (in scope).
2) Fault injection testing, where "faults are artificially introduced into the SUT [System Under Test]", can be used to evaluate the effectiveness of a test suite [7, p. 5-18], which is a part of V&V of the V&V itself (out of scope), or "to test the robustness of the system in the event of internal and external failures" [8, p. 42], which helps verify the actual code (in scope).
3) "Mutation [t]esting was originally conceived as a technique to evaluate test suites in which a mutant

Funding for this work was provided by the Ontario Graduate Scholarship and McMaster University.

is a slightly modified version of the SUT" [7, p. 5-15], which is in the realm of V&V of the V&V itself (out of scope). However, it "can also be categorized as a structure-based technique" and can be used to assist fuzz and metamorphic testing [7, p. 5-15] (in scope).

III. Methodology

A. Sources

As there is no single authoritative source on software testing terminology, we need to look at many to see how various terms are used in practice. Starting from some set of sources, we then use "snowball sampling" (a "method of … sample selection … used to locate hidden populations" [9]) to gather further sources (see Section III-D). Sources with a similar degree of "trustworthiness" are grouped into categories; sources that are more "trustworthy":

1) have gone through a peer-review process,
2) are written by numerous, well-respected authors,
3) are informed by many sources, and
4) are accepted and used in the field of software.

We therefore create the following categories, given in order of descending trustworthiness: Established Standards, "Meta-level" Collections, Textbooks, and Papers and Other Documents. Each category is given a unique colour to better track how their information appears in relevant graphs (see Figures 2 to 6). A summary of how many sources comprise each category is given in Figure 1.

1) Established Standards: [5], [8], [10]–[21]

- Colored green
- Information on software development and testing from standards bodies (such as IEEE and ISO)
- Written by reputable organizations for use in software engineering; for example, "the purpose of the ISO/IEC/IEEE 29119 series is to define an internationally agreed set of standards for software testing that can be used by any organization when performing any form of software testing" [8, p. vii]

2) "Meta-level" Collections: [6], [7], [22]–[24]

- Colored blue
- Collections of relevant terminology (such as ISTQB's glossary, the SWEBOK Guide, and Doğan et al.'s literature review [23])
- Built up from various sources, including Established Standards, and often written by a large organization (such as ISTQB); the SWEBOK Guide is "proposed as a suitable foundation for government licensing, for the regulation of software engineers, and for the development of university curricula in software engineering" [1, p. xix]

3) Textbooks: [1], [25]–[30]

- Colored maroon
- Textbooks trusted at McMaster [26], [29], [30] were the original (albeit ad hoc and arbitrary) starting point
- Written by smaller sets of authors, but with a formal review process before publication
- Used as resources for teaching software engineering and may be used as guides in industry

4) Papers and Other Documents: [3], [31]–[60]

- Colored black
- Mainly consists of academic papers: journal articles, conference papers, reports [47], [56], [57], and a thesis [31]
- Written by much smaller sets of authors with unknown peer review processes
- Much less widespread than other categories of sources
- Many of these sources were investigated to "fill in" missing definitions (see Section III-D)
- Also included (for brevity) are some less-than-academic sources to investigate how terms are used in practice, such as websites [32], [33] and a booklet [34]

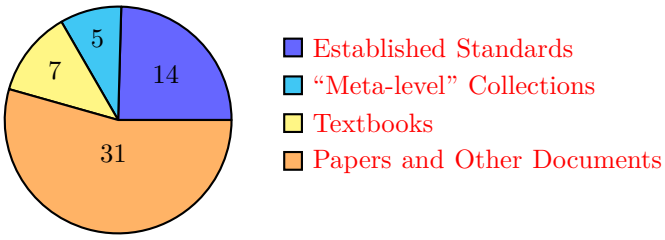

Fig. 1. Summary of how many sources comprise each source category.

B. Terminology

This research was intended to describe the current state of testing terminology instead of prematurely applying any classifications to reduce bias. Therefore, the notions of Categories of Testing Approaches and Parent-Child Relations arose naturally from the literature. Even though these are "results" of this research, they are defined here for clarity since they are used throughout this thesis. We also define the notion of Rigidity.

1) Categories of Testing Approaches: Different sources categorize software testing approaches in different ways. ISO/IEC and IEEE [8] provide a classification 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.

2) Parent-Child Relations: Many test approaches are multi-faceted and can be "specialized" into others, such as Performance(-related) Testing. These "specializations" will be referred to as "children" or "sub-approaches" of the multi-faceted "parent". This nomenclature also extends to other Categories of Testing Approaches from Table I, such as "sub-type".

3) Rigidity: Since there is a considerable degree of nuance introduced by the use of natural language, not all discrepancies are equal! To capture this nuance and

TABLE I
IEEE Testing Terminology

Term	Definition	Examples
Approach	A "high-level test implementation choice, typically made as part of the test strategy design activity" that includes "test level, test type, test technique, test practice and the form of static testing to be used" [8, p. 10]; described by a test strategy [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" [8, 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" [10, p. vii]; "a variety … is typically required to suitably cover any system" [8, p. 33] and is "often selected based on team skills and familiarity, on the format of the test basis", and on expectations [8, p. 23]	equivalence partitioning, boundary value analysis, branch testing [8, p. 11]
Level[b] (sometimes "Phase"[c] or "Stage"[d])	A stage of testing "typically associated with the achievement of particular objectives and used to treat particular risks", each performed in sequence [8, p. 12], [10, 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 [5, p. 467], [8, p. 12], [10, p. 6]
Practice	A "conceptual framework that can be applied to … [a] test process to facilitate testing" [5, p. 471], [8, p. 14]; more generally, a "specific type of activity that contributes to the execution of a process" [5, p. 331]	scripted testing, exploratory testing, automated testing [8, p. 20]
Type	"Testing that is focused on specific quality characteristics" [5, p. 473], [8, p. 15], [10, p. 7]	security testing, usability testing, performance testing [5, p. 473], [8, p. 15]

[a] "Design technique" is sometimes abbreviated to "technique" [6], [8, p. 11].
[b] "Test level" can also refer to the scope of a test process; for example, "across the whole organization" or only "to specific projects" [8, p. 24].
[c] "Test phase" can be a synonym for "test level" [5, p. 469], [13, 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], [27, p. 56].
[d] Used by [6], [7, pp. 5-6 to 5-7], [56, pp. 9, 13].

provide a more complete picture, we make a distinction between explicit and implicit discrepancies, such as in Tables II and III. A piece of information is "implicit" if:

- it is not directly given by a source but seems to be implied, and/or
- it is only true some of the time (e.g., under certain conditions).

Discrepancies based on implicit information are themselves implicit. These are automatically detected when generating graphs and analyzing discrepancies by looking for indicators of uncertainty, such as question marks, " (Testing)" (which indicates that a test approach isn't explicitly denoted as such), and the keywords "implied", "inferred", "can be", "ideally", "usually", "most", "likely", "often", "if", and "although" (see the relevant source code). These words were used when creating the glossaries to capture varying degrees of nuance, such as when a test approach "can be" a child of another or is a synonym of another "most of the time", but isn't always. As an example, Table IV contains relations that are explicit, implicit, and both; implicit relations are marked by the phrase "implied by".

C. Procedure

To track terminology used in the literature, we build a glossary of test approaches, including the term itself, its definition, and any synonyms or parents. Any additional notes, such as questions or sources to investigate further, are also recorded. Approach categorizations, such as those found in Table I and some outliers (e.g., "artifact"), are tracked for future investigation.

Most relevant sources are analyzed in their entirety to systematically extract terminology, with the exception of some sources that were only partially investigated. This is the case for sources chosen for a specific area of interest or based on a test approach that was determined to be out-of-scope, such as some sources given in Section III-D. Heuristics are used to guide this process, by investigating:

- glossaries and lists of terms,
- testing-related terms (e.g., terms containing "test(ing)", "validation", or "verification"),
- terms that had emerged as part of already-discovered testing approaches, especially those that were ambiguous or prompted further discussion (e.g., terms containing "performance", "recovery", "component", "bottom-up", or "configuration"), and
- terms that implied testing approaches.

When terms have multiple definitions, either the clearest and most concise version is kept, or they are merged to paint a more complete picture. If any discrepancies or ambiguities arise, they are reasonably investigated and always documented. If a testing approach is mentioned but not defined, it is added to the glossary to indicate it should be investigated further (see Section III-D). A similar methodology is used for tracking software qualities,

albeit in a separate document.

During the first pass of data collection, all software-testing-focused terms are included. Some of them are less applicable to test case automation or too broad, so they will be omitted over the course of analysis.

D. Undefined Terms

The search process led to some testing approaches being mentioned without definition; [8] and [22] in particular introduced many. Once Established Standards had been exhausted, we devised a strategy to look for sources that explicitly define these terms, consistent with our snowballing approach. This uncovers new approaches, both in and out of scope (such as EManations SECurity (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 442 to 526 and the number of undefined terms from 156 to 172 (the assumption can be made that about 81% of added terms also included a definition):

- Assertion Checking: [40], [51], [52]
- Loop Testing[1]: [38], [43], [44], [54]
- EMSEC Testing: [65], [66]
- Asynchronous Testing: [58]
- Performance(-related) Testing: [36]
- Web Application Testing: [23], [47]
  - HTML Testing: [46], [55], [57]
  - Document Object Model (DOM) Testing: [37]
- Sandwich Testing: [35], [53]
- Orthogonal Array Testing[2]: [42], [60]
- Backup Testing: [31]

E. Tools

To better visualize how test approaches relate to each other, we then developed a tool to automatically generate graphs of these relations. All parent-child relations are graphed, as well as synonym relations where either:

1) both synonyms have their own rows in the glossary, or
2) a term is a synonym to more than one term with its own row in the glossary.

Since these graphs tend to be large, it is useful to focus on specific subsets of them. These can be generated based on a given subset of approaches to include, such as those pertaining to recovery or scability, as shown in Figures 2 and 4, respectively (albeit with manually created legends). By specifying sets of approaches and relations to add or remove, these generated graphs can be updated in accordance with our Recommendations, as shown in Figures 3 and 5, respectively. These modifications can also be inherited, as in Figure 6, which was generated based on the modifications from Figures 3 and 5 and then manually tweaked.

[1]References [16] and [15] were used as reference for terms but not fully investigated, [61] and [62] were added as potentially in scope, and [63] and [64] were added as out-of-scope examples.

[2]References [45] and [48] were added as out-of-scope examples.

IV. Discrepancies and Ambiguities

After gathering all these data[3], we found many discrepancies. To better understand and analyze these discrepancies, each one is given a "class" and a "category". Discrepancy Classes describe how a discrepancy manifests syntactically; examples include Mistakes and Omissions. On the other hand, Discrepancy Categories describe the knowledge domain in which a discrepancy manifests semantically; examples include Synonym Relation Discrepancies and Parent-Child Relation Discrepancies. Within these sections, discrepancies are sorted based on their source category.

A summary of how many discrepancies there are by class and by category is shown in Tables II and III, respectively, where a given row corresponds to the number of discrepancies either within that source category and/or with a "more trusted" one (i.e., a previous row in the table). The numbers of (Exp)licit and (Imp)licit (see Section III-B3) discrepancies are also presented in these tables. Note that all manually tracked discrepancies appear in both Sections IV-A and IV-B, while those automatically uncovered based on semantics are only listed in their appropriate category for clarity (but still contribute to the counts in Table II).

Additionally, specific subsets of testing presented significant issues. These are given in their specific sections to keep related information together. These discrepancies still count towards one or more of the categories of discrepancies listed above. The "problem" subsets of testing include Functional Testing, Recovery Testing, Scalability Testing, and Compatibility Testing.

A. Discrepancy Classes

The following sections list observed discrepancies grouped by how the discrepancy manifests. These include Mistakes, Omissions, Contradictions, Ambiguities, Overlaps, and Redundancies[4].

1) Mistakes: The following are cases where information is incorrect; this includes cases Terminology included that should not have been, untrue claims about Citations, and simple typos:

- Since keyword-driven testing can be used for automated or manual testing [12, pp. 4, 6], the claim that "test cases can be either manual test cases or keyword test cases" [p. 6] is incorrect.
- The terms "acceleration tolerance testing" and "acoustic tolerance testing" seem to only refer to software testing in [22, p. 56]; elsewhere, they seem to refer to testing the acoustic tolerance of rats [67]

[3]Available in ApproachGlossary.csv and QualityGlossary.csv at https://github.com/samm82/TestGen-Thesis.

[4]Section omitted for brevity.

TABLE II
Breakdown of identified Discrepancy Classes by Source Category.

Source Category	Mistakes		Omissions		Contradictions		Ambiguities		Overlaps		Redundancies[a]		Total
	Exp	Imp	Exp	Imp	Exp	Imp	Exp	Imp	Exp	Imp	Exp	Imp	
Established Standards	4	1	2	0	14	4	1	0	5	0	0	0	31
"Meta-level" Collections	10	0	1	0	20	9	8	1	5	1	1	0	56
Textbooks	1	0	0	0	12	2	1	0	1	0	0	0	17
Papers and Other Documents	7	1	4	0	18	5	5	2	2	1	2	0	47
Total	22	2	7	0	64	20	15	3	13	2	3	0	151

[a]Section omitted for brevity.

TABLE III
Breakdown of identified Discrepancy Categories by Source Category.

Source Category	Synonyms		Parents		Categories		Definitions		Terminology		Citations		Total
	Exp	Imp	Exp	Imp	Exp	Imp	Exp	Imp	Exp	Imp	Exp	Imp	
Established Standards	1	2	5	1	10	1	7	1	3	0	0	0	31
"Meta-level" Collections	7	3	5	3	5	5	14	0	10	0	4	0	56
Textbooks	11	0	1	1	1	0	2	1	0	0	0	0	17
Papers and Other Documents	13	7	8	0	6	1	5	0	6	1	0	0	47
Total	32	12	19	5	22	7	28	2	19	1	4	0	151

or the acceleration tolerance of astronauts [68, p. 11], aviators [69, pp. 27, 42], or catalysts [70, p. 1463], which don't exactly seem relevant…

- Reference [29] claims that "structural testing subsumes white box testing" but they seem to describe the same thing: it says "structure tests are aimed at exercising the internal logic of a software system" and "in white box testing …, using detailed knowledge of code, one creates a battery of tests in such a way that they exercise all components of the code (say, statements, branches, paths)" on the same page [29, p. 447]!
- Reference [47, p. 46] says that the goal of negative testing is "showing that a component or system does not work" 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!

2) Omissions: The following are cases where information (usually Definitions) should have been included but was not:

- Integration testing, system testing, and system integration testing are all listed as "common test levels" [8, p. 12], [10, 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 [22, p. 23].
- Similarly, component testing, integration testing, 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? As above, it is listed as a child of integration testing by [6].
- Reference [47, p. 42] says "See boundary value analysis," for the glossary entry of "boundary value testing" but does not provide this definition.

3) Contradictions: The following are cases where multiple sources of information (sometimes within the same document!) disagree; note that cases where all sources of information are incorrect are considered contradictions and not Mistakes, since this would require analysis that has not been performed yet:

- ISO/IEC and IEEE categorize experience-based testing as both a test design technique and a test practice on the same page—twice [8, Fig. 2, p. 34]!
- The following test approaches are categorized as test techniques by [10, p. 38] and as test types by the sources provided:
  1) Capacity testing [8, p. 22], [13, p. 2],
  2) Endurance testing [13, p. 2],
  3) Load testing [5, p. 253], [6], [8, pp. 5, 20, 22],
  4) Performance testing [8, pp. 7, 22, 26-27], [10, p. 7], and
  5) Stress testing [5, p. 442], [8, pp. 9, 22].
- 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. However, "component" is sometimes defined differently than "module": "components differ from classical modules for being re-used in different contexts independently of their development" [50, p. 107], so distinguishing the two may be necessary.
- Performance testing and security testing are given as subtypes of reliability testing by [17], but these are all listed separately by [22, p. 53].
- There is disagreement on the structure of tours: they can either be quite general [8, p. 34] or "organized around a special focus" [6].
- 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" [7, p. 5-8], or "in the developer's test environment by roles outside the development organization" [6].
- "Installability testing" is given as a test type [8, p. 22], [10, p. 38] but is sometimes called a test level as "installation testing" [29, p. 445].
- "Load testing" is performed either using loads that are "between anticipated conditions of low, typical, and peak usage" [8, p. 5] or that are as large as possible [26, p. 86].
- State testing requires that "all states in the state model … [are] 'visited'" in [10, p. 19] which is only one of its possible criteria in [26, pp. 82-83].
- Likewise, "walkthroughs" and "structured walkthroughs" are given as synonyms by [6] but [29, p. 484] implies that they are different, saying a more structured walkthrough may have specific roles.
- Reference [26, p. 92] says that reviews are "the process[es] under which static white-box testing is performed" but correctness proofs are given as another example by [30, pp. 418-419].
- Model-based testing is categorized as both a test practice [8, p. 22], [10, p. viii] and a test technique [47, p. 4].
- Data-driven testing is categorized as both a test practice [8, p. 22] and a test technique [47, p. 43].

4) Ambiguities: The following are cases where information (usually Definitions or distinctions between Terminology) is unclear:

- The distinctions between development testing [5, p. 136], developmental testing [22, p. 30], and developer testing [22, p. 39], [56, p. 11] are unclear and seem miniscule.
- Reference [6] defines "Machine Learning (ML) model testing" and "ML functional performance" 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". The use of "performance" (or "correctness") in these definitions is at best ambiguous and at worst incorrect.
- Reference [6] claims that code inspections are related to peer reviews but [26, pp. 94-95] makes them quite distinct.

5) Overlaps: The following are cases where information overlaps, such as nonatomic Definitions and Terminology:

- The SWEBOK Guide V4 defines "privacy testing" as testing that "assess[es] the security and privacy of users' personal data to prevent local attacks" [7, p. 5-10]; this seems to overlap (both in scope and name) with the definition of "security testing" in [8, p. 7]: testing "conducted to evaluate the degree to which a test item, and associated data and information, [sic] are protected so that" only "authorized persons or systems" can use them as intended.
- "Orthogonal array testing" [7, pp. 5-1, 5-11] and "operational acceptance testing" [22, p. 30] have the same acronym ("OAT").

B. Discrepancy Categories

The following sections list observed discrepancies grouped by what area the discrepancy manifests in. These include Synonym Relation Discrepancies, Parent-Child Relation Discrepancies, Test Approach Category Discrepancies, Definition Discrepancies, Terminology Discrepancies, and Citation Discrepancies.

1) Synonym Relation Discrepancies: The same approach often has many names. For example, specification-based testing is also called:

1) Black-Box Testing [5, p. 431], [6], [7, p. 5-10], [8, p. 9], [30, p. 399], [10, p. 8], [41, p. 344]
2) Closed-Box Testing [5, p. 431], [8, p. 9]
3) Functional Testing[5] [5, p. 196], [30, p. 399], [47, p. 44]
4) Domain Testing [7, p. 5-10]

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. Unfortunately, there are many instances of incorrect or ambiguous synonyms, such as the following:

- 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

[5]This may be an outlier; see Section IV-C1.

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. However, "component" is sometimes defined differently than "module": "components differ from classical modules for being re-used in different contexts independently of their development" [50, p. 107], so distinguishing the two may be necessary.
- Reference [6] claims that code inspections are related to peer reviews but [26, pp. 94-95] makes them quite distinct.
- Likewise, "walkthroughs" and "structured walkthroughs" are given as synonyms by [6] but [29, p. 484] implies that they are different, saying a more structured walkthrough may have specific roles.
- Reference [29] claims that "structural testing subsumes white box testing" but they seem to describe the same thing: it says "structure tests are aimed at exercising the internal logic of a software system" and "in white box testing ..., using detailed knowledge of code, one creates a battery of tests in such a way that they exercise all components of the code (say, statements, branches, paths)" on the same page [29, p. 447]!

There are also 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. Ten of these cases were identified through automatic analysis of the generated graphs. The following four are the most prominent examples:

1) Invalid Testing:
   - Error Tolerance Testing [47, p. 45]
   - Negative Testing [6] (implied by [10, p. 10])
2) Soak Testing:
   - Endurance Testing [10, p. 39]
   - Reliability Testing[6] [57, Tab. 1, p. 26], [56, Tab. 2]
3) User Scenario Testing:
   - Scenario Testing [6]
   - Use Case Testing[7] [47, p. 48] (although "an actor can be a user or another system" [10, p. 20])
4) Link Testing:
   - Branch Testing (implied by [10, p. 24])
   - Component Integration Testing [47, p. 45]
   - Integration Testing (implied by [56, p. 13])

2) Parent-Child Relation Discrepancies: Parent-Child Relations are also not immune to difficulties; for example, performance testing and security testing are given as subtypes of reliability testing by [17], but these are all listed separately by [22, p. 53].

Additionally, some self-referential definitions imply that a test approach is a parent of itself. Since these are by nature self-contained within a given source, these are counted once as explicit discrepancies within their sources in ??. For example, performance and usability testing are both given as sub-approaches of themselves [56, Tab. 2], [57, Tab. 1].

There are also pairs of synonyms where one is described as a sub-approach of the other, abusing the meaning of "synonym" and causing confusion. We identified 11 of these pairs through automatic analysis of the generated graphs, with the most prominent given in Table IV.

3) Test Approach Category Discrepancies: While the IEEE categorization of testing approaches described in Table I is useful, it is not without its faults. 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" [10, 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:

- ISO/IEC and IEEE categorize experience-based testing as both a test design technique and a test practice on the same page—twice [8, Fig. 2, p. 34]!
- The following test approaches are categorized as test techniques by [10, p. 38] and as test types by the sources provided:
  1) Capacity testing [8, p. 22], [13, p. 2],
  2) Endurance testing [13, p. 2],
  3) Load testing [5, p. 253], [6], [8, pp. 5, 20, 22],
  4) Performance testing [8, pp. 7, 22, 26-27], [10, p. 7], and
  5) Stress testing [5, p. 442], [8, pp. 9, 22].
- Since keyword-driven testing can be used for automated or manual testing [12, pp. 4, 6], the claim that "test cases can be either manual test cases or keyword test cases" [p. 6] is incorrect.
- "Installability testing" is given as a test type [8, p. 22], [10, p. 38] but is sometimes called a test level as "installation testing" [29, p. 445].
- Model-based testing is categorized as both a test practice [8, p. 22], [10, p. viii] and a test technique [47, p. 4].
- Data-driven testing is categorized as both a test practice [8, p. 22] and a test technique [47, p. 43].

There are also instances of inconsistencies between parent and child test approach categorizations. This may

[6]Endurance testing is given as a kind of reliability testing by [22, p. 55], although the terms are not synonyms.

[7]"Scenario testing" and "use case testing" are given as synonyms by [6] and [47, pp. 47-49] but listed separately by [8, p. 22], which also gives "use case testing" as a "common form of scenario testing" [10, p. 20]. This implies that "use case testing" may instead be a child of "user scenario testing" (see Table IV).

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

"Child"	→	"Parent"	Parent-Child Source(s)	Synonym Source(s)
All Transitions Testing	→	State Transition Testing	[10, p. 19]	[47, p. 15]
Co-existence Testing	→	Compatibility Testing	[8, p. 3], [17], [10, Tab. A.1]	[10, p. 37]
Fault Tolerance Testing	→	Robustness Testing[a]	[22, p. 56]	[6]
Functional Testing	→	Specification-based Testing[b]	[10, p. 38]	[5, p. 196], [30, p. 399], [47, p. 44]
Orthogonal Array Testing	→	Pairwise Testing	[60, p. 1055]	[7, p. 5-11], [42, p. 473]
Performance Testing	→	Performance-related Testing	[8, p. 22], [10, p. 38]	[36, p. 1187]
Use Case Testing	→	Scenario Testing	[10, p. 20]	[6], [47, pp. 47-49]

[a]Fault tolerance testing may also be a sub-approach of reliability testing [5, p. 375], [7, p. 7-10], which is distinct from robustness testing [22, p. 53].
[b]See Section IV-C1.

indicate they aren't necessarily the same, or that more thought must be given to this method of classification.

4) Definition Discrepancies: Perhaps the most interesting category for those seeking to understand how to apply a given test approach, there are many discrepancies between how test approaches, as well as supporting terms, are defined:

- Integration testing, system testing, and system integration testing are all listed as "common test levels" [8, p. 12], [10, 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 [22, p. 23].
- Similarly, component testing, integration testing, 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? As above, it is listed as a child of integration testing by [6].
- The SWEBOK Guide V4 defines "privacy testing" as testing that "assess[es] the security and privacy of users' personal data to prevent local attacks" [7, p. 5-10]; this seems to overlap (both in scope and name) with the definition of "security testing" in [8, p. 7]: testing "conducted to evaluate the degree to which a test item, and associated data and information, [sic] are protected so that" only "authorized persons or systems" can use them as intended.
- There is disagreement on the structure of tours: they can either be quite general [8, p. 34] or "organized around a special focus" [6].
- 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" [7, p. 5-8], or "in the developer's test environment by roles outside the development organization" [6].
- Reference [6] defines "Machine Learning (ML) model testing" and "ML functional performance" 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". The use of "performance" (or "correctness") in these definitions is at best ambiguous and at worst incorrect.
- "Load testing" is performed either using loads that are "between anticipated conditions of low, typical, and peak usage" [8, p. 5] or that are as large as possible [26, p. 86].
- State testing requires that "all states in the state model … [are] 'visited'" in [10, p. 19] which is only one of its possible criteria in [26, pp. 82-83].
- Reference [26, p. 92] says that reviews are "the process[es] under which static white-box testing is performed" but correctness proofs are given as another example by [30, pp. 418-419].
- Reference [47, p. 46] says that the goal of negative testing is "showing that a component or system does not work" 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!
- Reference [47, p. 42] says "See boundary value analysis," for the glossary entry of "boundary value testing" but does not provide this definition.

5) Terminology Discrepancies: While some discrepancies exist because the definition of a term is wrong, others exist because term's name or label is wrong! This could be considered a "sister" category of Definition Discrepancies, but these discrepancies seemed different enough to merit their own category. The following are examples of these discrepancies:

- The distinctions between development testing [5, p. 136], developmental testing [22, p. 30], and developer testing [22, p. 39], [56, p. 11] are unclear and

seem miniscule.

- The terms "acceleration tolerance testing" and "acoustic tolerance testing" seem to only refer to software testing in [22, p. 56]; elsewhere, they seem to refer to testing the acoustic tolerance of rats [67] or the acceleration tolerance of astronauts [68, p. 11], aviators [69, pp. 27, 42], or catalysts [70, p. 1463], which don't exactly seem relevant…
- "Orthogonal array testing" [7, pp. 5-1, 5-11] and "operational acceptance testing" [22, p. 30] have the same acronym ("OAT").

6) Citation Discrepancies: Sometimes a document cites another for a piece of information that does not appear! For example, [23, p. 184] claims that [41] defines "prime path coverage", but it does not.

C. Functional Testing

"Functional testing" is described alongside many other, likely related, terms. This leads to confusion about what distinguishes these terms, as shown by the following five:

1) Specification-based Testing: This is defined as "testing in which the principal test basis is the external inputs and outputs of the test item" [8, p. 9]. This 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], despite them sometimes being defined separately. For example, the International Software Testing Qualifications Board (ISTQB) defines "specification-based testing" as "testing based on an analysis of the specification of the component or system" and "functional testing" as "testing performed to evaluate if a component or system satisfies functional requirements" [6]. Overall, specification-based testing [8, pp. 2-4, 6-9, 22] is a test design technique used to "derive corresponding test cases" [8, p. 11] from "selected inputs and execution conditions" [5, p. 196].

2) Correctness Testing: The SWEBOK Guide V4 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" [7, p. 5-7]; this mirrors previous definitions of "functional testing" [5, p. 196], [8, p. 21] but groups it with "correctness testing". Since "correctness" is a software quality [5, p. 104], [7, p. 3-13] which is what defines a "test type" [8, p. 15], it seems consistent to label "functional testing" as a "test type" [8, pp. 15, 20, 22], [10, pp. 7, 38, Tab. A.1], [12, p. 4]. However, this conflicts with its categorization as a "technique" if considered a synonym of Specification-based Testing. Additionally, "correctness testing" is listed separately from "functionality testing" by [22, p. 53].

3) Conformance Testing: Testing that ensures "that the functional specifications are correctly implemented", and can be called "conformance testing" or "functional testing" [7, p. 5-7]. "Conformance testing" is later defined as testing used "to verify that the SUT conforms to standards, rules, specifications, requirements, design, processes, or practices" [7, 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 in addition to specifications!

A complicating factor is that "compliance testing" is also (plausibly) given as a synonym of "conformance testing" [47, 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" [5, p. 93], which seems to describe something different.

4) Functional Suitability Testing: Procedure testing is called a "type of functional suitability testing" [8, 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" [17]. This seems to align with the definition of "functional testing" as related to "black-box/specification-based testing". "Functional correctness", a child of "functional suitability", is the "capability of a product to provide accurate results when used by intended users" [17] and seems to align with the quality/ies that would be tested by "correctness" testing.

5) 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 a synonym of "functional completeness" based on [17], which would make "functional suitability" a sub-approach. Its associated test type is implied to be a sub-approach of build verification testing [6] and made distinct from "functional testing" [56, Tab. 2]. "Functionality testing" is listed separately from "correctness testing" by [22, p. 53].

D. Recovery Testing

"Recovery testing" is "testing … aimed at verifying software restart capabilities after a system crash or other disaster" [7, p. 5-9] including "recover[ing] the data directly affected and re-establish[ing] the desired state of the system" [17] (similar in [7, p. 7-10]) so that the system "can perform required functions" [5, p. 370]. It is also called "recoverability testing" [47, p. 47] and potentially "restart & recovery (testing)" [56, Fig. 5]. The following terms, along with "recovery testing" itself [8, p. 22] are all classified as test types, and the relations between them can be found in Figure 2.

- Recoverability Testing: Testing "how well a system or software can recover data during an interruption or failure" [7, p. 7-10] (similar in [17]) and "re-establish the desired state of the system" [17]. Synonym for "recovery testing" in [47, p. 47].

- Disaster/Recovery Testing serves 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" [10, 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" [13, 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" [10, p. 37]. This seems to correspond to the definition of "disaster/recovery testing" in [10, 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]. It also seems to overlap with "backup and recovery testing", which adds confusion.
- 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" [10, 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" [7, 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 [22, p. 56] uses the term "failover and recovery testing", which could be a synonym of both of these terms.

E. Scalability Testing

There were three ambiguities around the term "scalability testing", listed below. The relations between these test approaches (and other relevant ones) are shown in Figure 4.

1) ISO/IEC and IEEE give "scalability testing" as a synonym of "capacity testing" [10, p. 39] while other sources differentiate between the two [22, p. 53], [31, pp. 22-23]
2) ISO/IEC and IEEE give the external modification of the system as part of "scalability" [10, p. 39], while [17] implies that it is limited to the system itself
3) The SWEBOK Guide V4's definition of "scalability testing" [7, p. 5-9] is really a definition of usability testing!

F. Compatibility Testing

"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)" [8, 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 [8, 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, pp. 73, 237], [6], the definition of "interoperability testing" from [5, p. 238], and the decomposition of "compatibility" into "co-existence" and "interoperability" by [17]! The "interoperability" element of "compatibility testing" is explicitly excluded by [10, p. 37], (incorrectly) implying that "compatibility testing" and "co-existence testing" are synonyms. Furthermore, the definition of "compatibility testing" in [47, p. 43] unhelpfully says "See interoperability testing", adding another layer of confusion to the direction of their relationship.

V. Recommendations

We provide different recommendations for resolving various discrepancies (see Section IV). 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)
3) Consistent (e.g., backup/recovery testing and failover/recovery testing explicitly exclude an aspect included in its parent disaster/recovery testing)

The following are our recommendations for the areas of Recovery, Scalability, and Performance(-related) Testing, along with graphs of these subsets.

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. The result of the proposed terminology, along with their relations, is demonstrated in Figures 2 and 3.

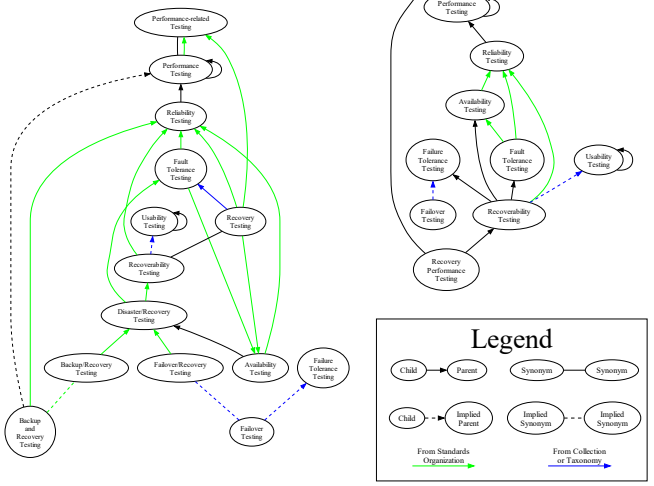

Fig. 2. Current relations between "recovery testing" terms.

Fig. 3. Proposed relations between rationalized "recovery testing" terms.

- Recoverability Testing: "Testing … aimed at verifying software restart capabilities after a system crash or other disaster" [7, p. 5-9] including "recover[ing] the data directly affected and re-establish[ing] the desired state of the system" [17] (similar in [7, p. 7-10]) so that the system "can perform required functions" [5, p. 370]. "Recovery testing" will be a synonym, as in [47, p. 47], since it is the more prevalent term throughout various sources, although "recoverability testing" is preferred to indicate that this explicitly 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" [7, p. 5-9] by entering a "backup operational mode in which [these responsibilities] … are assumed by a secondary system" [6]. 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" [10, 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" [10, 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" [7, p. 7-10] (similar in [17]) "within specified parameters of time, cost, completeness, and accuracy" [13, 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 [8, 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! [10, 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 [6], which defines "scalability" as "the degree to which a component or system can be adjusted for changing capacity". In contrast, capacity testing focuses 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 [17]. Because of this nuance, it makes more sense to consider these terms separate and not synonyms, as done by [22, p. 53] and [31, 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, [17] 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 the SWEBOK Guide V4's claim that one objective of elasticity testing is "to evaluate scalability" [7, p. 5-9]: [17]'s notion of "scalability" likely refers more accurately to "elasticity"! This also makes sense in the context of other definitions provided by the SWEBOK Guide V4:

- Scalability: "the software's ability to increase and scale up on its nonfunctional requirements, such as

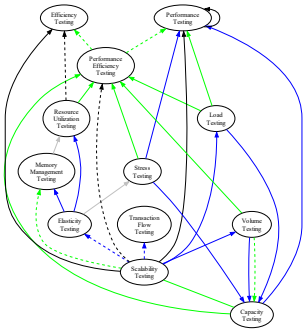

Fig. 4. Current relations between "scalability testing" terms.

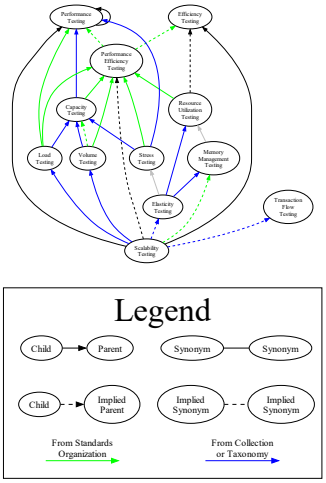

Fig. 5. Proposed relations between rationalized "scalability testing" terms.

load, number of transactions, and volume of data" [7, 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[8]: 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" [7, p. 5-9]. Based 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: [33] 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, the SWEBOK Guide 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" [7, 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 4 and 5.

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

C. Performance(-related) Testing

"Performance testing" is defined as testing "conducted to evaluate the degree to which a test item accomplishes its designated functions" [5, p. 320], [8, p. 7] (similar in [10, pp. 38-39], [36, p. 1187]). It does this by "measuring the performance metrics" [36, p. 1187] (similar in [6]) (such as the "system's capacity for growth" [57, p. 23]), "detecting the functional problems appearing under certain execution conditions" [36, p. 1187], and "detecting violations of non-functional requirements under expected and stress conditions" [36, p. 1187] (similar in [7, p. 5-9]). It is performed either …

1) … "within given constraints of time and other resources" [5, p. 320], [8, p. 7] (similar in [36, p. 1187]), or
2) … "under a 'typical' load" [10, 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'" [10, p. 38], along with other approaches like load and capacity testing [8, p. 22]. In contrast, [7, 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 [7, p. 5-9], [36, p. 1187], as noted in Section IV-B1. 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" [5, p. 320], [8, p. 7] (similar in [36, p. 1187]), and "performance testing" is a sub-approach of this performed "under a 'typical' load" [10, 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" [5, p. 253], [6], [8, p. 5], [10, p. 39], so it is a child of "performance-related testing" and a parent of "performance testing".

Finally, the "self-loops" mentioned in Section IV-B2 provide no new information and can be removed. These changes (along with those from Sections V-A and V-B made implicitly) result in the relations shown in Figure 6.

VI. Conclusion

While a good starting point, the current literature on software testing has much room to grow. The many ambiguities and confusions create unnecessary barriers to software testing. While there is merit to allowing the state-of-the-practice terminology to descriptively guide

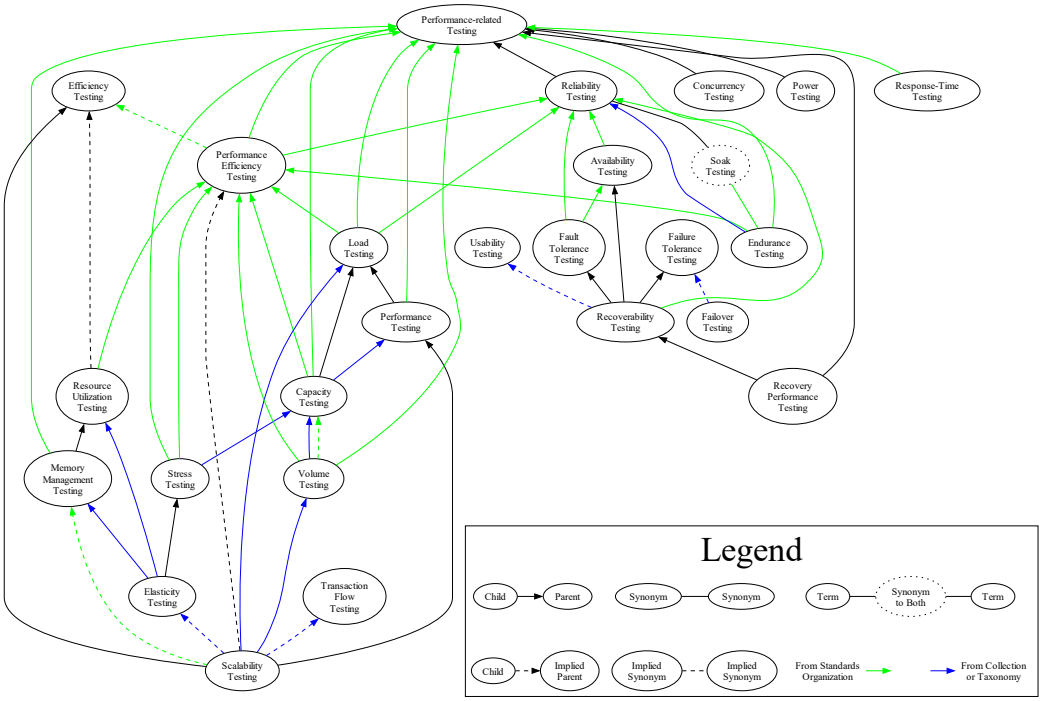

Fig. 6. Proposed relations between rationalized "performance-related testing" terms.

how terminology is used, there may be a need to prescriptively structure terminology to intentionally differentiate between and organize various test approaches. Future work in this area will continue to investigate the current use of terminology, in particular Undefined Terms, determine if IEEE's current Categories of Testing Approaches are sufficient, and rationalize the definitions of and relations between terms.

Acknowledgment

ChatGPT was used for proofreading and assistance with LaTeX formatting and supplementary Python code for constructing graphs and generating LaTeX code, including regex. Jason Balaci's McMaster thesis template provided many helper LaTeX functions.

References

[1] C. Kaner, J. Bach, and B. Pettichord, Lessons Learned in Software Testing: A Context-Driven Approach. John Wiley & Sons, Dec. 2011. [Online]. Available: https://www.wiley.com/en-ca/Lessons+Learned+in+Software+Testing%3A+A+Context-Driven+Approach-p-9780471081128

[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] 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

[8] ISO/IEC and IEEE, "ISO/IEC/IEEE International Standard - Systems and software engineering –Software testing –Part 1: General concepts," ISO/IEC/IEEE 29119-1:2022(E), Jan. 2022.

[9] T. P. Johnson, "Snowball Sampling: Introduction," in Wiley StatsRef: Statistics Reference Online. John Wiley & Sons, Ltd, 2014, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781118445112.stat05720. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118445112.stat05720

[10] 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.

[11] ——, "ISO/IEC/IEEE International Standard - Systems and software engineering –Systems and software assurance –Part 1: Concepts and vocabulary," ISO/IEC/IEEE 15026-1:2019, Mar. 2019.

[12] ——, "ISO/IEC/IEEE International Standard - Software and systems engineering –Software testing –Part 5: Keyword-Driven Testing," ISO/IEC/IEEE 29119-5:2016, Nov. 2016.

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

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

[15] ISO, "ISO 28881:2022 - Machine tools –Safety –Electrical discharge machines," ISO 28881:2022, Apr. 2022. [Online]. Available: https://www.iso.org/obp/ui#iso:std:iso:28881:ed-2:v1:en

[16] ——, "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

[17] 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

[18] ——, "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] ——, "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

[20] ——, "ISO/IEC 2382:2015 - Information technology –Vocabulary," ISO/IEC 2382:2015, May 2015. [Online]. Available: https://www.iso.org/obp/ui/en/#iso:std:iso-iec:2382:ed-1:v2:en

[21] ——, "ISO/IEC 25010:2011 - Systems and software engineering –Systems and software Quality Requirements and Evaluation (SQuaRE) –System and software quality models," ISO/IEC 25010:2011, Mar. 2011. [Online]. Available: https://www.iso.org/obp/ui/#iso:std:iso-iec:25010:ed-1:v1:en

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

[23] 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

[24] 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

[25] A. Dennis, B. H. Wixom, and R. M. Roth, System Analysis and Design, 5th ed. John Wiley & Sons, 2012. [Online]. Available: https://www.uoitc.edu.iq/images/documents/informatics-institute/Competitive_exam/Systemanalysisanddesign.pdf

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

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

[28] P. Gerrard and N. Thompson, Risk-based E-business Testing, ser. Artech House computing library. Norwood, MA, USA: Artech House, 2002. [Online]. Available: https://books.google.ca/books?id=54UKereAdJ4C

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

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

[31] 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

[32] LambdaTest, "What is Operational Testing: Quick Guide With Examples," 2024. [Online]. Available: https://www.lambdatest.com/learning-hub/operational-testing

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

[34] 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

[35] 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

[36] 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

[37] M. Bajammal and A. Mesbah, "Web Canvas Testing Through Visual Inference," in 2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST). Västerås, Sweden: IEEE, 2018, pp. 193–203. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8367048

[38] M. Dhok and M. K. Ramanathan, "Directed Test Generation to Detect Loop Inefficiencies," in Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, ser. FSE 2016. New York, NY, USA: Association for Computing Machinery, Nov. 2016, pp. 895–907. [Online]. Available: https://dl.acm.org/doi/10.1145/2950290.2950360

[39] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "The Oracle Problem in Software Testing: A Survey," IEEE Transactions on Software Engineering, vol. 41, no. 5, pp. 507–525, 2015.

[40] 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

[41] 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

[42] 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

[43] 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
[44] P. Godefroid and D. Luchaup, "Automatic Partial Loop Summarization in Dynamic Test Generation," in Proceedings of the 2011 International Symposium on Software Testing and Analysis, ser. ISSTA '11. New York, NY, USA: Association for Computing Machinery, Jul. 2011, pp. 23–33. [Online]. Available: https://dl.acm.org/doi/10.1145/2001420.2001424
[45] H. Yu, C. Y. Chung, and K. P. Wong, "Robust Transmission Network Expansion Planning Method With Taguchi's Orthogonal Array Testing," IEEE Transactions on Power Systems, vol. 26, no. 3, pp. 1573–1580, Aug. 2011. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5620950
[46] S. R. Choudhary, H. Versee, and A. Orso, "A Cross-browser Web Application Testing Tool," in 2010 IEEE International Conference on Software Maintenance. Timisoara, Romania: IEEE, Sep. 2010, pp. 1–6, iSSN: 1063-6773. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/5609728
[47] 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
[48] K.-L. Tsui, "An Overview of Taguchi Method and Newly Developed Statistical Methods for Robust Design," IIE Transactions, vol. 24, no. 5, pp. 44–57, May 2007, publisher: Taylor & Francis. [Online]. Available: https://doi.org/10.1080/07408179208964244
[49] 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.
[50] 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
[51] J. Berdine, C. Calcagno, and P. W. O'Hearn, "Smallfoot: Modular Automatic Assertion Checking with Separation Logic," 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. 115–137.
[52] 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.
[53] R. S. Sangwan and P. A. LaPlante, "Test-Driven Development in Large Projects," IT Professional, vol. 8, no. 5, pp. 25–29, Oct. 2006. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1717338
[54] P. Forsyth, T. Maguire, and R. Kuffel, "Real Time Digital Simulation for Control and Protection System Testing," in 2004 IEEE 35th Annual Power Electronics Specialists Conference (IEEE Cat. No.04CH37551), vol. 1. Aachen, Germany: IEEE, 2004, pp. 329–335.
[55] 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
[56] 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
[57] ——, "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
[58] C. Jard, T. Jéron, L. Tanguy, and C. Viho, "Remote testing can be as powerful as local testing," in Formal Methods for Protocol Engineering and Distributed Systems: Forte XII / PSTV XIX'99, ser. IFIP Advances in Information and Communication Technology, J. Wu, S. T. Chanson, and Q. Gao, Eds., vol. 28. Beijing, China: Springer, Oct. 1999, pp. 25–40. [Online]. Available: https://doi.org/10.1007/978-0-387-35578-8_2
[59] 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
[60] 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
[61] D. Trudnowski, B. Pierre, F. Wilches-Bernal, D. Schoenwald, R. Elliott, J. Neely, R. Byrne, and D. Kosterev, "Initial closed-loop testing results for the pacific DC intertie wide area damping controller," in 2017 IEEE Power & Energy Society General Meeting, 2017, pp. 1–5.
[62] B. J. Pierre, F. Wilches-Bernal, D. A. Schoenwald, R. T. Elliott, J. C. Neely, R. H. Byrne, and D. J. Trudnowski, "Open-loop testing results for the pacific DC intertie wide area damping controller," in 2017 IEEE Manchester PowerTech, 2017, pp. 1–6.
[63] W. Goralski, "xDSL loop qualification and testing," IEEE Communications Magazine, vol. 37, no. 5, pp. 79–83, 1999.
[64] M. Dominguez-Pumar, J. M. Olm, L. Kowalski, and V. Jimenez, "Open loop testing for optimizing the closed loop operation of chemical systems," Computers & Chemical Engineering, vol. 135, p. 106737, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0098135419312736
[65] C. Zhou, Q. Yu, and L. Wang, "Investigation of the Risk of Electromagnetic Security on Computer Systems," International Journal of Computer and Electrical Engineering, vol. 4, no. 1, p. 92, Feb. 2012, publisher: IACSIT Press. [Online]. Available: http://ijcee.org/papers/457-JE504.pdf
[66] 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
[67] D. C. Holley, G. D. Mele, and S. Naidu, "NASA Rat Acoustic Tolerance Test 1994-1995: 8 kHz, 16 kHz, 32 kHz Experiments," San Jose State University, San Jose, CA, USA, Tech. Rep. NASA-CR-202117, Jan. 1996. [Online]. Available: https://ntrs.nasa.gov/api/citations/19960047530/downloads/19960047530.pdf
[68] V. V. Morgun, L. I. Voronin, R. R. Kaspransky, S. L. Pool, M. R. Barratt, and O. L. Novinkov, "The Russian-US Experience with Development Joint Medical Support Procedures for Before and After Long-Duration Space Flights," NASA, Houston, TX, USA, Tech. Rep., 1999. [Online]. Available: https://ntrs.nasa.gov/api/citations/20000085877/downloads/20000085877.pdf
[69] R. B. Howe and R. Johnson, "Research Protocol for the Evaluation of Medical Waiver Requirements for the Use of Lisinopril in USAF Aircrew," Air Force Materiel Command, Brooks Air Force Base, TX, USA, Interim Technical Report AL/AO-TR-1995-0116, Nov. 1995. [Online]. Available: https://apps.dtic.mil/sti/tr/pdf/ADA303379.pdf
[70] D. Liu, S. Tian, Y. Zhang, C. Hu, H. Liu, D. Chen, L. Xu, and J. Yang, "Ultrafine SnPd nanoalloys promise high-efficiency electrocatalysis for ethanol oxidation and oxygen reduction," ACS Applied Energy Materials, vol. 6, no. 3, pp. 1459–1466, Jan. 2023, publisher: ACS Publications. [Online]. Available: https://pubs.acs.org/doi/pdf/10.1021/acsaem.2c03355?casa_token=ItHfKxeQNbsAAAAA:8zEdU5hi2HfHsSony3ku-lbH902jkHpA-JZw8jIeODzUvFtSdQRdbYhmVq47