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—Despite the prevalence and importance of software testing, it lacks a standardized and consistent taxonomy, instead relying on a large body of literature with many flaws—even within individual documents! This hinders precise communication, contributing to misunderstandings when planning and performing testing. In this paper, we explore the current state of software testing terminology by:

1) identifying established standards and prominent testing resources,
2) capturing relevant testing terms from these sources, along with their definitions and relationships (both explicit and implicit), and
3) constructing graphs to visualize and analyze these data.

This process uncovers 561 test approaches and 77 software qualities that may imply additional related test approaches. We also build a tool for generating graphs that illustrate relations between test approaches and track flaws captured by this tool and manually through the research process. This reveals 307 flaws, including nine terms used as synonyms to two (or more) disjoint test approaches and 18 pairs of test approaches that may either be synonyms or have a parent-child relationship. This also highlights notable confusion surrounding functional testing, recovery resting, and scalability testing. Our 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

As with all fields of science and technology, software development should be approached systematically and rigorously. Reference [1] claims that "to be successful, development of software systems requires an engineering approach" that is "characterized by a practical, orderly, and measured development of software" [p. 3]. When a NATO study group decided to hold a conference to discuss "the problems of software" in 1968, they chose the phrase "software engineering" to "imply[] the need for software manufacture to be based on the types of theoretical foundations and practical disciplines, [sic] that are traditional in the established branches of engineering" [2, p. 13]. "The term was not in general use at that time", but conferences such as this "played a major role in gaining general acceptance … for the term" [3]. While one of the goals of the conference was to "discuss possible techniques, methods and developments which might lead to the[] solution" to these problems [2, p. 14], the format of the conference itself was difficult to document. Two competing classifications of the report emerged: "one following from the normal sequence of steps in the development of a software product" and "the other related to aspects like communication, documentation, management, [etc.]" [p. 10]. Furthermore, "to retain the spirit and liveliness of the conference, … points of major disagreement have been left wide open, and … no attempt … [was] made to arrive at a consensus or majority view" [p. 11]!

Perhaps unsurprisingly, there are still concepts in software engineering without consensus, and many of them can be found in the subdomain of software testing. Reference [4] gives the example of complete testing, which may 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 [p. 7]. Having a clear definition of "complete testing" would reduce the chance for miscommunication and, ultimately, the tester getting "blamed for not doing … [their] job" [p. 7]. Because software testing uses "a subtantial percentage of a software development budget (in the range of 30 to 50%)", which is increasingly true "with the growing complexity of software systems" [1, p. 438], this is crucial to the efficiency of software development. Even more foundationally, 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. [5] 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. [6] prioritize organizing test approaches over defining them,
- Firesmith [7] similarly defines relations between test approaches but not the approaches themselves, and
- Unterkalmsteiner et al. [8] 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

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

survey of [either domain]" [p. A:2].

In addition to these taxonomies, many standards documents (see Section II-E1) and terminology collections (see Section II-E2) define testing terminology, albeit with their own issues.

For example, a common point of discussion in the field of software is the distinction between terms for when software does not work correctly. We find the following four to be most prevalent:

- Error: "a human action that produces an incorrect result" [9, p. 128], [10, p. 399].
- Fault: "an incorrect step, process, or data definition in a computer program" [9, p. 140] inserted when a developer makes an error [9, pp. 128, 140], [10, pp. 399–400], [11, p. 12-3].
- Failure: the inability of a system "to perform a required function or … within previously specified limits" [12, p. 7], [13] that is "externally visible" [12, p. 7] and caused by a fault [10, p. 400], [11, p. 12-3].
- Defect: "an imperfection or deficiency in a project component where that component does not meet its requirements or specifications and needs to be either repaired or replaced" [9, p. 96].

This distinction is sometimes important, but not always [14, p. 4-3]. The term "defect" is "overloaded with too many meanings, as engineers and others use the word to refer to all different types of anomalies" [15, p. 12-3] (similar in [9, p. 96], [16, p. 124]). Software testers may even choose to ignore these nuances completely! Reference [17, pp. 13–14] "just call[s] it what it is and get[s] on with it", abandoning these four terms, "problem", "incident", "anomaly", "variance", "inconsistency", "feature" (!), and "a list of unmentionable terms" in favour of "bug"; after all, "there's no reason to dice words"!

But why are minor differences between terms like these even important? The previously defined terms "error", "fault", "failure", and "defect" are used to describe many test approaches, including:

1) Defect-based testing
2) Error forcing
3) Error guessing
4) Error tolerance testing
5) Error-based testing
6) Error-oriented testing
7) Failure tolerance testing
8) Fault injection testing
9) Fault seeding
10) Fault sensitivity testing
11) Fault tolerance testing
12) Fault tree analysis
13) Fault-based testing

When considering which approaches to use or when actually using them, the meanings of these four terms inform what their related approaches accomplish and how to they are performed. For example, the tester needs to know what a "fault" is to perform fault injection testing; otherwise, what would they inject? Information such as this is critical to the testing team, and should therefore be standardized.

These kinds of inconsistencies can lead to miscommunications—such as that previously mentioned by [4, p. 7]—and are prominent in the literature. ISO/IEC and IEEE categorize experience-based testing as both a test design technique and a test practice in the same figure [18, Fig. 2]! The structure of tours can be defined as either quite general [18, p. 34] or "organized around a special focus" [19]. Load testing is performed with loads "between anticipated conditions of low, typical, and peak usage" [18, p. 5] or loads that are as large as possible [17, p. 86]. Alpha testing is performed by "users within the organization developing the software" [16, p. 17], "a small, selected group of potential users" [15, p. 5-8], or "roles outside the development organization" conducted "in the developer's test environment" [19]. It is clear that there is a notable gap in the literature, one which we attempt to describe. 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. The main way we accomplish this is by identifying "flaws" or "inconsistencies" in the literature, or areas where there is room for improvement. We track these flaws according to both what information is wrong and how (described in more detail in Section II-B), which allows us to analyze them more thoroughly and reproducibly.

Based on this observed gap in software testing terminology and our original motivation for this research, we only consider the component of Verification and Validation (V&V) that tests code itself. However, some test approaches are only used to testing other artifacts, while others can be used for both! In these cases, we only consider the subsections that focus on code. For example, reliability testing and maintainability testing can start without code by "measur[ing] structural attributes of representations of the software" [20, p. 18], but only reliability and maintainability testing performed on code itself is in scope of this research.

This document describes our process, as well as our results, in more detail. We start by documenting the 561 test approaches mentioned by 75 sources (described in Section II-E), recording their names, categories[1], definitions, synonyms[2], parents[3], and flaws[4] as applicable. We also record any other relevant notes, such as prerequisites, uncertainties, other sources. We follow the procedure laid out in Section III and use these Research Questions (RQs) as a guide:

1) What testing approaches do the literature describe?
2) Are these descriptions consistent?
3) Can we systematically resolve any of these inconsistencies?

[1]Defined in Section II-A1.
[2]Defined in Section II-A2.
[3]Defined in Section II-A3.
[4]Defined in Section II-B.

We then create tools to support our analysis of our findings (Section III-G). Despite the amount of well understood and organized knowledge, the literature is still quite flawed (Section IV). This reinforces the need for a proper taxonomy! We then provide some potential solutions covering some of these flaws (Section V).

II. Terminology

Our research aims to describe the current state of software testing literature, including its flaws. Since we critique the lack of clarity, consistency, and robustness in the literature, we need to hold ourselves to a high standard in these areas when defining and using terms. For example, since we focus on how the literature describes "test approaches", we first define this term (Section II-A). Likewise, before we can constructively describe the flaws in the literature, we need to define what we mean by "flaw" (Section II-B). To further prevent bias, we only use classifications and relations already implicitly present in the literature instead of inventing our own; for example, test approaches can have categories (Section II-A1), synonyms (Section II-A2), and parent-child relations (Section II-A3). We also observe flaws having both manifestations (Section II-B1) and domains (Section II-B2) and use these terms to refer to these implicit concepts in the literature. All of these classifications and relations follow logically from the literature and as such are technically "results" of our research, but we define them here for clarity since we use them throughout this paper.

Since the literature is flawed, we need to be careful with what information we take at face value. We do this by tracking the nuance, or "explicitness", of information (Section II-C) found in sources and the "credibility" of these sources themselves (Section II-D). We then use this heuristic of credibility to group our identified sources into "tiers" (Section II-E). Defining these terms helps reduce the effect of our preconceptions on our analysis (or at least makes it more obvious to future researchers), as there may be other equally valid ways to analyze the literature and its flaws. To be clear: we do not prescribe what terminology software testers should use, we simply observe the terminology that the literature uses and try to use it as consistently and logically as possible.

A. Test Approaches

Software testing is defined as the "process of operating a system or component under specified conditions, observing or recording the results, and making an evaluation of some aspect of the system or component" [23]. For each test, the main steps are to:

1) identify the goal(s) of the test,
2) decide on an approach,
3) develop the tests,
4) determine the expected results,
5) run the tests, and
6) compare the expected results to the actual results [1, p. 443].

The end goal is to evaluate "some aspect of the system or component" based on the results of step 6 [18, p. 10], [21, p. 6], [23]. When this evaluation reveals errors, "the faults causing them are what can and must be removed" [11, p. 5-3].

Of course, the approach chosen in step 2 influences what kinds of test cases should be developed and executed in later steps, so it is important that test approaches are defined correctly, consistently, and unambiguously. A "test approach" is a "high-level test implementation choice" [18, p. 10] used to "pick the particular test case values" [16, p. 465] used in step 5. The only approach that can "fully" test a system (exhaustive testing) is infeasible in most non-trivial situations [1, pp. 439, 461], [10, p. 421], [11, p. 5-5], [18, p. 4], so multiple approaches are needed [p. 18] to "suitably cover any system" [p. 33]. This is why this process should be repeated and there are so many test approaches described in the literature.

1) Approach Categories: Since there are so many test approaches, it is helpful to categorize them. The literature provides many ways to do so, but we use the one given by ISO/IEC and IEEE [18] because of its wide usage. This schema divides test approaches into levels, types, techniques, and practices [18, Fig. 2; see Table I]. These categories seem to be pervasive throughout the literature, particularly "level" and "type". For example, six non-IEEE sources also give unit testing, integration testing, system testing, and acceptance testing as examples of test levels [1, pp. 443–445], [11, pp. 5-6 to 5-7], [19], [24, pp. 807–808], [25, pp. 9, 13], [26, p. 218]. These categories seem to be orthogonal based on their definitions and usage. For example, "a test type can be performed at a single test level or across several test levels" [18, p. 15], [21, p. 7], [22, p. 8], and test practices can be "defined … for a specific level or type of testing" [27, p. 9]. We may assess this assumption more rigorously in the future, but for now, it implies that one can derive a specific test approach by combining multiple test approaches from different categories. We loosely describe these categories based on what they specify as follows:

- Level: What code is tested
- Practice: How the test is structured and executed
- Technique: How inputs and/or outputs are derived
- Type: Which software quality is evaluated

While [18]'s schema includes "static testing" as a test approach category, we omit it from our scope as it seems non-orthogonal to the others and thus less helpful for grouping test approaches. We also introduce a "supplemental" category of "artifact" since some terms can refer to both the application of a test approach and the resulting document(s). Therefore, we do not consider approaches categorized as an artifact and another category as flaws in Section IV-B1. Finally, we also record the test category

TABLE I: Categories of test approaches given by ISO/IEC and IEEE.

Term	Definition	Examples
Test Level[a]	A stage of testing "typically associated with the achievement of particular objectives and used to treat particular risks", each performed in sequence [18, p. 12], [21, p. 6], [22, p. 6] with their "own documentation and resources" [16, p. 469]	unit/component testing, integration testing, system testing, acceptance testing [16, p. 467], [18, p. 12], [21, p. 6], [22, p. 6]
Test Practice	A "conceptual framework that can be applied to … [a] test process to facilitate testing" [16, p. 471], [18, p. 14]	scripted testing, exploratory testing, automated testing [18, p. 20]
Test Technique[b]	A "procedure used to create or select a test model …, identify test coverage items …, and derive corresponding test cases" [18, p. 11], [22, p. 5] (similar in [16, p. 467]) that "generate evidence that test item requirements have been met or that defects are present in a test item" [21, p. vii] "typically used to achieve a required level of coverage" [22, p. 5]	equivalence partitioning, boundary value analysis, branch testing [18, p. 11], [22, p. 5]
Test Type	"Testing that is focused on specific quality characteristics" [16, p. 473], [18, p. 15], [21, p. 7]	security testing, usability testing, performance testing [16, p. 473], [18, p. 15], [22, p. 8]

[a] Also called "test phase" or "test stage" (see relevant synonym flaws in Section IV-B2).
[b] Also called "test design technique" [18, p. 11], [22, p. 5], [19].

of "process" for completeness, although this seems to be a higher level classification and these approaches will likely be excluded during later analysis.

2) Synonym Relations: The same approach often has many names. For example, "specification-based testing" is also called "black-box testing" [7, pp. 46–47[5]], [10, p. 399], [11, p. 5-10], [16, p. 431], [18, p. 9], [19], [21, p. 8], [28, p. 344]. Throughout our work, we use the terms "specification-based testing" and "structure-based testing" to articulate the source of the information for designing test cases, but a team or project also using grey-box testing may prefer the terms "black-box" and "white-box testing" for consistency.

We can formally define the synonym relation S on the set T of terms used by the literature to describe test approaches based on how synonyms are used in natural language. S is symmetric and transitive, and although pairs of synonyms in natural language are implied to be distinct, a relation that is symmetric and transitive is provably reflexive; this implies that all terms are trivially synonyms of themselves. Since S is symmetric, transitive, and reflexive, it is an equivalence relation, reflecting the role of synonyms in natural language where they can be used interchangeably. While synonyms may emphasize different aspects or express mild variations, their core meaning is nevertheless the same.

3) Parent-Child Relations: Many test approaches are multi-faceted and can be "specialized" into others; for example, load testing and stress testing are some subtypes of performance-related testing (described in more detail in Section V-C). We refer to these "specializations" as "children" or "subapproaches" of their multi-faceted "parent(s)". This nomenclature also extends to approach categories (such as "subtype"; see Section II-A1 and Table I) and software qualities ("subquality").

[5]Reference [7] excludes the hyphen, calling it "black box testing".

We can formally define the parent-child relation P on the set T of terms used by the literature to describe test approaches based on directed relations between approach pairs. This relation should be irreflexive, asymmetric, and transitive, making it a strict partial order. A consequence of this is that there should be no directed cycles, although since a given child approach c may have more than one parent approach p, undirected cycles may exist.

Parent-child relations often manifest when a "well-understood" test approach p is decomposed into smaller, independently performable approaches $c_1, \ldots, c_n$, each with its own focus or nuance. This is frequently the case for hierarchies of approaches given in the literature [7], [18, Fig. 2], [21, Fig. 2]. Another way for these relations to occur is when the completion of p indicates that "sufficient testing has been done" in regards to c [10, p. 402]. While this only "compares the thoroughness of test techniques, not their ability to detect faults" [p. 434], it is sufficient to justify a parent-child relation between the two approaches. These relations may also be represented as hierarchies [Fig. 13.17], [21, Fig. F.1].

B. Flaws

Ideally, software testing literature would describe test approaches correctly, completely, consistently, and modularly, but this is not the case in reality. We use the term "flaw" to refer to any instance of the literature violating these ideals, not to instances of software doing the same[6]. We classify flaws by both their "manifestations"

[6]When picking a word to describe these violations, we wanted to avoid words that are "overloaded with too many meanings" like "error" and "fault" [11, p. 12-3; see Section I for more detailed discussion]. A small literature review revealed that established standards (see Section II-E1) use the term "flaw" to refer to requirements [18, p. 38], design [p. 43], "system security procedures … and internal controls" [29, p. 194], but only once to refer to code itself [p. 92]. Reference [11, p. 7-9] also uses the word "flaw" in a way that aligns with our goal of analyzing what is wrong with software engineering's testing literature.

TABLE II: Observed flaw manifestations.

Manifestation	Description	Key
Mistake	Information is incorrect	WRONG[a]
Omission	Information that should be included is not	MISS[b]
Contradiction	Information from multiple places conflicts	CONTRA
Ambiguity	Information is unclear	AMBI
Overlap	Information is nonatomic or used in multiple contexts	OVER
Redundancy	Information is redundant	REDUN

[a] We use WRONG here to avoid clashing with MISS.
[b] We use MISS here to be more meaningful in isolation, as it implies the synonym of "missing"; OMISS is less intuitive and OMIT would be inconsistent with the keys being adjective-based.

TABLE III: Observed flaw domains.

Domain	Description	Key
Categories	Approach categories, defined in Section II-A1	CATS
Synonyms	Synonym relations, defined in Section II-A2	SYNS
Parents	Parent-child relations, defined in Section II-A3	PARS
Definitions	Definitions given to terms	DEFS
Labels	Labels or names given to terms	LABELS
Scope	Scope of the information	SCOPE
Traceability	Records of the source(s) of information	TRACE

(how information is wrong; see Section II-B1) and their "domains" (what information is wrong; see Section II-B2). These are orthogonal classifications, since each flaw manifests in a particular domain, which we track by assigning each flaw one "key" for each classification. We list these keys in Tables II and III, respectively, and use them to count and analyze the flaws we uncover in Section IV. We also introduce some terms to be used when discussing flaws based on how many sources contribute to them (Section II-B3).

1) Flaw Manifestations: Perhaps the most obvious example of something being "wrong" with the literature is that a piece of information it presents is incorrect—"wrong" in the literal sense. However, if our standards for correctness require clarity, consistency, and robustness, then there are many ways for a flaw to manifest. This is one view we take when observing, recording, and analyzing flaws: how information is "wrong". We observe the "manifestations" described in Table II throughout the literature, and give each a unique key for later analysis and discussion. We list them in descending order of severity, although this is partially subjective. While some may disagree with our ranking, it is clear that information being incorrect is worse than it being repeated. Our ordering has the benefit of serving as a "flowchart" for classifying flaws. For example, if a piece of information is not intrinsically incorrect, then there are five remaining manifestation types for the flaw. Note that some flaws involve information from multiple sources (contradictions and overlaps in particular). We do not categorize these flaws as "mistakes" if finding the ground truth requires analysis that has not been performed yet.

2) Flaw Domains: Another way to categorize flaws is by what information is wrong, which we call the flaw's "domain". We describe those we observe in Table III, and tracking these uncovers which knowledge domains are less standardized (and should therefore be approached with more rigour) than others. We explicitly define some of these domains in previous sections and thus present them in that same order. These are the domains in which we automatically detect flaws as described in ?? and present in Section IV-B, so these are the only ones that are hyperlinked. We automatically detect the following classes of flaws:

- Test approaches with more than one category that violate our assumption of orthogonality (see Section II-A1)
- Synonyms that violate transitivity (see Section II-A2); if two distinct approaches share a synonym but are not synonyms themselves, at least one relation is incorrect or missing.
- Synonyms between independently defined approaches; if two separate approaches have their own definitions, nuances, etc. but are also labelled as synonyms, this indicates that:
  1) the terms are interchangeable and this relation is trivially reflexive (see Section II-A2),
  2) at least one of these terms is defined incorrectly, and/or
  3) this synonym relation is incorrect.
- Parent-child relations that violate irreflexivity, or approaches that are parents of themselves.
- Pairs of synonyms where one is a subapproach of the other; these relations cannot coexist as synonym relations are symmetric while parent-child relations are asymmetric as outlined in Sections II-A2 and II-A3, respectively.

Despite their nuance, the remaining domains are relatively straightforward, so we define them briefly as follows instead of defining them more rigorously in their own sections. Terms can be thought of as definition-label pairs, but there is a meaningful distinction between definition flaws and label flaws. Definition flaws are quite self-explanatory, but label flaws are harder to detect, despite occurring independently. Examples of label flaws include terms that share the same acronym or contain typos or redundant information. Sometimes, an author may use one term when they mean another. One could argue that their "internal" definition of the term is the cause of

this mistake, but we consider this a label flaw where the wrong label is used as we would change the label to fix it. Additionally, some information is presented with an incorrect scope and sometimes should not have been included at all! Finally, some traceability information is flawed, such as how one document cites another or even what information is included within a document.

3) Additional Flaw Terminology: Some flaws involve information from more than one source, but referring to this as a "flaw between two sources" is awkward. We instead refer to this kind of flaw as an "inconsistency" between the sources. This clearly indicates that there is disagreement between the sources, but also does not imply that either one is correct—the inconsistency could be with some ground truth if neither source is correct!

Other flaws only involve one source, but we make a distinction between "self-contained" flaws and "internal" flaws. Self-contained flaws are those that manifest by comparing a document to an assertion of ground truth. These may appear once in a document or consistently throughout it. Sometimes, these do not require an explicit comparison to ground truth; these often include omissions as the lack of information is contained within a single source and does not need to be cross-checked against an assertion of ground truth. On the other hand, internal flaws arise when a document disagrees with itself by containing two conflicting pieces of information; this includes many contradictions and overlaps. Internal flaws can even occur on the same page, such as when a source gives the same acronym to two distinct terms!

C. Explicitness

When information is written in natural language, a considerable degree of nuance can get lost when interpreting or using it. We call this nuance "explicitness", or how explicit a piece of information is (or is not). For example, a source may provide data from which the reader can logically draw a conclusion, but may not state or prove this conclusion explicitly. In the cases where information is not explicit, we record it (see Section III-D for more detailed discussion) and present it using (at least) one of the following keywords: "implied", "can be", "sometimes", "should be", "ideally", "usually", "most", "likely", "often", "if", and "although". Most information provided by sources we investigate is given explicitly; all sources cited throughout this paper support their respective claims explicitly unless specified otherwise, usually via one of these keywords. It is important to note that we use the term "implicit" (as well as "implied by" when describing sources of information) to refer to any instance of "not explicit" information for brevity. Any kind of information can be implicit, including the names, definitions, categories (see Section II-A1), synonyms (see Section II-A2), and parents (see Section II-A3) of identified test approaches.

As our research focuses on the flaws present in the literature, the explicitness of information affects how seriously we take it. We call flaws based on explicit information "objective", since they are self-evident in the literature. On the other hand, we call flaws based on implicit information "subjective", since some level of judgement is required to assess whether these flaws are actually problematic. By looking for the indicators of uncertainty mentioned above, we can automatically detect subjective flaws when generating graphs and performing analysis (see Section III-G).

D. Credibility

In the same way we distinguish between the explicitness of information from different sources, we also wish to distinguish between the "explicitness" of the sources themselves! Of course, we do not want to overload terms, so we define a source as more "credible" if it:

- has gone through a peer-review process,
- is written by numerous, well-respected authors,
- cites a (comparatively) large number of sources, and/or
- is accepted and used in the field of software.

Sources may meet only some of these criteria, so we use our judgement (along with the format of the sources themselves) when comparing them.

E. Source Tiers

For ease of discussion and analysis, we group the complete set of sources into "tiers" based on their format, method of publication, and our heuristic of credibility. In order of descending credibility, we define the following tiers:

1) established standards (Section II-E1),
2) terminology collections (Section II-E2),
3) textbooks (Section II-E3), and
4) papers and other documents (Section II-E4).

We provide a summary of how many sources comprise each tier in Figure 1. The "papers" tier is quite large since we often "snowball" on terminology itself when a term requires more investigation (e.g., its definition is missing or unclear). This includes performing a miniature literature review on this subset to "fill in" missing information (see Section III-E) and potentially fully investigating these additional sources, as opposed to just the original subset of interest, based on their credibility and how much extra information they provide. We use standards the second most frequently due to their high credibility and broad scope; for example, the glossary portion of [16] has 514 pages! Using these standards allows us to record many test approaches in a similar context from a source that is widely used and well-respected.

1) Established Standards: These are documents written for the field of software engineering by reputable standards bodies, namely ISO, the International Electrotechnical Commission (IEC), and IEEE, so we consider them to be the most credible sources. Their purpose is to "encourage the use of systems and software engineering

standards" and "collect and standardize terminology" by "provid[ing] definitions that are rigorous, uncomplicated, and understandable by all concerned" [16, p. viii] "that can be used by any organization when performing any form of software testing" [18, p. vii]. Only standards for software development and testing are in scope for this research (see Section I for a high-level overview of what is in scope).

2) Terminology Collections: These are collections of software testing terminology built up from multiple sources, such as the established standards outlined in Section II-E1. For example, 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" [4, p. xix]. Even though it is "published by the IEEE Computer Society", it "reflects the current state of generally accepted, consensus-driven knowledge derived from the interaction between software engineering theory and practice" [30]. Due to this combination of IEEE standards and state-of-the-practice observations, we designate it as a collection of terminology as opposed to an established standard. Collections such as this are often written by a large organization, such as the International Software Testing Qualifications Board (ISTQB), but not always. Firesmith [7]'s taxonomy presents relations between many test approaches and Doğan et al. [31]'s literature review cites many sources from which we can "snowball" if desired (see Section III-A), so we include them in this tier as well.

3) Textbooks: We consider textbooks to be more credible than papers (see Section II-E4) because they are widely used as resources for teaching software engineering and industry frequently uses them as guides. Although textbooks have smaller sets of authors, they follow a formal review process before publication. Textbooks used at McMaster University [1], [10], [17] served as the original (albeit ad hoc and arbitrary) starting point of this research, and we investigate other books as they arise. For example, [19] cites [32] as the original source for its definition of "scalability" (see Section V-B) which we verify by looking at this original source.

4) Papers and Other Documents: The remaining documents all have much smaller sets of authors and are much less widespread than those in higher source tiers. While most of these are journal articles and conference papers, we also include the following document types. Some of these are not peer-reviewed works but are still useful for observing how terms are used in practice:

- Report [25], [33], [34]
- Thesis [35]
- Website [36], [37]
- Booklet [38]
- ChatGPT [39] with its claims supported by [40]

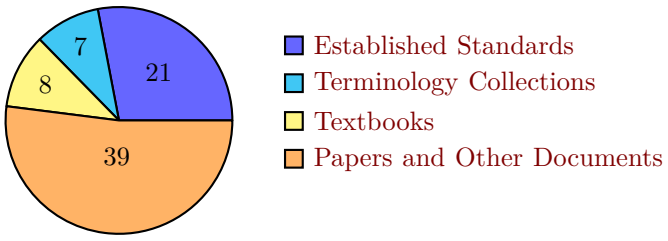


Fig. 1: Summary of how many sources comprise each source tier.

III. Methodology

We collect data from a wide variety of documents related to software testing, focusing on test approaches and supporting information. This results in a large glossary of software approaches, some glossaries of supplementary terms, and a list of flaws. To ensure this data can be analyzed and expanded thoroughly and consistently, we need a process that can be repeated for future developments in the field of software testing or by independent researchers seeking to verify our work. Our methodology is as follows:

1) Identify authoritative sources (Section III-A)
2) Identify all test approaches[7] and testing-related terms (Section III-B) described in these authoritative sources
3) Record all relevant data (Section III-C), including implicit data (Section III-D), for each term identified in step 2; test approach data are comprised of:
   a) Names
   b) Categories[8]
   c) Definitions
   d) Synonyms[9]
   e) Parents[10]
   f) Flaws[11]
   g) Other relevant notes (prerequisites, uncertainties, other sources, etc.)
4) Repeat steps 1 to 3 for any missing or unclear terms (Section III-E) until the stopping criteria (Section III-F) is reached

A. Identifying Sources

As there is no single authoritative source on software testing terminology, we need to look at many sources to observe how this terminology is used in practice. We start from the vocabulary document for systems and software engineering [16] and three versions of the Guide to the SoftWare Engineering Body Of Knowledge (SWEBOK Guide) ([14]; [11]; [15]; see Section IV). To gather further sources, we then use a version of "snowball sampling", which "is commonly used to locate hidden populations … [via] referrals from initially sampled respondents to other persons" [41]. We apply this concept to "referrals" between sources. For example, [19] cites [32] as the original source

[7]Defined in Section II-A.
[8]Defined in Section II-A1.
[9]Defined in Section II-A2.
[10]Defined in Section II-A3.
[11]Defined in Section II-B.

for its definition of "scalability" (see Section V-B) which we verify by looking at this original source. We group all sources into the source tiers we define in Section II-E.

B. Identifying Relevant Terms

Before we can consistently track software testing terminology used in the literature, we must first determine what to record. We use heuristics to guide this process to increase confidence that we identify all relevant terms, paying special attention to the following when investigating a new source:

- glossaries, taxonomies, hierarchies, 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 test approaches, especially those that were ambiguous or prompted further discussion (e.g., terms containing "performance", "recovery", "component", "bottom-up", or "configuration"), and
- terms that imply test approaches, including:
  - coverage metrics that may imply test techniques,
  - software qualities that may imply test types, and
  - requirements that may imply test approaches.

C. Recording Relevant Information

Once we have identified which terms from the literature are relevant, we can then track them consistently by building glossaries. We give each test approach its own row in our test approach glossary, recording its name and any given definitions, categories, synonyms, and parents (along with any other notes, such as questions, prerequisites, and other resources to investigate) following the procedure in ??. Note that only the name and category fields are required; all other fields may be left blank, although a lack of definition indicates that the approach should be investigated further to see if its inclusion is meaningful (see Section III-E). Flawed data may be documented here as dubious information (see Section III-D).

For example, when we first encounter "A/B Testing" in [18, p. 1, 36] as shown in Figure 2, we apply our procedure as follows:

1) Create a new row with the name "A/B testing" and the category "Approach".
2) Record the synonym "Split-Run Testing".
3) Record the parent "Statistical Testing".
4) Record the definition "Testing 'that allows testers to determine which of two systems or components performs better' "; note that we abstract away information that we have previously captured (i.e., its synonym and parent).

We also include the relevant citation in the author-year citation format with these entries (excluding its name) to track where they came from. This source also provides the following information, which we capture as follows:

1) Record the note "It 'can be time-consuming, although tools can be used to support it', 'is a means of solving the test oracle problem by using the existing system as a partial oracle', and is 'not a test case generation technique as test inputs are not generated' " [p. 36].
2) Replace the category of "Approach" with the more specific "Practice" [Fig. 2]; note that this is consistent with the exclusion of "Technique" as a possible category for this approach [p. 36].

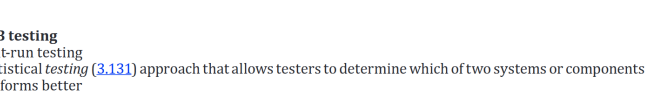


Fig. 2: Reference [18, p. 1]'s glossary entry for "A/B testing".

As we investigate other sources, we learn more about this approach. Firesmith [7, p. 58] includes it in his taxonomy as shown in Figure 3. We add to our entry for "A/B Testing" as follows:

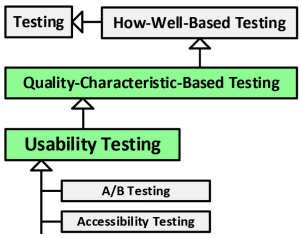


Fig. 3: A/B testing's inclusion in Firesmith [7, p. 58]'s taxonomy.

1) Add the parent "Usability Testing".
2) Since usability testing is a test type [18, pp. 22, 26-27], [21, pp. 7, 40, Tab. A.1], add the category "Type" with the citation "(inferred from usability testing)".

This second change introduces an inference that violates our assumption that categories are orthogonal, so we consider this to be an inferred flaw that we automatically detect and document (details omitted for brevity).

We use this same procedure to track software qualities and supplementary terminology that is either shared by multiple approaches or too complicated to explain inline. We create a separate glossary for both qualities and supplementary terms, each with a similar format to our test approach glossary. Since these terms do not have categories, the process of recording them is much simpler, only requiring us to record the name, definition, and synonym(s) of these terms, along with any additional notes. The only new information we capture is the "precedence" for a software quality to have an associated test type, since each test type measures a particular software quality (see Table I). These precedences are instances where a given software quality is related to, is covered by, or is a child, parent, or prerequisite of another quality with an associated test type, as given by the literature.

Tracking information about software qualities helps us investigate the literature more thoroughly, since these data may become relevant based on information from other yet-uninvestigated sources. When the literature mentions (or implies) a test approach that corresponds to a software

quality we have recorded, we first follow the procedure given in ?? with the information provided in the source that mentions it. We then remove the relevant data from our quality glossary and repeat our procedure with it to upgrade the quality to a test type in our test approach glossary.

D. Recording Implicit Information

As described in Section II-C, the use of natural language introduces significant nuance that we need to document. Keywords such as "implied", "can be", "sometimes", "should be", "ideally", "usually", "most", "likely", "often", "if", and "although" indicate that information from the literature is not explicit. These keywords often appear directly within the literature, but even when they do not, we use them to track explicitness in our test approach glossary to provide a more complete summary of the state of software testing literature without getting distracted by less relevant details. We find the following non-mutually exclusive cases of implicit information from the literature:

1) The information follows logically from the source and information from others, but is not explicitly stated.
2) The information is not universal but still applies in certain cases.
3) The information is dubious; while it is present in the literature, there is reason to doubt its accuracy.

When we encounter information that meets one of these criteria, we use an appropriate keyword to capture this nuance in our test approach glossary. This also helps us identify implicit information when performing later analysis. Despite "implicit" only describing the first of these cases, we use it (as well as "implied by" when describing sources of information) as a shorthand for all "not explicit" manifestations throughout this paper for clarity.

E. Undefined Terms

The literature mentions many software testing terms without defining them. While this includes test approaches, software qualities, and more general software terms, we focus on the former as the main focus of our research. In particular, [7] and [18] name many undefined test approaches. Once we exhaust the standards in Section II-E1, we perform miniature literature reviews on these subsets to "fill in" the missing definitions (along with any relations), essentially "snowballing" on these terms as described in Section III-A. This process uncovers even more approaches, on which we can then repeat this process.

F. Stopping Criteria

Unfortunately, continuing to look for test approaches indefinitely is infeasible. We therefore need a "stopping criteria" to let us know when we are "finished" looking for test approaches in the literature. A reasonable heuristic is to repeat step 4 until it yields diminishing returns; i.e., investigating new sources does not reveal new approaches, relations between them, or information about them. This implies that something close to a complete taxonomy has been achieved!

G. Tools

To better understand the relations between test approaches, we develop a tool to visualize them. However, since each term is trivially a synonym of itself and there are many non-problematic synonyms that do not imply flaws (see Section II-A2), we only visualize the synonym relations that may indicate flaws given in Section II-B2; i.e., intransitive synonyms and synonyms between independently defined approaches. These visualizations tend to be large, so it is often useful to focus on specific subsets of them. We can generate more focused visualizations from a given subset of approaches, such as those in a selected approach category (defined in Section II-A1) or those pertaining to recovery testing. We use these visualizations to better understand the relations within these subsets of approaches, but we can also update them based on our recommendations by specifying sets of approaches and relations to add or remove. When doing so in Section V, we colour any proposed approaches or relations orange to distinguish them.

IV. Observed Flaws

After gathering all these data[12], we find many flaws. ?? shows where these flaws appear within each source tier (see Section II-E) which reveals a lot about software testing literature:

1) Established standards (Section II-E1) aren't actually standardized, since:
   a) other documents disagree with them very frequently and
   b) they are the most internally inconsistent source tier!
2) Less standardized (or "credible"; see Section II-D) documents, such as terminology collections and textbooks (Sections II-E2 and II-E3, respectively) are also not followed to the extent they should be.
3) Documents across the board have flaws within the same document, between documents with the same author(s), or even with reality!

To better understand and analyze these flaws, we group them by their manifestations and their domains as defined in Section II-B. We present the total number of flaws by manifestation and by domain in Tables IV and V, respectively, where a given row corresponds to the number of flaws either within that source tier and/or with a "more credible" one (i.e., a previous row in the table; see Sections II-D and II-E). We also group these flaws by their explicitness (defined in Section II-C) by counting (Obj)ective and (Sub)jective flaws separately,

[12] Available in ApproachGlossary.csv, QualityGlossary.csv, and SuppGlossary.csv at https://github.com/samm82/TestingTesting.

TABLE IV: Breakdown of identified flaws by manifestation and source tier.

Source Tier	Mistakes		Omissions		Contradictions		Ambiguities		Overlaps		Redundancies		Total
	Obj	Sub	Obj	Sub	Obj	Sub	Obj	Sub	Obj	Sub	Obj	Sub	
Established Standards	8	2	2	0	25	6	6	1	8	0	3	0	61
Terminology Collections	17	0	3	0	44	14	16	1	6	0	2	0	103
Textbooks	8	2	2	0	34	6	2	0	1	0	0	0	55
Papers and Others	16	1	6	0	31	21	9	0	0	1	3	0	88
Total	49	5	13	0	134	47	33	2	15	1	8	0	307

TABLE V: Breakdown of identified flaws by domain and source tier.

Source Tier	Categories		Synonyms		Parents		Definitions		Labels		Scope		Traceability		Total
	Obj	Sub	Obj	Sub	Obj	Sub	Obj	Sub	Obj	Sub	Obj	Sub	Obj	Sub	
Established Standards	12	2	4	4	11	1	16	2	9	0	0	0	0	0	61
Terminology Collections	14	10	12	2	12	3	24	0	16	0	5	0	5	0	103
Textbooks	0	0	14	1	10	6	17	1	5	0	0	0	1	0	55
Papers and Others	14	13	19	6	12	3	11	0	8	1	0	0	1	0	88
Total	40	25	49	13	45	13	68	3	38	1	5	0	7	0	307

since additional context may rectify them. Since we give each flaw a manifestation and a domain, the totals per source and grand totals in these tables are equal. From these tables, we can draw some conclusions about how the literature is flawed:

1) Contradictions are by far the most common way for a flaw to manifest, which makes sense: if two (groups of) authors do not communicate or work with different resources, there is a higher chance that they will disagree.
2) Approach categorizations are the most subjective and one of the most common flaw domains, likely due to the lack of standardization about what categories to use.

We summarize the flaws that we discover manually in Section IV-A based on their manifestation. This lets us separately summarize the flaws we automatically detect based on their domain in Section IV-B. Moreover, certain "subsets" of testing contain many interconnected flaws, which we present in subsections as a "third view" to keep related information together. These subsets include recovery testing (Section IV-C), scalability testing (Section IV-D), and compatibility testing (Section IV-E). The counts of flaws given in Tables IV and V can be thought of as the sum of the flaws we describe by manifestation, domain, and "subset".

Note that due to time constraints, this collection of flaws is still not comprehensive! While we apply our heuristics for identifying relevant terms (see Section III-B) to the entirety of most investigated sources, especially established standards (see Section II-E1), we are only able to investigate some sources in part. These mainly comprise of sources chosen for a specific area of interest or based on a test approach that was later determined to be out-of-scope. These include the following sources: [42]–[49]. Since our research began, the draft version of the SWEBOK Guide v4.0 [11] has been updated and published [15]; we revisit this version to see which flaws in the draft were resolved (we find two notable cases of this) following our heuristics based on these subsets, but lack the time to investigate this new version in full. Finally, some heuristics only arose as research progressed, particularly those for deriving test approaches (see Section III-B); while reiterating over investigated sources would be ideal, this is infeasible due to time constraints.

From this partial implementation of our methodology, we identify 561 test approaches, 34% of which are undefined. With more time, we would continue to snowball on these undefined terms following the procedure in Section III-E. Likewise, we were unable to reach our stopping criteria outlined in Section III-F. We considered the discovery of property-based testing as an alternate stopping point for our research since we were surprised that it was not mentioned in any sources we investigated. Even so, we had to stop our snowballing approach before we discovered property-based testing in the literature!

A. Flaws by Manifestation

The following sections list observed flaws grouped by how they manifest as presented in Section II-B1. These include mistakes (Section IV-A1), omissions (Section IV-A2), contradictions (Section IV-A3), ambiguities (Section IV-A4), overlaps (Section IV-A5), and redundancies (Section IV-A6).

1) Mistakes: There are many ways that information can be incorrect which we identify in Table VI. We provide an example below for those that are less straightforward.

a) Information is incorrect based on an assertion from another source: [15, p. 5-4] says that quality improvement, along with quality assurance, is an aspect of testing that involves "defining methods, tools, skills, and practices to achieve the specific quality level and objectives"; while

testing that a system possesses certain qualities is in scope, actively improving the system in response is not part of testing.

b) Information is provided with an incorrect scope: "Par sheet testing" from [19] seems to refer to some specific example it does not cite and does not seem more widely applicable, since a "PAR sheet" is "a list of all the symbols on each reel of a slot machine" [50].

c) Incorrect information makes other information incorrect: The incorrect claim that "white-box testing", "grey-box testing", and "black-box testing" are synonyms for "module testing", "integration testing", and "system testing", respectively, casts doubt on the claim that "red-box testing" is a synonym for "acceptance testing" [51, p. 18].

2) Omissions: We find four cases where a definition is omitted, one where a category is omitted, and one where a term (along with its relations) is omitted.

3) Contradictions: There are many cases where multiple sources of information (sometimes within the same document!) disagree. We find this happen with six categories, six synonym relations, seven parent-child relations, 17 definitions, and three labels.

4) Ambiguities: Some information given in the literature is unclear; there is definitely something "wrong", but we cannot deduce the intent of the original author(s). We identify the kinds of ambiguous information given in Table VII.

5) Overlaps: While information given in the literature should be atomic, this is not always the case. We find three definitions that overlap, two terms with multiple definitions, and three terms that share acronyms.

6) Redundancies: We find redundancies in two parent-child relations, two definitions, and two labels.

TABLE VI: Different kinds of mistakes found in the literature.

Description	Count
Information is incorrect based on an assertion from another source	10
Information is provided with an incorrect scope	7
Information is not present where it is claimed to be	6
Information contains a minor mistake	4[a]
Incorrect information makes other information incorrect	2

[a] Comprises three typos and one duplication.

TABLE VII: Different kinds of ambiguities found in the literature.

Description	Count
A term is defined ambiguously	7
A term is used inconsistently	3
The distinction between two terms is unclear	2

B. Flaws by Domain

The following sections present flaws that we detect automatically grouped by what information is flawed as presented in Section II-B2. We also provide more detailed information on specific areas of these domains that may require further investigation. The domains we focus on here are test approach categories (Section IV-B1), synonym relations (Section IV-B2), and parent-child relations (Section IV-B3).

1) Approach Category Flaws: 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. The most blatant example of this is that ISO/IEC and IEEE [16, 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; we track these differences so we can analyze them more systematically.

In particular, we automatically detect test approaches with more than one category that violate our assumption of orthogonality (see Section II-A1). We identify 28 such cases and list the most prominent along with their sources in Table VIII. These flaws mostly involve the category of "test technique", which shows up in 23 of them. This may simply be because authors use the term "test technique" in a more general sense where we would use the term "test approach". A more specific reason for this is how the line between "test technique" and "test practice" can be blurred when these categories are not transitive. For example, "some test practices, such as exploratory testing or model-based testing are sometimes [incorrectly] referred to as 'test techniques' … as they are not themselves providing a way to create test cases, but instead use test design techniques to achieve that" [18, p. 11], [22, p. 5]. This seems to be the case for the following approaches, which may explain these apparent "flaws":

- Exploratory testing [18, p. 33], [21, p. viii], [52, p. 13]
- Experience-based testing [18, p. 4], [21, pp. viii, 4], [52, p. 33]
- Scripted testing [18, p. 33]

2) Synonym Relation Flaws: While synonyms do not inherently signify a flaw (as we discuss in Section II-A2), the software testing literature is full of incorrect and ambiguous synonyms that do. As described in Section II-B2, we pay special attention to synonyms between indepen-

TABLE VIII: Test approaches with more than one category.

Approach	Category 1	Category 2
Ad Hoc Testing	Practice [52, p. 33]	Technique [11, p. 5-14]
Capacity Testing	Technique [21, pp. 38–39]	Type [18, p. 22], [52, p. 2]
Checklist-based Testing	Practice [18, p. 34]	Technique [19]
Data-driven Testing	Practice [18, p. 22]	Technique [33, p. 43]
End-to-end Testing	Type [19]	Technique [7, p. 47], [53, pp. 601, 603, 605–606]
Endurance Testing	Technique [21, pp. 38–39]	Type [52, p. 2]
Error Guessing	Practice [52, p. 33]	Technique [18, pp. 4, 34, Fig. 2], [21, Fig. 2, Tab. A.2][a]
Experience-based Testing	Technique [7, pp. 46, 50], [18, Fig. 2]	Practice [18, Fig. 2], [21, p. viii], [52, pp. iii, 31, 33]
Exploratory Testing	Technique [7, p. 50], [11, p. 5-14]	Practice [18, pp. 11, 20, 34, Fig. 2], [21, p. viii], [22, p. 5][b]
Load Testing	Technique [21, pp. 38–39]	Type [16, p. 253], [18, pp. 5, 20, 22], [19]
Performance Testing	Technique [21, pp. 38–39]	Type [18, pp. 7, 22, 26–27], [21, p. 7], [22, pp. 2, 8]
Stress Testing	Technique [21, pp. 38–39]	Type [16, p. 442], [18, pp. 9, 22]

[a]Some sources omitted for brevity.
[b]Some sources omitted for brevity.

TABLE IX: 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]	[33, p. 15]
Co-existence Testing	→	Compatibility Testing	[18, p. 3], [21, Tab. A.1], [54]	[21, p. 37]
Fault Tolerance Testing	→	Robustness Testing[a]	[7, p. 56]	[19]
Functional Testing	→	Specification-based Testing[b]	[21, p. 38]	[16, p. 196], [10, p. 399], [33, p. 44]
Orthogonal Array Testing	→	Pairwise Testing	[55, p. 1055]	[11, p. 5-11], [56, p. 473]
Path Testing	→	Exhaustive Testing	[1, pp. 466–467, 476]	[10, p. 421]
Performance Testing	→	Performance-related Testing	[18, p. 22], [21, p. 38]	[57, p. 1187]
Static Analysis	→	Static Testing	[18, pp. 9, 17, 25, 28], [19], [22, p. 3]	[1, p. 438]
Structural Testing	→	Structure-based Testing	[17, pp. 105–121]	[18, p. 9], [19], [16, pp. 443–444]
Use Case Testing	→	Scenario Testing[c]	[21, p. 20]	[19], [33, pp. 47–49]

[a]Fault tolerance testing may also be a subapproach of reliability testing [16, p. 375], [15, p. 7-10], which is distinct from robustness testing [7, p. 53].
[b]Reference [19] cites [16, p. 431] for its definition of "functional testing" but excludes the transitive synonym relationship (see Section II-A2) it gives to "specification-based testing". These terms are also defined separately elsewhere [18], [21], further supporting that they are not synonyms.
[c]Reference [18, Fig. 2] also lists "use case testing" and "scenario testing" separately, further supporting that these terms are not synonyms.

dently defined approaches (which may be flaws) and to intransitive synonyms (which definitely are flaws).

3) Parent-Child Relation Flaws: Parent-child relations are also not immune to flaws. For example, some approaches are given as parents of themselves which violates the irreflexivity of this relation as defined in Section II-A3. For example, performance and usability testing are both given as subapproaches of themselves [25, Tab. 2], [34, Tab. 1].

There are also pairs of synonyms where one is a subapproach of the other; these relations cannot coexist as synonym relations are symmetric while parent-child relations are asymmetric as outlined in Sections II-A2 and II-A3, respectively. We identify 18 of these pairs through automatic analysis of our generated visualizations and list the most prominent in Table IX. Of particular note is the relation between path testing and exhaustive testing. While [10, p. 421] claims that path testing done completely "is equivalent to exhaustively testing the program", this overlooks the effects of input data [1, p. 467], [17, p. 121], [21, p. 129–130] and implementation issues [p. 476] on the code's behaviour. Exhaustive testing requires "all combinations of input values and preconditions … [to be] tested" [18, p. 4] (similar in [19], [17, p. 121]).

C. Recovery Testing

"Recovery testing" is "testing … aimed at verifying software restart capabilities after a system crash or other disaster" [11, p. 5-9] including "recover[ing] the data directly affected and re-establish[ing] the desired state of the system" [54] (similar in [11, p. 7-10]) so that the system "can perform required functions" [16, p. 370]. However, the literature also describes similar test approaches with vague or non-existent distinctions between them. We describe these approaches and their flaws here and present the relations between them in Figure 4a.

- Recoverability testing evaluates "how well a system or software can recover data during an interruption or failure" [11, p. 7-10] (similar in [54]) and "re-establish

the desired state of the system" [54]. Reference [33, p. 47] gives this as a synonym for "recovery testing".

- Disaster/recovery testing evaluates if a system can "return to normal operation after a hardware or software failure" [16, 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].
  - (OVER, DEFS) 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, making this term nonatomic.
- 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" [52, p. 2]. This may be what is meant by "recovery testing" in the context of performance-related testing [18, Fig. 2].
- Backup/recovery testing determines the ability of a system "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].
  - (CONTRA, PARS) This given as a subtype of "disaster/recovery testing" which tests if "operation of the test item can be transferred to a different operating site" [21, p. 37], even though this is explicitly excluded from its definition on the same page!
  - (OVER, LABELS) Its name is also quite similar to "backup and recovery testing", adding further confusion.
- Failover/recovery testing 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].
  - (CONTRA, PARS) This is also given as a subtype of "disaster/recovery testing" which tests if "operation of the test item can be transferred to a different operating site" [p. 37], even though this is explicitly excluded from its definition on the same page!
  - (AMBI, PARS) While not explicitly related to recovery, failover testing "validates the SUT's ability to manage heavy loads or unexpected failure to continue typical operations" [11, p. 5-9] by entering a "backup operational mode in which [these responsibilities] … are assumed by a secondary system" [19]. Its name implies that it is a child of "failover/recovery testing" but its definition makes it more broad (as it includes handling "heavy loads" where failover/recovery testing does not) which may reverse the direction of this relation.
  - (AMBI, SYNS) Reference [7, p. 56] uses the term "failover and recovery testing" which may be a synonym of "failover/recovery testing".
- Restart & recovery (testing) is listed as a test approach by [25, Fig. 5] but is not defined (MISS, DEFS) and may simply be a synonym to "recovery testing" (AMBI, SYNS).

D. Scalability Testing

a) (CONTRA, SYNS): ISO/IEC and IEEE [21, p. 39] give "scalability testing" as a synonym of "capacity testing" while other sources differentiate between the two [7, p. 53], [35, pp. 22–23].

b) (CONTRA, DEFS): ISO/IEC and IEEE [21, p. 39] also include the external modification of the system as part of "scalability" but [54] 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.

E. Compatibility Testing

a) (OVER, DEFS): "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)" [18, p. 3]. This definition is nonatomic as it combines the ideas of "co-existence" and "interoperability".

b) (WRONG, SYNS): The "interoperability" element of "compatibility testing" is explicitly excluded by [21, p. 37], (incorrectly) implying that "compatibility testing" and "co-existence testing" are synonyms.

c) (AMBI, SYNS): Furthermore, the definition of "compatibility testing" in [33, p. 43] unhelpfully says "see interoperability testing", adding another layer of confusion to the direction of their relationship.

d) (WRONG, LABELS): Reference [18, pp. 22, 43] says "interoperability testing helps confirm that applications can work on multiple operating systems and devices", but this seems to instead describe "portability testing", which evaluates the "capability of a product to be adapted to changes in its requirements, contexts of use, or system environment" [54] (similar in [16, pp. 184, 329], [18, p. 7], [19]), such as being "transferred from one hardware … environment to another" [21, p. 39].

V. Recommendations

As we have shown in Section IV, "testing is a mess" [Mosser, 2023, priv. comm.]! It will take a lot of time, effort, expertise, and training to organize these terms (and their relations) logically. However, the hardest step is often the first one, so we attempt to give some examples of how this "rationalization" can occur. These changes often arise when we notice an issue with the current state of the terminology and think about what we would do to make it better. We do not claim that these are correct, unbiased, or exclusive, just that they can be used as an inspiration for those wanting to pick up where we leave off.

When redefining terms, we seek to make 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)

Likewise, we seek to eliminate classes of flaws that can be detected automatically, such as test approaches that are given as synonyms to multiple distinct approaches (??) or as parents of themselves (Section IV-B3), or pairs of approaches with both a parent-child and synonym relation (Section IV-B3).

We give recommendations for the areas of recovery testing (Section V-A), scalability testing (Section V-B), performance-related testing (Section V-C), and compatibility testing (Section V-D). We provide graphical representations (see Section III-G) of these subsets when helpful in Figures 4 and 5, in which arrows representing relations between approaches are coloured based on the source tier (see Section II-E) that defines them. We colour all proposed approaches and relations orange.

A. Recovery Testing

To remedy the flaws we describe in Section IV-C, we recommend that the literature uses these terms more consistently, resulting in the improved graph in Figure 4b. The following proposals "recapture" information from the literature more consistently:

1) Prefer the term "recoverability testing" over "recovery testing" to indicate its focus on a system's ability to recover, not its performance of recovering [33, p. 47]. "Recovery testing" may be an acceptable synonym, since it seems to be more prevelant in the literature.
2) Introduce the term recovery performance testing when evaluating performance metrics of a system's recovery as a subapproach of recoverability testing and performance-related testing[13] ([18, Fig. 2]; [52, p. 2]).
3) Introduce separate terms for the different methods of recovery which are all subapproaches of recoverability testing:
   a) from backup memory (backup recovery testing) ([21, p. 37]; [52, p. 2]),
   b) from a back-up system (failover testing) ([11, p. 5-9]; [19]), or
   c) by transferring operations elsewhere (transfer recovery testing) [21, p. 37].

B. Scalability Testing

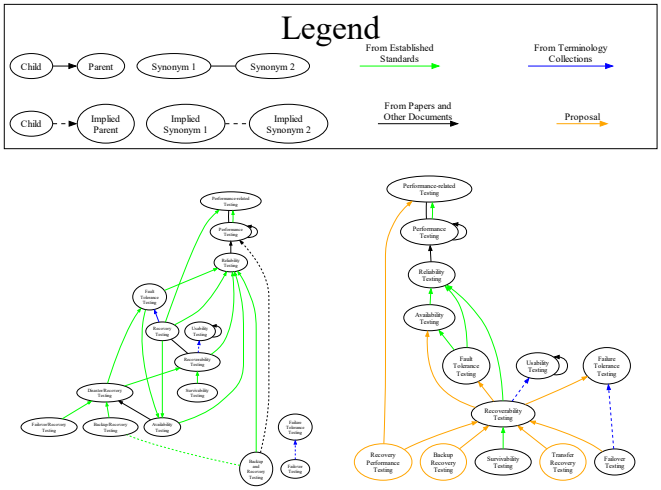
The issues with scalability testing terminology we describe in Section IV-D are resolved and/or explained by other sources! Taking this extra information into account provides a more accurate description of scalability testing.

a) (CONTRA, SYNS): ISO/IEC and IEEE [21, p. 39] define "scalability testing" as the testing of a system's ability to "perform under conditions that may need to be supported in the future". This focus on "the future" is supported by [19], 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" [54]. Therefore, these terms should not be synonyms, as done by [7, p. 53] and [35, pp. 22–23].

b) (CONTRA, DEFS): The underlying reason that sources disagree on whether external modification of the system is part of scalability testing is its confusion with elasticity testing. [Unknown Author: BertolinoEtAl2019] say the two approaches are "closely related" [58, p. 93:28], even claiming that one objective of elasticity testing is "to evaluate scalability" [p. 93:14]! However, [15] (which cites [58]) distinguishes between these approaches:

- Scalability Testing: testing that evaluates "the software's ability to scale up non-functional requirements such as load, number of transactions, and volume of data" [15, p. 5-9; similar on p. 5-5].
- Elasticity Testing: testing that evaluates the ability of a system to "dynamically scal[e] up and down … resources as needed" [58, p. 93:18; similar on p. 93:13] "without compromising the capacity to meet peak utilization" [15, p. 5-9].

This distinction is consistent with how the terms are used in industry: [37] says that scalability is the ability to



(a) Visualization of current relations.

(b) Visualization of proposed relations.

Fig. 4: Visualizations of relations between terms related to recovery testing.

[13] See Section V-C.

"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". Therefore, external modification of a system is part of scalability testing but not elasticity testing. This also implies that [54]'s notion of "scalability" actually refers to "elasticity".

C. Performance(-related) Testing

"Performance testing" is defined as testing "conducted to evaluate the degree to which a test item … accomplishes its designated functions" [16, p. 320], [18, p. 7], [22, p. 2] (similar in [21, pp. 38-39], [57, p. 1187]). It does this by "measuring the performance metrics" [57, p. 1187] (similar in [19]) (such as the "system's capacity for growth" [34, p. 23]), "detecting the functional problems appearing under certain execution conditions" [57, p. 1187], and "detecting violations of non-functional requirements under expected and stress conditions" [57, p. 1187] (similar in [11, p. 5-9]). It is performed either …

1) "within given constraints of time and other resources" [16, p. 320], [18, p. 7] (similar in [57, 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 [18, p. 22]. Note that "performance, load and stress testing might considerably overlap in many areas" [57, p. 1187]. In contrast, [11, p. 5-9] gives "capacity and response time" as examples of "performance characteristics" that performance testing would seek to "assess", which seems to imply that these are subapproaches to performance testing instead. This is consistent with how some sources treat "performance testing" and "performance-related testing" as synonyms [11, p. 5-9], [57, p. 1187], as noted in Section IV-B2. 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" [16, p. 320], [18, p. 7], [22, p. 2] (similar in [57, p. 1187]), and "performance testing" is a subapproach of this performed "under a 'typical' load" [21, p. 39]. This has other implications for relations between these types of testing; for example, "load testing" usually occurs "between anticipated conditions of low, typical, and peak usage" [16, p. 253], [19], [18, p. 5], [21, p. 39], so it is a child of "performance-related testing" and a parent of "performance testing".

After these changes, some finishing touches remain. The reflexive parent relations are incorrect (as described in Section IV-B3) and can be removed. Similarly, since "soak testing" is given as a synonym to both "endurance testing" and "reliability testing" (see ??), it makes sense to just use these terms instead of one that is potentially ambiguous. These changes (along with those from Sections V-A and V-B) result in the proposed relations shown in Figure 5.

D. Compatibility Testing

"Co-existence" and "interoperability" are often defined separately [16, pp. 73, 237], [19], sometimes explicitly as a decomposition of "compatibility" [54]! Following this precedent, "co-existence testing" and "interoperability testing" should be defined as their own test approaches to make their definitions atomic; [16] defines "interoperability testing" [p. 238] but not "co-existence testing". The term "compatibility testing" may still be a useful test approach to define, but it should be defined independently of its children: "co-existence testing" and "interoperability testing".

VI. Conclusion

While a good starting point, the current literature on software testing has much room to grow. The many flaws 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 Approach Categories are sufficient, and rationalize the definitions of and relations between terms.

Acknowledgment

ChatGPT was used to help generate supplementary Python code for constructing graphs and generating LaTeX code, including regex. ChatGPT and GitHub Copilot were both used for assistance with LaTeX formatting. ChatGPT and ProWritingAid were both used for proofreading. Jason Balaci's McMaster thesis template provided many helper LaTeX functions. A special "thank you" to Christopher William Schankula for help with LaTeX and various friends for discussing software testing with me and providing many of the approaches in Section III-F (ChatGPT also provided pointers to the potential existence of some of these approaches). Finally, Dr. Spencer Smith and Dr. Jacques Carette have been great supervisors and valuable sources of guidance and feedback.

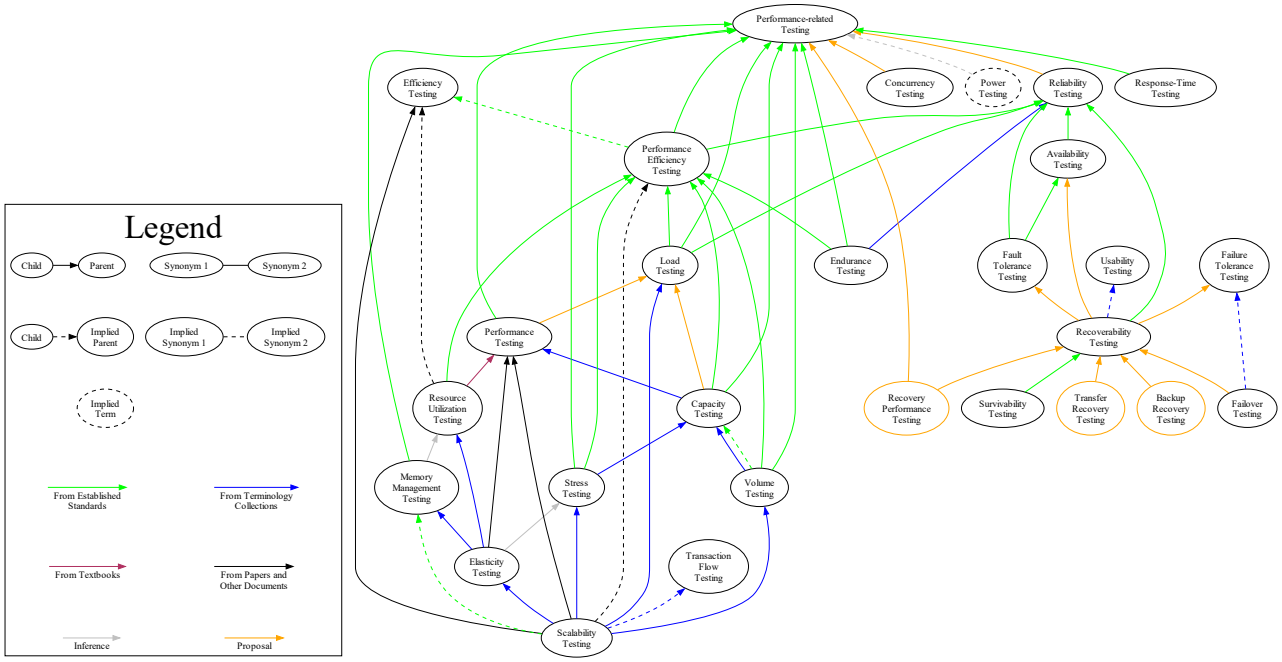


Fig. 5: Visualization of proposed relations between terms related to performance-related testing.

References

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

[2] P. Naur and B. Randell, "Software Engineering: Report on a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7th to 11th October 1968." Brussels, Belgium: Scientific Affairs Division, NATO, Jan. 1969. [Online]. Available: http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF

[3] R. M. McClure, "Introduction," Jul. 2001. [Online]. Available: http://homepages.cs.ncl.ac.uk/brian.randell/NATO/Introduction.html

[4] 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

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

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

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

[8] 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

[9] ISO/IEC and IEEE, "ISO/IEC/IEEE International Standard - Systems and software engineering–Vocabulary," ISO/IEC/IEEE 24765:2010(E), Dec. 2010.

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

[11] H. Washizaki, Ed., Guide to the Software Engineering Body of Knowledge, Version 4.0, Jan. 2024.

[12] ISO/IEC and IEEE, "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.

[13] ISO/IEC, "ISO/IEC 25000:2005 - Software Engineering –Software product Quality Requirements and Evaluation (SQuaRE) –Guide to SQuaRE," ISO/IEC 25000:2005, Aug. 2005. [Online]. Available: https://www.iso.org/obp/ui/#iso:std:iso-iec:25000:ed-1:v1:en

[14] 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

[15] H. Washizaki, Ed., Guide to the Software Engineering Body of Knowledge, Version 4.0a, May 2025a. [Online]. Available: https://ieeecs-media.computer.org/media/education/swebok/swebok-v4.pdf

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

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

[18] 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.

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

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

[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. 2021c.

[22] ——, "ISO/IEC/IEEE International Standard - Software and systems engineering –Software testing –Part 2: Test processes," ISO/IEC/IEEE 29119-2:2021(E), Oct. 2021a.

[23] ISO/IEC, "ISO/IEC 25051:2014 - Software engineering –Systems and software Quality Requirements and Evaluation (SQuaRE) –Requirements for quality of Ready to Use

Software Product (RUSP) and instructions for testing," ISO/IEC 25051:2014, Feb. 2014. [Online]. Available: https://www.iso.org/obp/ui/#iso:std:iso-iec:25051:ed-2:v1:en
[24] W. E. Perry, Effective Methods for Software Testing, 3rd ed. Indianapolis, IN, USA: Wiley Publishing, Inc., 2006.
[25] 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
[26] 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.
[27] ISO/IEC and IEEE, "ISO/IEC/IEEE International Standard - Software and systems engineering –Software testing –Part 3: Test documentation," ISO/IEC/IEEE 29119-3:2021(E), Oct. 2021b.
[28] K. Sakamoto, K. Tomohiro, D. Hamura, H. Washizaki, and Y. Fukazawa, "POGen: A Test Code Generator Based on Template Variable Coverage in Gray-Box Integration Testing for Web Applications," in Fundamental Approaches to Software Engineering, V. Cortellessa and D. Varró, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, Mar. 2013, pp. 343–358. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-642-37057-1_25
[29] IEEE, "IEEE Standard for System and Software Verification and Validation," IEEE Std 1012-2012 (Revision of IEEE Std 1012-2004), 2012.
[30] H. Washizaki, "Software Engineering Body of Knowledge (SWEBOK)," Sep. 2025b. [Online]. Available: https://www.computer.org/education/bodies-of-knowledge/software-engineering/
[31] 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
[32] 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
[33] 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
[34] 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
[35] 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
[36] LambdaTest, "What is Operational Testing: Quick Guide With Examples," 2024. [Online]. Available: https://www.lambdatest.com/learning-hub/operational-testing
[37] P. Pandey, "Scalability vs Elasticity," Feb. 2023. [Online]. Available: https://www.linkedin.com/pulse/scalability-vs-elasticity-pranav-pandey/
[38] 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
[39] ChatGPT (GPT-4o), "Defect Clustering Testing," Nov. 2024. [Online]. Available: https://chatgpt.com/share/67463dd1-d0a8-8012-937b-4a3db0824dcf
[40] 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
[41] 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
[42] 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
[43] ——, "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
[44] 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
[45] 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.
[46] 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.
[47] 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
[48] K.-L. Tsui, "An Overview of Taguchi Method and Newly Developed Statistical Methods for Robust Design," IIE Transactions, vol. 24, no. 5, pp. 44–57, May 2007, publisher: Taylor & Francis. [Online]. Available: https://doi.org/10.1080/07408179208964244
[49] W. Goralski, "xDSL loop qualification and testing," IEEE Communications Magazine, vol. 37, no. 5, pp. 79–83, 1999.
[50] M. Bluejay, "Slot Machine PAR Sheets," May 2024. [Online]. Available: https://easy.vegas/games/slots/par-sheets
[51] 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
[52] ISO/IEC and IEEE, "ISO/IEC/IEEE International Standard - Systems and software engineering –Software testing –Part 1: General concepts," ISO/IEC/IEEE 29119-1:2013, Sep. 2013.
[53] 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
[54] 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
[55] 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
[56] 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
[57] 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

[58] A. Bertolino, G. D. Angelis, M. Gallego, B. García, F. Gortázar, F. Lonetti, and E. Marchetti, "A Systematic Review on Cloud Testing," ACM Computing Surveys, vol. 52, no. 5, Sep. 2019, place: New York, NY, USA Publisher: Association for Computing Machinery. [Online]. Available: https://doi.org/10.1145/3331447