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 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 532 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 discrepancies captured by this tool and manually through the research process. Our results reveal 234 discrepancies, including ten terms used as synonyms to two (or more) disjoint test approaches and 14 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. The productivity of testing and testing research would benefit from a standard language for communication. For example, [1, p. 7] gives the example of complete testing, which could require the tester to discover "every bug in the product", exhaust the time allocated to the testing phase, or simply implement every test previously agreed upon. Having a clear definition of "complete testing" reduces the chance for miscommunication and, ultimately, the tester getting "blamed for not doing … [their] job" [p. 7]. These benefits permeate software testing terminology. If software engineering holds code to high standards of clarity, consistency, and robustness, the same should apply to its supporting literature! Unfortunately, a search for a systematic, rigorous, and complete taxonomy for software testing revealed that the existing ones are inadequate and mostly focus on the high-level testing process rather than the testing approaches themselves:

- Tebes et al. [2] focus on parts of the testing process (e.g., test goal, test plan, testing role, testable entity) and how they relate to one another,
- Souza et al. [3] prioritize organizing testing approaches over defining them, and
- Unterkalmsteiner et al. [4] focus on the "information linkage or transfer" [p. A:6] between requirements engineering and software testing and "do[] not aim at providing a systematic and exhaustive state-of-the-art survey of [either domain]" [p. A:2].

Some existing collections of software testing terminology were found, but in addition to being incomplete, they also contained many oversights. In particular, discrepancies between the definitions of test approaches can lead to the miscommunication mentioned by [1, p. 7]. ISO/IEC and IEEE categorize experience-based testing as both a test design technique and a test practice on the same page—twice [5, Fig. 2, p. 34]! The structure of tours can be defined as either quite general [5, p. 34] or "organized around a special focus" [6]. Load testing is performed with loads "between anticipated conditions of low, typical, and peak usage" [5, p. 5] or loads that are as large as possible [7, p. 86]. Alpha testing is performed by "users within the organization developing the software" [8, p. 17], "a small, selected group of potential users" [9, p. 5-8], or "roles outside the development organization" conducted "in the developer's test environment" [6]. With inconsistencies such as these, it is clear that there is a notable gap in the literature, one which we attempt to describe and fill. While the creation of a complete taxonomy is unreasonable, especially considering the pace at which the field of software changes, we can make progress towards this goal that others can extend and update as new test approaches emerge.

This document describes this process, as well as its results, in more detail. 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 in the literature, either within the same source or between various sources (Section IV). This reinforces the need for a proper taxonomy! We

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

provide some potential solutions covering some of these discrepancies (Section V).

II. Scope

Since our motivation is restricted to testing code, only this component of Verification and Validation (V&V) is considered. However, some test approaches are used for testing things other than code, and some approaches can be used for both! In these cases, only the subsections of these approaches focused on code are considered. For example, reliability testing and maintainability testing can start without code by "measur[ing] structural attributes of representations of the software" [10, p. 18], but only reliability and maintainability testing performed on code itself is in scope of this research. Therefore, some practices are excluded from consideration either in part or in full; hardware testing (Section II-A) and the V&V of other artifacts (Section II-B) are completely out of scope, as well as relevant areas of other testing approaches that are otherwise in scope. Static testing can be performed on code, so while it isn't relevant to the original motivation of this work, it is a useful component of software testing and is therefore included at this level of analysis (Section II-C).

A. Hardware Testing

While testing the software run on or in control of hardware is in scope, testing performed on the hardware itself is out of scope. The following are some examples of hardware testing approaches:

- Ergonomics testing and proximity-based testing (see [6]) are out of scope, since they are used for testing hardware.
- Similarly, EManations SECurity (EMSEC) testing [11], [12, p. 95], which deals with the "security risk" of "information leakage via electromagnetic emanation" [12, p. 95], is also out of scope.
- Orthogonal Array Testing (OAT) can be used when testing software [13] (in scope) but can also be used for hardware [14, pp. 471-472], such as "processors … made from pre-built and pre-tested hardware components" [p. 471] (out of scope). A subset of OAT called "Taguchi's Orthogonal Array Testing (TOAT)" is used for "experimental design problems in manufacturing" [15, p. 1573] or "product and manufacturing process design" [16, p. 44] and is thus also out of scope.

B. V&V of Other Artifacts

The only testing of a software artifact produced by the software life cycle that is in scope is testing of the software itself, as demonstrated by the following examples:

- Design reviews and documentation reviews are out of scope, as they focus on the V&V of design [8, pp. 132] and documentation [8, pp. 144], respectively.
- Error seeding is the "process of intentionally adding known faults[1] 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" [8, p. 165], which helps verify the actual code (in scope).
- Fault injection testing, where "faults are artificially introduced[1] into the SUT [System Under Test]", can be used to evaluate the effectiveness of a test suite [9, p. 5-18], which is a part of V&V of the V&V itself (out of scope), or "to test the robustness of the system in the event of internal and external failures" [5, p. 42], which helps verify the actual code (in scope).
- "Mutation [t]esting was originally conceived as a technique to evaluate test suites in which a mutant is a slightly modified version of the SUT" [9, p. 5-15], which is in the realm of V&V of the V&V itself (out of scope). However, it "can also be categorized as a structure-based technique" and can be used to assist fuzz and metamorphic testing [9, p. 5-15] (in scope).

C. Static Testing

Sometimes, static testing is excluded from software testing [17, p. 222], [18, p. 13], restricting "testing" to mean dynamic validation [9, p. 5-1] or verification "in which a system or component is executed" [8, p. 427]. However, "terminology is not uniform among different communities, and some use the term 'testing' to refer to static techniques[2] as well" [9, p. 5-2]. This is done by [19, pp. 8-9] and [5, p. 17]; the latter even explicitly exclude static testing in another document [8, p. 440]!

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" [20], 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

[1] While error seeding and fault injection testing both introduce faults as part of testing, they do so with different goals: to "estimat[e] the number of faults remaining" [8, p. 165] and "test the robustness of the system" [5, p. 42], respectively. Therefore, these approaches are not considered synonyms, and the lack of this relation in the literature is not included in Section IV-B1 as a synonym discrepancy.

[2] Not formally defined, but distinct from the notion of "test technique" described in Table I.

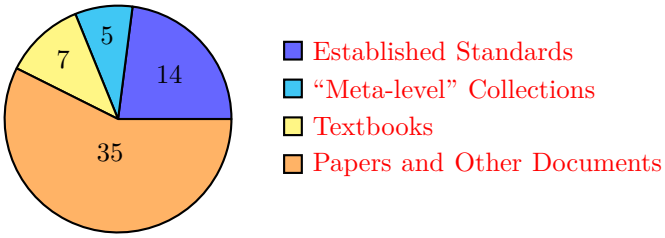

Fig. 1. Summary of how many sources comprise each source category.

(Section III-A1), "meta-level" collections (Section III-A2), textbooks (Section III-A3), and papers and other documents (Section III-A4). Each category is given a unique colour in Figures 2 to 6 to better visualize the source of information in these graphs. A summary of how many sources comprise each category is given in Figure 1.

1) Established Standards: [5], [8], [21]–[32]

- 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" [5, p. vii]

2) "Meta-level" Collections: [6], [9], [18], [33], [34]

- Colored blue
- Collections of relevant terminology, such as ISTQB's glossary [6], the SWEBOK Guide [9], [34], and Doğan et al.'s literature review [33]
- Built up from various sources, including established standards (see Section III-A1), 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], [7], [35]–[39]

- Colored maroon
- Textbooks trusted at McMaster [7], [38], [39] 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], [13]–[16], [19], [40]–[68]

- Colored black
- Mainly consists of academic papers: journal articles, conference papers, reports [19], [57], [66], and a thesis [40]
- 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 [42], [43], a booklet [44], and ChatGPT [41] (with claims supported by [58])

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 test approach categories (Section III-B1) and parent-child relations (Section III-B2) 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" in Section III-B3.

1) Categories of Testing Approaches: Different sources categorize software testing approaches differently. For example, ISO/IEC and IEEE [5] provide the classification of test approaches in Table I. These categories ("test level" and "test type" in particular) seem to be widely used. For example, in addition to the IEEE sources (given in Table I), six additional sources give unit testing, integration testing, system testing, and acceptance testing as examples of test levels [6], [9, pp. 5-6 to 5-7], [19, pp. 9, 13], [36, p. 807-808], [38, pp. 443-445], [51, p. 218], although they may use a different term for "test level" (see Table I). Because of their widespread use and their usefulness when focusing on a particular subset of testing, these categories are used for now. We did, however, note the potential significance of an "artifact" category, since some terms could refer to the application of a test approach and/or the resulting document(s). Because of this, a test approach being categorized as a category from Table I and an artifact is not a discrepancy.

One important side effect of the particularity of these terms is that they can be "overloaded"; for example, someone could reasonably yet imprecisely use any of these four categories as a synonym for "approach". Even the prompt in [41] was imprecise, asking for the "type of software testing that focuses on looking for bugs where others have already been found." Interestingly, ChatGPT later "corrected" this by calling detect-based testing an "approach"! Because of this, careful consideration needs to be given to discrepancies of this nature. For example, [57, p. 45]'s definition of "interface testing" is "an integration test type that is concerned with testing … interfaces", but since he does not define "test type", it may not have special significance.

2) Parent-Child Relations: Many test approaches are multi-faceted and can be "specialized" into others, such as performance-related testing (see Section V-C). These "specializations" will be referred to as "children" or "sub-approaches" of the multi-faceted "parent". This nomen-

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" [5, p. 10]; described by a test strategy [8, p. 472] and is also used to "pick the particular test case values" [8, p. 465]	black or white box, minimum and maximum boundary value testing [8, p. 465]
(Design)[a] Technique	A "defined" and "systematic" [8, p. 464] "procedure used to create or select a test model, identify test coverage items, and derive corresponding test cases" [5, p. 11] (similar in [8, p. 467]) "that … generate evidence that test item requirements have been met or that defects are present in a test item" [21, p. vii]; "a variety … is typically required to suitably cover any system" [5, p. 33] and is "often selected based on team skills and familiarity, on the format of the test basis", and on expectations [5, p. 23]	equivalence partitioning, boundary value analysis, branch testing [5, 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 [5, p. 12], [21, p. 6] with their "own documentation and resources" [8, p. 469]; more generally, "designat[es] … the coverage and detail" [8, p. 249]	unit/component testing, integration testing, system testing, acceptance testing [8, p. 467], [5, p. 12], [21, p. 6]
Practice	A "conceptual framework that can be applied to … [a] test process to facilitate testing" [8, p. 471], [5, p. 14]; more generally, a "specific type of activity that contributes to the execution of a process" [8, p. 331]	scripted testing, exploratory testing, automated testing [5, p. 20]
Type	"Testing that is focused on specific quality characteristics" [8, p. 473], [5, p. 15], [21, p. 7]	security testing, usability testing, performance testing [8, p. 473], [5, p. 15]

[a] "Design technique" is sometimes abbreviated to "technique" [6], [5, 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" [5, p. 24].
[c] "Test phase" can be a synonym for "test level" [8, p. 469], [24, p. 9] but can also refer to the "period of time in the software life cycle" when testing occurs [8, p. 470], usually after the implementation phase [8, pp. 420, 509], [36, p. 56].
[d] Used by [6], [9, pp. 5-6 to 5-7], [39, pp. 438-440], [19, p. 9].

clature also extends to other categories given in Section III-B1 and Table I, such as "sub-type". One example of these specializations is the "stronger than" relation (also called the "subsumes" relation) described by [39, p. 432]: when comparing adequacy criteria (which "specif[y] requirements for testing" [p. 402]), "criterion X is stronger than criterion Y if, for all programs P and all test sets T, X-adequacy implies Y-adequacy". While this relation only "compares the thoroughness of test techniques, not their ability to detect faults" [p. 434], it is sufficient to consider one a child of the other.

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 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 (see Section III-E) by looking for indicators of uncertainty, such as question marks, " (Testing)" (which indicates that a test approach isn't explicitly denoted as such; note the inclusion of the space), and the keywords "implied", "inferred", "can be", "should 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 (see Section III-B2). Any other notes, such as uncertainties, prerequisites, and other resources to investigate, are also recorded. If an approach is assigned a category, such as those found in Table I and some outliers (e.g., "artifact"), this is also tracked for future investigation.

Most sources are analyzed in their entirety to systematically extract terminology, especially established standards (see Section III-A1). Sources that were only partially investigated include those 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 are given similar definitions by multiple sources, the clearest and most concise version is kept. If definitions from different sources overlap, provide different information, or contradict, they are merged to paint a more complete picture. When contradictions or other discrepancies (see Section IV) arise, they are investigated and documented. Any test approaches that are mentioned but not defined are added to the glossary to indicate they should be investigated further (see Section III-D). Similar methodologies are used for tracking software qualities and supplementary terminology that is shared by multiple approaches or is too complicated to explain inline; these are tracked in separate documents. The name, definition, and synonym(s) of all terms are tracked, as well as any precedence for a related test type for a given software quality.

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 during future analysis.

D. Undefined Terms

The search process led to some testing approaches being mentioned without definition; [5] and [18] in particular introduced many. Once the standards in Section III-A1 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 and aspects of orthogonal array testing; see Section II).

The following terms (and their respective related terms) were explored in the following sources, bringing the number of testing approaches from 448 to 532 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: [52], [61], [62]
- Loop Testing[3]: [49], [54], [55], [64]
- EMSEC Testing: [11], [12]
- Asynchronous Testing: [67]
- Performance(-related) Testing: [47]
- Web Application Testing: [33], [57]
  - HTML Testing: [56], [65], [66]
  - Document Object Model (DOM) Testing: [48]
- Sandwich Testing: [45], [63]
- Orthogonal Array Testing[4]: [13], [14]
- Backup Testing[5]: [40]

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 (see Sections IV-D and IV-E, respectively), 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 in Section V, 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.

IV. Discrepancies

After gathering all these data[6], 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. As an example, the structure of tours can be defined as either quite general [5, p. 34] or "organized around a special focus" [6], which is a case of contradictory definitions; therefore, it is included with the Contradictions (its class) and with the Definition Discrepancies (its category). Within these sections, "less significant" discrepancies are omitted for brevity, and those remaining are then sorted based on their source category (see Section III-A).

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 (see Section III-A) and/or with a "more trusted" one (i.e., a previous row in the table). The numbers of (Exp)licit

[3]References [27] and [26] were used as reference for terms but not fully investigated, [69] and [70] were added as potentially in scope, and [71] and [72] were added as out-of-scope examples.

[4]References [15] and [16] were added as out-of-scope examples.

[5]See Section IV-D.

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

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	17	10	3	0	5	0	0	0	42
"Meta-level" Collections	10	0	1	0	31	19	9	3	5	1	2	0	81
Textbooks	6	0	1	0	33	5	5	0	1	0	0	0	51
Papers and Other Documents	7	1	4	0	20	11	9	3	2	1	2	0	60
Total	27	2	8	0	101	45	26	6	13	2	4	0	234

[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	3	2	6	0	10	7	9	2	3	0	0	0	42
"Meta-level" Collections	8	4	9	3	9	14	17	0	11	2	4	0	81
Textbooks	12	1	8	3	2	0	17	1	7	0	0	0	51
Papers and Other Documents	14	7	8	0	10	7	5	0	7	2	0	0	60
Total	37	14	31	6	31	28	48	3	28	4	4	0	234

and (Imp)licit (see Section III-B3) discrepancies are also presented in these tables. Since each discrepancy is given a class and a category, the totals per source and grand totals in these tables are equal. However, the numbers of discrepancies listed in Sections IV-A and IV-B are not equal, as those automatically uncovered based on semantics are only listed in the corresponding category section for clarity; they still contribute to the counts in Table II!

Moreover, certain "subsets" of testing revealed many interconnected discrepancies. These are given in their respective sections as a "third view" to keep related information together, but still count towards the classes and categories of discrepancies listed above, causing a further mismatch between the counts in Tables II and III and the counts in Sections IV-A and IV-B. 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[7].

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 [23, 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 [18, p. 56]; elsewhere, they seem to refer to testing the acoustic tolerance of rats [73] or the acceleration tolerance of astronauts [74, p. 11], aviators [75, pp. 27, 42], or catalysts [76, p. 1463], which don't exactly seem relevant…
- Reference [7, p. 119] says that branch testing is "the simplest form of path testing" which seems incorrect.
- Reference [38] 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 [38, p. 447]!
- Reference [57, 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 inte-

[7]Section omitted for brevity.

gration testing are all listed as "common test levels" [5, p. 12], [21, p. 6], but no definitions are given for the latter two, making it unclear what "system integration testing" is; it is a combination of the two? somewhere on the spectrum between them? It is listed as a child of integration testing by [6] and of system testing by [18, p. 23].
- Similarly, component testing, integration testing, and component integration testing are all listed in [8], but "component integration testing" is only defined as "testing of groups of related components" [8, 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 [57, 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:

- While ISO/IEC and IEEE [5, p. 8] say regression testing and retesting are two distinct approaches (as does [18, p. 34]), they define regression testing as a form of "selective retesting" in [8, p. 372] (as does [9, pp. 5-8, 6-5, 7-5 to 7-6]). Moreover, the two possible variations of regression testing given by [39, p. 411] are "retest-all" and "selective retest", which is possibly the source of the above misconception; these two in tandem create a cyclic relation between regression testing and selective retesting.
- A component is an "entity with discrete structure … within a system considered at a particular level of analysis" [29] 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 [8, p. 82]. For example, [6] defines them as synonyms while [60, p. 107] says "components differ from classical modules for being re-used in different contexts independently of their development". Additionally, since components are structurally, functionally, or logically discrete [8, p. 419] and "can be tested in isolation" [6], "unit/component/module testing" could refer to the testing of both a module and a specific function in a module, introducing a further level of ambiguity.
- Performance testing and security testing are given as subtypes of reliability testing by [28], but these are all listed separately by [18, p. 53].
- Similarly, random testing is a subtechnique of specification-based testing [5, pp. 7, 22], [6], [9, p. 5-12], [21, pp. 5, 20, Fig. 2] but is listed separately by [18, p. 46].
- Path testing can "aim[] to execute all entry-to-exit control flow paths in a SUT's control flow graph" [9, p. 5-13] or "all or selected paths through a computer program" [8, p. 316] (similar in [7, p. 119]).
- The structure of tours can be defined as either quite general [5, p. 34] or "organized around a special focus" [6].
- Alpha testing is performed by "users within the organization developing the software" [8, p. 17], "a small, selected group of potential users" [9, p. 5-8], or "roles outside the development organization" conducted "in the developer's test environment" [6].
- While [7, p. 120] implies that condition testing is a sub-technique of path testing, [39, Fig. 13.17] says that multiple condition coverage (which seems to be a synonym of condition coverage [p. 422]) does not subsume and is not subsumed by path coverage.
- Load testing is performed with loads "between anticipated conditions of low, typical, and peak usage" [5, p. 5] or loads that are as large as possible [7, p. 86].
- State testing requires that "all states in the state model … [are] 'visited'" in [21, p. 19] which is only one of its possible criteria in [7, pp. 82-83].
- Reference [8, p. 456] says system testing is "conducted on a complete, integrated system" (which [38, Tab. 12.3] and [39, p. 439] agree with), while [7, p. 109] says it can also be done on "at least a major portion" of the product.
- "Walkthroughs" and "structured walkthroughs" are given as synonyms by [6] but [38, p. 484] implies that they are different, saying a more structured walkthrough may have specific roles.
- Reference [7, 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 [39, pp. 418-419].

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

- The distinctions between development testing [8, p. 136], developmental testing [18, p. 30], and developer testing [18, p. 39], [19, 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.
- "Installability testing" is given as a test type [5, p. 22], [21, p. 38], [8, p. 228], while "installation

testing" is given as a test level [39, p. 439]. Since "installation testing" is not given as an example of a test level throughout the sources that describe them (see Section III-B1), it is likely that the term "installability testing" with all its related information should be used instead.
- Reference [6] claims that code inspections are related to peer reviews but [7, 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" [9, p. 5-10]; this seems to overlap (both in scope and name) with the definition of "security testing" in [5, 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" [9, pp. 5-1, 5-11] and "operational acceptance testing" [18, 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 [8, p. 431], [6], [9, p. 5-10], [5, p. 9], [39, p. 399], [21, p. 8], [53, p. 344]
2) Closed-Box Testing [8, p. 431], [5, p. 9]
3) Functional Testing[8] [8, p. 196], [39, p. 399], [57, p. 44]
4) Domain Testing [9, 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" [29] 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 [8, p. 82]. For example, [6] defines them as synonyms while [60, p. 107] says "components differ from classical modules for being re-used in different contexts independently of their development". Additionally, since components are structurally, functionally, or logically discrete [8, p. 419] and "can be tested in isolation" [6], "unit/component/module testing" could refer to the testing of both a module and a specific function in a module, introducing a further level of ambiguity.
- Reference [6] claims that code inspections are related to peer reviews but [7, pp. 94-95] makes them quite distinct.
- "Walkthroughs" and "structured walkthroughs" are given as synonyms by [6] but [38, p. 484] implies that they are different, saying a more structured walkthrough may have specific roles.
- Reference [38] 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 [38, p. 447]!

There are also cases in which a term is given as a synonym to two (or more) disjoint, unrelated terms, making the relation between the given synonyms ambiguous. 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 [57, p. 45]
   - Negative Testing [6] (implied by [21, p. 10])
2) Soak Testing:
   - Endurance Testing [21, p. 39]
   - Reliability Testing[9] [66, Tab. 1, p. 26], [19, Tab. 2]
3) User Scenario Testing:
   - Scenario Testing [6]
   - Use Case Testing[10] [57, p. 48] (although "an actor can be a user or another system" [21, p. 20])
4) Link Testing:
   - Branch Testing (implied by [21, p. 24])
   - Component Integration Testing [57, p. 45]
   - Integration Testing (implied by [19, p. 13])

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

[9] Endurance testing is given as a kind of reliability testing by [18, p. 55], although the terms are not synonyms.

[10] "Scenario testing" and "use case testing" are given as synonyms by [6] and [57, pp. 47-49] but listed separately by [5, p. 22], which also gives "use case testing" as a "common form of scenario testing" [21, 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	[21, p. 19]	[57, p. 15]
Co-existence Testing	→	Compatibility Testing	[5, p. 3], [28], [21, Tab. A.1]	[21, p. 37]
Fault Tolerance Testing	→	Robustness Testing[a]	[18, p. 56]	[6]
Functional Testing	→	Specification-based Testing[b]	[21, p. 38]	[8, p. 196], [39, p. 399], [57, p. 44]
Orthogonal Array Testing	→	Pairwise Testing	[13, p. 1055]	[9, p. 5-11], [14, p. 473]
Path Testing	→	Exhaustive Testing	[38, pp. 466-467, 476]	[39, p. 421]
Performance Testing	→	Performance-related Testing	[5, p. 22], [21, p. 38]	[47, p. 1187]
Use Case Testing	→	Scenario Testing	[21, p. 20]	[6], [57, pp. 47-49]

[a]Fault tolerance testing may also be a sub-approach of reliability testing [8, p. 375], [9, p. 7-10], which is distinct from robustness testing [18, p. 53].
[b]See Section IV-C1.

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 [28], but these are all listed separately by [18, 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 Tables II and III. For example, performance and usability testing are both given as sub-approaches of themselves [19, Tab. 2], [66, 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 14 of these pairs through automatic analysis of the generated graphs, with the most prominent given in Table IV. Of particular note is the relation between path testing and exhaustive testing. While [39, p. 421] claims that path testing done completely "is equivalent to exhaustively testing the program"[11], this overlooks the effect of implementation issues [38, p. 476] and input data [38, p. 467], [7, p. 121] on the code's behaviour. Exhaustive testing requires "all combinations of input values and preconditions … [to be] tested" [5, p. 4] (similar in [6], [7, p. 121]).

3) Test Approach Category Discrepancies: While the IEEE categorization of testing approaches described in Table I is useful, it is not without its faults. One issue, which is not inherent to the categorization itself, is the fact that it is not used consistently (see ??). The most blatant example of this is that [8, p. 286] describe mutation testing as a methodology, even though this is not one of the categories they created! Additionally, the boundaries between approaches within a category may be unclear: "although each technique is defined independently of all others, in practice [sic] some can be used in combination with other techniques" [21, p. 8]. For example, "the test coverage items derived by applying equivalence partitioning can be used to identify the input parameters of test cases derived for scenario testing" [p. 8]. Even the categories themselves are not consistently defined, and some approaches are categorized differently by different sources; these differences are tracked so they can be analyzed more systematically.

- Since keyword-driven testing can be used for automated or manual testing [23, pp. 4, 6], the claim that "test cases can be either manual test cases or keyword test cases" [p. 6] is incorrect.

Some category discrepancies can be detected automatically, given in ??; of these, experience-based testing is of particular note. ISO/IEC and IEEE categorize experience-based testing as both a test design technique and a test practice on the same page—twice [5, Fig. 2, p. 34]! These authors previously say "experience-based testing practices like exploratory testing … are not … techniques for designing test cases", although they "can use … test techniques" [21, p. viii]. In addition to contradicting other authors (including themselves in [5, p. 34]!), it also implies that "experience-based test design techniques" are techniques used by the practice of experience-based testing, not that experience-based testing is itself a test technique. If this is the case, it makes the distinction between "practice" and "technique" less useful, which may explain why experience-based testing is categorized inconsistently in the literature.

Subapproaches of experience-based tesing, such as error guessing and exploratory testing, are also categorized ambiguously, causing confusion on how categories and parent-child relations (see Section III-B2) interact. [5, p. 34] says "[another standard [21]] describes the experience-based test design technique of error guessing. Other experience-based test practices include (but are not limited to) exploratory testing …, tours, attacks, and checklist-based testing"; this seems to imply that error guessing is both a technique and a practice, which does not make sense if these categories are orthogonal. Similarly, the conflicting

[11]The contradictory definitions of path testing given in Section IV-B4 add another layer of complexity to this claim.

categorizations of beta testing in ?? may also propagate to closed beta testing and open beta testing; since this is an inferrence, it is omitted from that table and instead included in ??. These kinds of inconsistencies between parent and child test approach categorizations may indicate that categories are not transitive or that more thought must be given to them.

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" [5, p. 12], [21, p. 6], but no definitions are given for the latter two, making it unclear what "system integration testing" is; it is a combination of the two? somewhere on the spectrum between them? It is listed as a child of integration testing by [6] and of system testing by [18, p. 23].
- Similarly, component testing, integration testing, and component integration testing are all listed in [8], but "component integration testing" is only defined as "testing of groups of related components" [8, 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" [9, p. 5-10]; this seems to overlap (both in scope and name) with the definition of "security testing" in [5, 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.
- Path testing can "aim[] to execute all entry-to-exit control flow paths in a SUT's control flow graph" [9, p. 5-13] or "all or selected paths through a computer program" [8, p. 316] (similar in [7, p. 119]).
- The structure of tours can be defined as either quite general [5, p. 34] or "organized around a special focus" [6].
- Alpha testing is performed by "users within the organization developing the software" [8, p. 17], "a small, selected group of potential users" [9, p. 5-8], or "roles outside the development organization" conducted "in the developer's test environment" [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 with loads "between anticipated conditions of low, typical, and peak usage" [5, p. 5] or loads that are as large as possible [7, p. 86].
- State testing requires that "all states in the state model … [are] 'visited'" in [21, p. 19] which is only one of its possible criteria in [7, pp. 82-83].
- Reference [8, p. 456] says system testing is "conducted on a complete, integrated system" (which [38, Tab. 12.3] and [39, p. 439] agree with), while [7, p. 109] says it can also be done on "at least a major portion" of the product.
- Reference [7, 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 [39, pp. 418-419].
- Reference [57, 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 [57, 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 [8, p. 136], developmental testing [18, p. 30], and developer testing [18, p. 39], [19, p. 11] are unclear and seem miniscule.
- The terms "acceleration tolerance testing" and "acoustic tolerance testing" seem to only refer to software testing in [18, p. 56]; elsewhere, they seem to refer to testing the acoustic tolerance of rats [73] or the acceleration tolerance of astronauts [74, p. 11], aviators [75, pp. 27, 42], or catalysts [76, p. 1463], which don't exactly seem relevant…
- "Orthogonal array testing" [9, pp. 5-1, 5-11] and "operational acceptance testing" [18, p. 30] have the same acronym ("OAT").
- "Installability testing" is given as a test type [5, p. 22], [21, p. 38], [8, p. 228], while "installation testing" is given as a test level [39, p. 439]. Since "installation testing" is not given as an example of a test level throughout the sources that describe them (see Section III-B1), it is likely that the term "installability testing" with all its related information should be used instead.

6) Citation Discrepancies: Sometimes a document cites another for a piece of information that does not appear! For example, [33, p. 184] claims that [53] 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" [5, 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" [8, p. 196]. Notably, [8] 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 [5, pp. 2-4, 6-9, 22] is a test design technique used to "derive corresponding test cases" [5, p. 11] from "selected inputs and execution conditions" [8, 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" [9, p. 5-7]; this mirrors previous definitions of "functional testing" [8, p. 196], [5, p. 21] but groups it with "correctness testing". Since "correctness" is a software quality [8, p. 104], [9, p. 3-13] which is what defines a "test type" [5, p. 15], it seems consistent to label "functional testing" as a "test type" [5, pp. 15, 20, 22], [21, pp. 7, 38, Tab. A.1], [23, 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 [18, p. 53].

3) Conformance Testing: Testing that ensures "that the functional specifications are correctly implemented", and can be called "conformance testing" or "functional testing" [9, 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" [9, 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" [57, 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" [8, p. 93], which seems to describe something different.

4) Functional Suitability Testing: Procedure testing is called a "type of functional suitability testing" [5, 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" [28]. 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" [28] 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" [8, 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 [28], 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" [19, Tab. 2]. "Functionality testing" is listed separately from "correctness testing" by [18, p. 53].

D. Recovery Testing

"Recovery testing" is "testing … aimed at verifying software restart capabilities after a system crash or other disaster" [9, p. 5-9] including "recover[ing] the data directly affected and re-establish[ing] the desired state of the system" [28] (similar in [9, p. 7-10]) so that the system "can perform required functions" [8, p. 370]. It is also called "recoverability testing" [57, p. 47] and potentially "restart & recovery (testing)" [19, Fig. 5]. The following terms, along with "recovery testing" itself [5, 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" [9, p. 7-10] (similar in [28]) and "re-establish the desired state of the system" [28]. Synonym for "recovery testing" in [57, p. 47].
- Disaster/Recovery Testing serves to evaluate if a system can "return to normal operation after a hardware or software failure" [8, p. 140] or if "operation of the test item can be transferred to a different operating site and … be transferred back again once the failure has been resolved" [21, p. 37]. These two definitions seem to describe different aspects of the system, where the first is intrinsic to the hardware/software and the second might not be.
- Backup and Recovery Testing "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" [24, 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 [8, p. 140].

- Backup/Recovery Testing: Testing that determines the ability "to restor[e] from back-up memory in the event of failure, without transfer[ing] to a different operating site or back-up system" [21, p. 37]. This seems to correspond to the definition of "disaster/recovery testing" in [21, p. 37]. It is also given as a sub-type of "disaster/recovery testing", even though that tests if "operation of the test item can be transferred to a different operating site" [p. 37]. 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" [21, p. 37]. This is given as a sub-type of "disaster/recovery testing", even though that tests if "operation of the test item can be transferred to a different operating site" [p. 37].
- Failover Testing: Testing that "validates the SUT's ability to manage heavy loads or unexpected failure to continue typical operations" [9, p. 5-9] by entering a "backup operational mode in which [these responsibilities] … are assumed by a secondary system" [6]. While not explicitly related to recovery, "failover/recovery testing" also describes the idea of "failover", and [18, 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" [21, p. 39] while other sources differentiate between the two [18, p. 53], [40, pp. 22-23]
2) ISO/IEC and IEEE give the external modification of the system as part of "scalability" [21, p. 39], while [28] implies that it is limited to the system itself
3) The SWEBOK Guide V4's definition of "scalability testing" [9, 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)" [5, 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 [5, 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 [8, pp. 73, 237], [6], the definition of "interoperability testing" from [8, p. 238], and the decomposition of "compatibility" into "co-existence" and "interoperability" by [28]! The "interoperability" element of "compatibility testing" is explicitly excluded by [21, p. 37], (incorrectly) implying that "compatibility testing" and "co-existence testing" are synonyms. Furthermore, the definition of "compatibility testing" in [57, 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.

- Recoverability Testing: "Testing … aimed at verifying software restart capabilities after a system crash or other disaster" [9, p. 5-9] including "recover[ing] the data directly affected and re-establish[ing] the desired state of the system" [28] (similar in [9, p. 7-10]) so that the system "can perform required functions" [8, p. 370]. "Recovery testing" will be a synonym, as in [57, 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

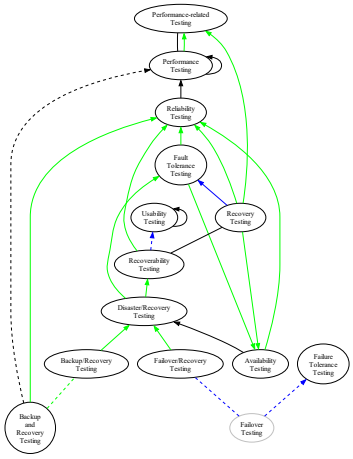

Fig. 2. Current relations between "recovery testing" terms.

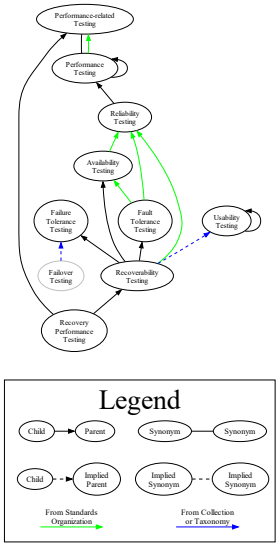

Fig. 3. Proposed relations between rationalized "recovery testing" terms.

to continue typical operations" [9, p. 5-9] by entering a "backup operational mode in which [these responsibilities] … are assumed by a secondary system" [6]. 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" [21, 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" [21, 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" [9, p. 7-10] (similar in [28]) "within specified parameters of time, cost, completeness, and accuracy" [24, 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 [5, 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! [21, 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 [28]. Because of this nuance, it makes more sense to consider these terms separate and not synonyms, as done by [18, p. 53] and [40, 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, [28] 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" [9, p. 5-9]: [28]'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 load, number of transactions, and volume of data" [9, 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[12]: 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" [9, 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.

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

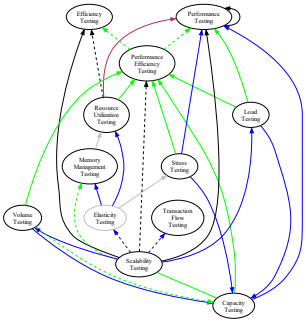

Fig. 4. Current relations between "scalability testing" terms.

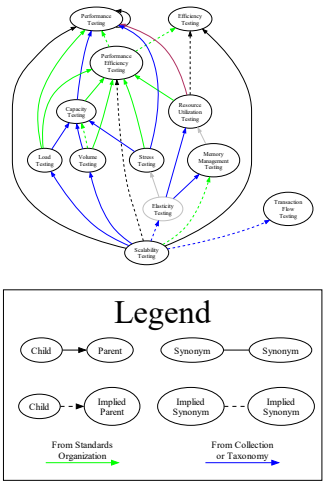

Fig. 5. Proposed relations between rationalized "scalability testing" terms.

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, 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" [9, 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.

C. Performance(-related) Testing

"Performance testing" is defined as testing "conducted to evaluate the degree to which a test item accomplishes its designated functions" [8, p. 320], [5, p. 7] (similar in [21, pp. 38-39], [47, p. 1187]). It does this by "measuring the performance metrics" [47, p. 1187] (similar in [6]) (such as the "system's capacity for growth" [66, p. 23]), "detecting the functional problems appearing under certain execution conditions" [47, p. 1187], and "detecting violations of non-functional requirements under expected and stress conditions" [47, p. 1187] (similar in [9, p. 5-9]). It is performed either …

1) "within given constraints of time and other resources" [8, p. 320], [5, p. 7] (similar in [47, p. 1187]), or
2) "under a 'typical' load" [21, p. 39].

It is listed as a subset of performance-related testing, which is defined as testing "to determine whether a test item performs as required when it is placed under various types and sizes of 'load' " [21, p. 38], along with other approaches like load and capacity testing [5, p. 22]. Note that "performance, load and stress testing might considerably overlap in many areas" [47, p. 1187]. In contrast, [9, p. 5-9] gives "capacity and response time" as examples of "performance characteristics" that performance testing would seek to "assess", which seems to imply that these are sub-approaches to performance testing instead. This is consistent with how some sources treat "performance testing" and "performance-related testing" as synonyms [9, p. 5-9], [47, 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" [8, p. 320], [5, p. 7] (similar in [47, p. 1187]), and "performance testing" is a sub-approach of this performed "under a 'typical' load" [21, p. 39]. This has other implications for relations between these types of testing; for example, "load testing" usually occurs "between anticipated conditions of low, typical, and peak usage" [8, p. 253], [6], [5, p. 5], [21, 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 discrepancies create unnecessary barriers to software testing. While there is merit to allowing the state-of-the-practice terminology to descriptively guide 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.

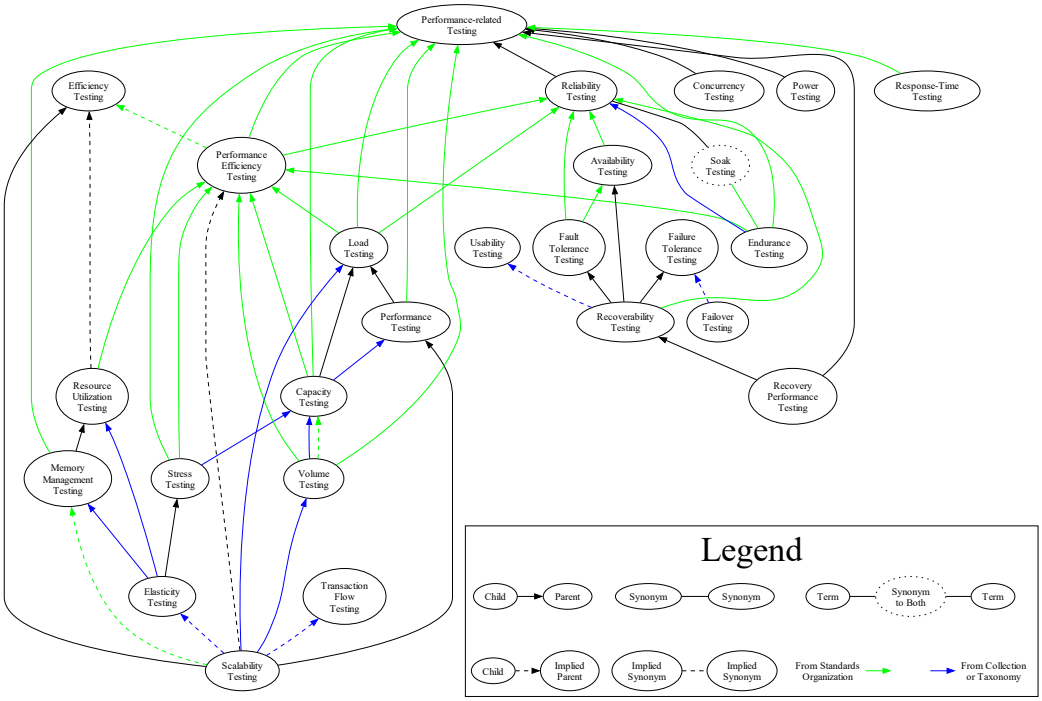

Fig. 6. Proposed relations between rationalized "performance-related testing" terms.

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 –Software testing –Part 1: General concepts," ISO/IEC/IEEE 29119-1:2022(E), Jan. 2022.

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

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

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

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

[10] N. E. Fenton and S. L. Pfleeger, Software Metrics: A Rigorous & Practical Approach, 2nd ed. Boston, MA, USA: PWS Publishing Company, 1997.

[11] 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

[12] 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

[13] 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

[14] 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

[15] 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

[16] 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

[17] 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

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

[19] 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
[20] 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
[21] ISO/IEC and IEEE, "ISO/IEC/IEEE International Standard - Software and systems engineering –Software testing –Part 4: Test techniques," ISO/IEC/IEEE 29119-4:2021(E), Oct. 2021.
[22] ——, "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.
[23] ——, "ISO/IEC/IEEE International Standard - Software and systems engineering –Software testing –Part 5: Keyword-Driven Testing," ISO/IEC/IEEE 29119-5:2016, Nov. 2016.
[24] ——, "ISO/IEC/IEEE International Standard - Systems and software engineering –Software testing –Part 1: General concepts," ISO/IEC/IEEE 29119-1:2013, Sep. 2013.
[25] IEEE, "IEEE Standard for System and Software Verification and Validation," IEEE Std 1012-2012 (Revision of IEEE Std 1012-2004), 2012.
[26] 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
[27] ——, "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
[28] 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
[29] ——, "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
[30] ——, "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] ——, "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
[32] ——, "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
[33] 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
[34] 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
[35] 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
[36] W. E. Perry, Effective Methods for Software Testing, 3rd ed. Indianapolis, IN, USA: Wiley Publishing, Inc., 2006.
[37] 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
[38] J. Peters and W. Pedrycz, Software Engineering: An Engineering Approach, ser. Worldwide series in computer science. John Wiley & Sons, Ltd., 2000.
[39] H. van Vliet, Software Engineering: Principles and Practice, 2nd ed. Chichester, England: John Wiley & Sons, Ltd., 2000.
[40] 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
[41] ChatGPT (GPT-4o), "Defect Clustering Testing," Nov. 2024. [Online]. Available: https://chatgpt.com/share/67463dd1-d0a8-8012-937b-4a3db0824dcf
[42] LambdaTest, "What is Operational Testing: Quick Guide With Examples," 2024. [Online]. Available: https://www.lambdatest.com/learning-hub/operational-testing
[43] P. Pandey, "Scalability vs Elasticity," Feb. 2023. [Online]. Available: https://www.linkedin.com/pulse/scalability-vs-elasticity-pranav-pandey/
[44] 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
[45] 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
[46] U. Kanewala and T. Yueh Chen, "Metamorphic testing: A simple yet effective approach for testing scientific software," Computing in Science & Engineering, vol. 21, no. 1, pp. 66–72, 2019.
[47] 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
[48] 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
[49] 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
[50] 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.
[51] 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.
[52] 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
[53] 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

[54] 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

[55] 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

[56] 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

[57] 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

[58] V. Rus, S. Mohammed, and S. G. Shiva, "Automatic Clustering of Defect Reports," in Proceedings of the Twentieth International Conference on Software Engineering & Knowledge Engineering (SEKE 2008). San Francisco, CA, USA: Knowledge Systems Institute Graduate School, Jul. 2008, pp. 291–296. [Online]. Available: https://core.ac.uk/download/pdf/48606872.pdf

[59] 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.

[60] 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

[61] 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.

[62] 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.

[63] 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

[64] 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.

[65] 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

[66] 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

[67] 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

[68] 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

[69] 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.

[70] 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.

[71] W. Goralski, "xDSL loop qualification and testing," IEEE Communications Magazine, vol. 37, no. 5, pp. 79–83, 1999.

[72] 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

[73] 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

[74] 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

[75] 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

[76] 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