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 563 test approaches and 75 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 276 flaws, including nine terms used as synonyms to two (or more) disjoint test approaches and 16 pairs of test approaches that may either be synonyms or have a parent-child relationship. This also highlights notable confusion surrounding functional, operational acceptance, recovery, 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 III-A1) and terminology collections (see Section III-A2) 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" [9, p. 139], [12, p. 7] 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 [13, p. 4-3]. The term "fault" is "overloaded with too many meanings, as engineers and others use the word to refer to all different types of anomalies" [11, p. 12-3], and "defect" may be used as a "generic term that can refer to either a fault (cause) or a failure (effect)" [9, p. 96], [14, p. 124]. Software testers may even choose to ignore these nuances completely! Reference [15, 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 on the same page—twice [16, Fig. 2, p. 34]! The structure of tours can be defined as either quite general [16, p. 34] or "organized around a special focus" [17]. Load testing is performed with loads "between anticipated conditions of low, typical, and peak usage" [16, p. 5] or loads that are as large as possible [15, p. 86]. Alpha testing is performed by "users within the organization developing the software" [14, p. 17], "a small, selected group of potential users" [11, p. 5-8], or "roles outside the development organization" conducted "in the developer's test environment" [17]. It is clear that there is a notable gap in the literature, one which we attempt to describe and fill. While the creation of a complete taxonomy is unreasonable, especially considering the pace at which the field of software changes, we can make progress towards this goal that others can extend and update as new test approaches emerge. 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-A), 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" [18, 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 563 test approaches mentioned by 66 sources (described in Section III-A), recording their names, categories[1] (see Section II-D), definitions, synonyms (see Section II-E), parents (see Section II-F), and flaws (see Section II-A) (in a separate document) as applicable. We also record any other relevant notes, such as prerequisites, uncertainties, and other resources. We follow the procedure laid out in

[1] A single test approach with more than one category indicates an underlying flaw; see Section IV-B1.

Section III-B 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?

We then create tools to support our analysis of our findings (Section III-C). 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 do our best to hold ourselves to a higher standard by defining and using terms consistently. For example, before we can constructively describe the flaws in the literature, we need to define what we mean by "flaw" and its related terms (Section II-A). We also track if information found in our sources is more implicit (and therefore more subjective) by defining the notion of "rigidity" (Section II-B) and similarly track the "trustworthiness" of the sources themselves (Section II-C). We make these distinctions to reduce how much our preconceptions affect our analysis (or at least make it more obvious to future researchers). To further prevent bias, we do not invent or add our own classifications or kinds of relations. Instead, the notions of test approach categories (Section II-D), synonyms (Section II-E), and parent-child relations (Section II-F) we present here follow logically from the literature. We define them here for clarity since we use them throughout this paper, even though they are "results" of our research.

A. Flaws/Inconsistencies

Before we can start tracking and discussing flaws, we need to be clear about what we mean by the term. Our research shows that the literature is full of incorrect, missing, contradictory, unclear, nonatomic, and redundant information. This should not be the case for a field as rigorous as software engineering; we should be correct, complete, consistent, and clear, keeping separate ideas separate and only including what is necessary! We refer to any instance where one of these ideals is violated as a "flaw", as it implies that something is wrong with the literature and an opportunity to improve it, while avoiding words that are "overloaded with too many meanings" like "error" and "fault" [11, p. 12-3; see Section I]. Terms such as these are primarily used to describe software itself, while we want a term to describe its supporting documents[2]. Reference [11, p. 7-9] mentions that "techniques and indicators can help engineers measure technical debt, including … the number of engineering flaws and violations", which aligns with our goal of analyzing what is wrong with software engineering's testing literature.

[2]A small literature review reveals that established standards (see Section III-A1) only use "flaw" to refer to requirements [16, p. 38], design [p. 43], "system security procedures … and internal controls" [19, p. 194], or code itself [p. 92].

TABLE I: 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[c]	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 would be unintuitive and OMIT would be inconsistent with the keys being adjective-based.
[c] Section omitted for brevity.

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 I 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. This 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 the flaw can be!

While we use the general term "flaw" to encompass all of these manifestations, it has its limitations. In particular, when referring to a flaw with more than one source (such as a contradiction or an overlap), it is awkward to call it a "flaw between two sources". A more intuitive way to describe this situation is that there is 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! Note that these cases are not categorized as "mistakes" if finding this 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 II, and

TABLE II: Observed flaw domains.

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

tracking these uncovers which knowledge domains are less standardized (and should therefore be approached with more rigour) than others. Note that this is an orthogonal categorization to that of a flaw's manifestation: each flaw manifests in a particular domain. This means that we give each flaw two keys (one for each classification) and present our observations according to these two views in Tables IV and V. We explicitly define some of these domains in future chapters and thus present them in that same order. Despite their nuance, the remaining domains are relatively straightforward, so we do not define them more rigorously in their own sections.

For example, 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; while one could argue that their "internal" definition of the term is the cause of this mistake, we consider this a label flaw where the wrong label is used and we would change the label to fix it. Additionally, some traceability information is flawed, such as how one document cites other or even what information is included within a document (see ??)!

B. Rigidity

When recording information from unstandardized sources written in natural language, there is a considerable degree of nuance that can get lost. 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. We call this nuance "rigidity" and capture it when citing sources using (at least) one of the following keywords: "implied", "inferred", "can be", "should be", "ideally", "usually", "most", "likely", "often", "if", and "although". While these keywords often appear directly within the literature, we also use them to track rigidity without getting distracted by less relevant details by summarizing the relevant nuance. This allows us to provide a more complete picture of the state of the literature. All sources cited throughout this paper support their respective claims explicitly unless specified otherwise, usually via one of the keywords given above. The following non-mutually exclusive reasons for information to be considered "implicit" emerged, and we use the keywords from the above list during data collection to identify them when performing later analysis (see the relevant source code):

1) The information can be logically deduced but is not explicit. The implicit categorizations of "test type" by [7, pp. 53–58] (cf. Table VI) are examples of this. The given test approaches are not explicitly called "test types", as the term is used more loosely to refer to different kinds of testing—what should be called "test approaches" as per Table III. However, this set of test approaches are "based on the associated quality characteristic and its associated quality attributes" [p. 53], implying that they are test types. Cases such as this are indicated by a question mark or one of the following keywords: "implied", "inferred", or "likely".
2) The information is not universal. Reference [14, p. 372] defines "regression testing" as "testing required to determine that a change to a system component has not adversely affected functionality, reliability or performance and has not introduced additional defects". While reliability testing, for example, is not always a subset of regression testing (since it may be performed in other ways), it can be accomplished by regression testing, so there is sometimes a parent-child relation (defined in Section II-F) between them. Cases such as this are indicated by one of the following keywords: "can be", "should be", "ideally", "usually", "most", "likely", "often", or "if".
3) The information is conditional. As a more specific case of information not being universal, sometimes prerequisities must be satisfied for information to apply. For example, branch condition combination testing is equivalent to (and is therefore a synonym of) exhaustive testing if "each subcondition is viewed as a single input" [1, p. 464]. Likewise, statement testing can be used for (and is therefore a child of) unit testing if there are "less than 5000 lines of code" [p. 481]. Cases such as this are indicated by the keyword "can be" or "if".
4) The information is dubious. This happens when there is reason to doubt the information provided. If a source claims one thing that is not true, related claims lose credibility. For example, 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" [20, p. 18]. Doubts such as this can also originate from other sources. Reference [21, p. 48] gives "user

scenario testing" as a synonym of "use case testing", even though "an actor [in use case testing] can be … another system" [22, p. 20], which does not fit as well with the label "user scenario testing". However, since a system can be seen as a "user" of the test item, this synonym relation is treated as implicit instead of as an outright flaw. Cases such as this are indicated by a question mark or one of the following keywords: "inferred", "should be", "ideally", "likely", "if", or "although".

Any kind of information can be implicit, including the names, definitions, categories (see Section II-D), synonyms (see Section II-E), and parents (see Section II-F) of identified test approaches. Flaws based on implicit information are themselves implicit. By looking for the indicators of uncertainty mentioned above (see the relevant source code), we can automatically detect implicit flaws when generating graphs and performing analysis (see Section III-C). For example, we can view implicit flaws separately in Tables IV and V, since additional context may rectify them.

C. Trustworthiness

In the same way we distinguish between the rigidity of information from different sources, we also wish to distinguish between the "rigidity" of the sources themselves! Of course, we do not want to overload terms, so we define a source as more "trustworthy" 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 (cf. Section III-A).

D. Approach Categories

While there are many ways to categorize software testing approaches, perhaps the most widely used is the one given by ISO/IEC and IEEE [16]. This schema divides test approaches—a generic, catch-all term for any form of "testing"—into levels, types, techniques, and practices [16, Fig. 2; see Table III]. 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], [17], [23, pp. 807–808], [24, pp. 9, 13], [25, p. 218], although they may use a different term for "test level" (see Table III). Because of their widespread use and their usefulness in dividing the domain of software testing into more manageable subsets, we use these categories for now. These four subcategories of test approaches can be loosely described by 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

For example, boundary value analysis is a test technique since its inputs are "the boundaries of equivalence partitions" [16, p. 2], [22, p. 1]. Similarly, acceptance testing is a test level since its goal is to "enable a user, customer, or other authorized entity to determine whether to accept a system or component" [14, p. 5], which requires the system or component to be developed and ready for testing.

Based on their definitions and usage, the categories given in Table III seem to be orthogonal (excluding the "approach" supercategory). For example, "a test type can be performed at a single test level or across several test levels" [16, p. 15], [22, p. 7], and "Keyword-Driven Testing can be applied at all testing levels … and for various types of testing" [26, p. 4]. Therefore, we assume these categories to be orthogonal throughout this paper (e.g., when identifying flaws). We may assess this assumption more rigorously in the future, but for now, it implies that a specific test approach can be derived by combining multiple test approaches from different categories. For example, usability test script(ing) [17] is a combination of usability testing, a test type [16, pp. 22, 26–27], [22, pp. 7, 40, Tab. A.1], and scripted testing, a test practice [16, pp. 20, 22].

In addition to the categories of approach, level, type, technique, and practice, [16, Fig. 2] also includes "static testing", which seems to be non-orthogonal to the others and thus less helpful for grouping test approaches. For example, static assertion checking (mentioned by [27, p. 343], [28, p. 345]) is a subapproach of assertion checking, which can also be performed dynamically. This parent-child relation (defined in Section II-F) means that static assertion checking may inherit assertion checking's inferred category of "practice". Based on observations such as this, we categorize testing approaches, including static ones, based on the remaining categories from [16]. While we can categorize the vast majority of identified test approaches based on [16]'s categories, there are some outliers. For example, we categorize some test approaches as "artifacts", since some terms can refer to the application of a test approach as well as the resulting document(s). Therefore, we do not consider these cases to violate our assumption that [16]'s categories are orthogonal and do not include them as flaws in Section IV-B1.

A side effect of using the terms "level", "type", "technique", and "practice" is that they—perhaps excluding "level"—can be used interchangably or as synonyms for "approach". For example, [15, p. 88] says that if a specific defect is found, it is wise to look for other defects in the same location and for similar defects in other locations, but does not provide a name for this approach. After

researching in vain, we ask ChatGPT[3] to name the "type of software testing that focuses on looking for bugs where others have already been found" [29], using the word "type" in a general sense, akin to "kind" or "subset". Interestingly, ChatGPT "corrects" our imprecise term in its response, using the more correct term "approach" (although it may have been biased by our previous usage of these terms)! Because natural language can be ambiguous, we need to exercise judgement when determining if these terms are being used in a general or technical sense. For example, [21, p. 45] defines interface testing as "an integration test type that is concerned with testing … interfaces", but since it does not define "test type", this may not have special significance.

E. Synonym Relations

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

1) Black-Box Testing [16, p. 9], [17], [14, p. 431], [11, p. 5-10], [22, p. 8], [10, p. 399], [31, p. 344]
2) Closed-Box Testing [16, p. 9], [14, p. 431]
3) Functional Testing[4] [14, p. 196], [10, p. 399], [21, p. 44] (implied by [14, p. 431], [22, p. 129])
4) Domain Testing [11, p. 5-10]
5) Specification-oriented Testing [1, p. 440, Fig. 12.2]
6) Input Domain-Based Testing (implied by [13, pp. 4-7 to 4-8])

These synonyms are the same as synonyms in natural language; while they may emphasize different aspects or express mild variations, their core meaning is nevertheless the same. 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. Thus, synonyms are not inherently problematic, although they can be (see Section IV-B2).

Synonym relations are often given explicitly in the literature. For example, [16, p. 9] lists "black-box testing" and "closed box testing" beneath the glossary entry for "specification-based testing", meaning they are synonyms. "Black-box testing" is likewise given under "functional testing" in [14, p. 196], meaning it is also a synonym for "specification-based testing" through transitivity. However, these relations can also be less "rigid" (see Section II-B); "functional testing" is listed in a cf. footnote to the glossary entry for "specification-based testing" [14, p. 431], which supports the previous claim but would not necessarily indicate a synonym relation on its own.

Similarly, [11, p. 5-10] says "specification-based techniques … [are] sometimes also called domain testing techniques" in the SWEBOK Guide V4, from which the synonym of "domain testing" follows logically. However, its predecessor V3 only implies the more specific "input domain-based testing" as a synonym. The section on test techniques says "the classification of testing techniques presented here is based on how tests are generated: from the software engineer's intuition and experience, the specifications, the code structure …" [13, p. 4-7], and the first three subsections on the following page are "Based on the Software Engineer's Intuition and Experience", "Input Domain-Based Techniques", and "Code-Based Techniques" [p. 4-8]. The order of the introductory list lines up with these sections, implying that "input domain-based techniques" are "generated[] from … the specifications" (i.e., that input domain-based testing is the same as specification-based testing). Furthermore, the examples of input domain-based techniques given—equivalence partitioning, pairwise testing, boundary-value analysis, and random testing—are all given as children[5] of specification-based testing [16], [17], [22, Fig. 2]; even V4 agrees with this [11, pp. 5-11 to 5-12]!

F. Parent-Child Relations

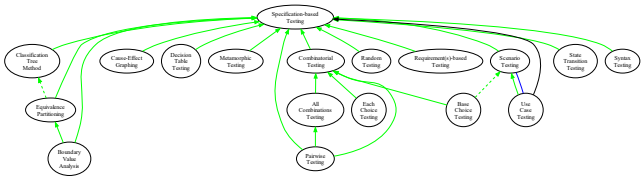
Many test approaches are multi-faceted and can be "specialized" into others; for example, there are many subtypes of performance-related testing, such as load testing and stress testing (see Section V-C). These "specializations" will be referred to as "children" or "subapproaches" of the multi-faceted "parent". This nomenclature also extends to other categories (such as "subtype"; see Section II-D and Table III) and software qualities ("subquality"). There are many reasons two approaches may have a parent-child relation, such as:

1) One is a superset of the other. In other words, for one (parent) test approach to be performed in its entirety, the other (child) approach will necessarily be performed as well. This is often the case when one "well-understood" subset of testing can be decomposed into smaller, independently performable approaches. When all of these have been completed, we can logically conclude that the parent approach has also been performed! In practice, this is much harder to prove; although many hierarchies exist [7], [16, Fig. 2], [22, Fig. 2], these are likely incomplete. As an example, we graph the parent-child relations from [16, Fig. 2], [22, Fig. 2] in the subdomain of specification-based testing in Figure 1a (along with relevant data from other sources).
2) One is "stronger than" or "subsumes" the other. When comparing adequacy criteria that "specif[y]

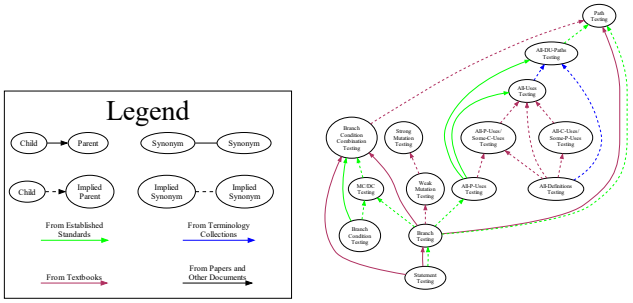
[3]We do not take ChatGPT's output to be true at face value; this approach seems to be called "defect-based testing" based on the principle of "defect clustering" [29], which [30] supports.

[4]"Functional testing" may not actually be a synonym for "specification-based testing"; see Section IV-C1.

[5]Pairwise testing is given as a child of combinatorial testing, which is itself a child of specification-based testing, by [11, pp. 5-11 to 5-12] and [22, Fig. 2], making it a "grandchild" of specification-based testing according to these sources.



(a) "Superset" relations.



(b) "Subsume" relations.

Fig. 1: Graphs of different classes of parent-child relations.

requirements for testing" [10, p. 402], "criterion X is stronger than criterion Y if, for all programs P and all test sets T, X-adequacy implies Y-adequacy" [p. 432]. While this relation only "compares the thoroughness of test techniques, not their ability to detect faults" [p. 434], it is sufficient to consider one a child of the other in a sense. We capture this nuance by considering these parent-child relations implicit (see Section II-B). As an example, we graph the parent-child relations from [Fig. 13.17], [22, Fig. F.1] in the subdomain of data flow testing can be found in Figure 1b (along with relevant data from other sources).

3) The parent approach is part of an orthogonal set. When presented with a set of generic test approaches that are orthogonal to each other, it is often trivial to classify a given test approach as a child of just one them. For example, ISO/IEC and IEEE say that "testing can take two forms: static and dynamic" [16, p. 17] and provide examples of subapproaches of static and dynamic testing [Fig. 1]. Likewise, Gerrard says "tests can be automated or manual" [24, p. 13] and gives subapproaches of automated and manual testing [Tab. 2], [32, Tab. 1]. However, the orthogonality of these subsets does not mean they are mutually exclusive; in these same tables, Gerrard labels usability testing as both static and dynamic and 12 approaches as able to "be done manually or using a tool" [24, p. 13].

III. Methodology

The main goal of our research is to produce a complete, correct taxonomy of software testing terminology. However, we cannot do this without understanding its current state. Therefore, we start by documenting how the literature currently uses testing terminology, both correctly and incorrectly. This results in a big messy glossary of software testing terms, along with a list of flaws. Once we analyze these data, including how and why these flaws emerge, we can start to resolve them. For now, we document the current (messy) state of software testing terminology by:

1) Identifying authoritative sources (Section III-A)
2) Identifying all test approaches from each source and recording their: (Section III-B)
   a) Names
   b) Categories[6] (Section II-D)
   c) Definitions
   d) Synonyms (Section II-E)
   e) Parents (Section II-F)
   f) Flaws (Section II-A) (in a separate document)
   g) Other relevant notes (e.g., prerequisites, uncertainties, and other resources)
3) Alongside step 2, identifying and recording related testing terms that:
   a) Imply related test approaches (Section III-B1)
   b) Are used repeatedly
   c) Have complex definitions
4) Repeating steps 1 to 3 for any missing or unclear terms (Section III-B2) until some stopping criteria

A. 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. Since we are particularly interested in software engineering, we start from the vocabulary document for systems and software engineering [14] and two versions of the Guide to the SoftWare Engineering Body Of Knowledge (SWEBOK Guide)—the newest one [13] and one submitted for public review[7] [11]. 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" [33]. We apply this concept to "referrals" between sources. For example, [17] cites [34] as the original source for its definition of "scalability" (see Section V-B); we verified this by looking at this original source. We similarly "snowball" on terminology itself; when a term requires more investigation (e.g., its definition is missing or unclear), we perform a miniature literature review on this subset to "fill in" this missing information (see Section III-B2). We may then investigate these additional sources in their entirety, as opposed to just the original subset of interest, based on

[6] A single test approach with more than one category indicates an underlying flaw; see Section IV-B1.

[7] Reference [11] has been published since we investigated these sources; if time permits, we will revisit this published version.

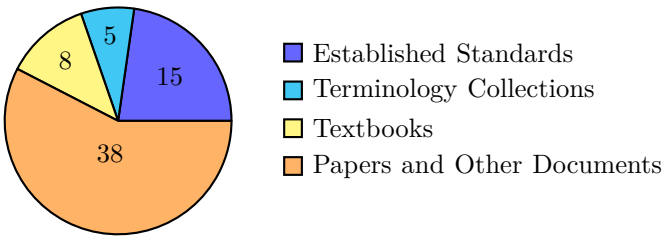


Fig. 2: Summary of how many sources comprise each source tier.

their trustworthiness (defined in Section II-C) and how much extra information they provide.

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 trustworthiness (defined in Section II-C). We therefore create the following tiers, given in order of descending trustworthiness:

1) established standards (Section III-A1),
2) terminology collections (Section III-A2),
3) textbooks (Section III-A3), and
4) papers and other documents (Section III-A4).

A summary of how many sources comprise each tier is given in Figure 2. We often use papers to "fill in" missing information in a more specific subdomain and do not always investigate them entirely (see Section III-B2), which results in a large number of them. We use standards the second most frequently due to their high trustworthiness and broad scope; for example, the glossary portion of [14] 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. 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" [14, p. viii]. For these reasons, they are the most trustworthy sources. However, this does not imply perfection, as we identify 50 flaws within these standards (see Tables IV and V)! Only standards for software development and testing are in scope for this research (see Section I). For example, "the purpose of the ISO/IEC/IEEE 29119 series is to define an internationally agreed set of standards for software testing that can be used by any organization when performing any form of software testing" [16, p. vii].

2) Terminology Collections: These are collections of software testing terminology built up from multiple sources, such as the established standards outlined in Section III-A1. 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" [35]. 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. [36]'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 trustworthy than papers (see Section III-A4) 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], [15] served as the original (albeit ad hoc and arbitrary) starting point of this research, and we investigate other books as they arise. For example, [17] cites [34] as the original source for its definition of "scalability" (see Section V-B); we verified this 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 [21], [24], [32]
- Thesis [37]
- Website [38], [39]
- Booklet [40]
- ChatGPT [29] (with its claims supported by [30])

B. Procedure

We track terminology used in the literature by building glossaries. The one most central to our research is our test approach glossary, where we give each test approach its own row to record its name and any given definitions, categories (see Section II-D), synonyms (see Section II-E), and parents (see Section II-F). If no category is given, the "approach" category is assigned (with no accompanying citation) as a "catch-all" category. All other fields may be left blank, but a lack of definition indicates that the approach should be investigated further to see if its inclusion is meaningful (see Section III-B2). Any additional information from other sources is added to or merged with the existing information in our glossary where appropriate. This includes the generic "approach"

TABLE III: Categories of testing given by ISO/IEC and IEEE.

Term	Definition	Examples
Test Approach	A "high-level test implementation choice" that includes "test level, test type, test technique, test practice and … static testing" [16, p. 10] and is used to "pick the particular test case values" [14, p. 465]	black or white box, minimum and maximum boundary value testing [14, p. 465]
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 [16, p. 12], [22, p. 6] with their "own documentation and resources" [14, p. 469]	unit/component testing, integration testing, system testing, acceptance testing [14, p. 467], [16, p. 12], [22, p. 6]
Test Practice	A "conceptual framework that can be applied to … [a] test process to facilitate testing" [14, p. 471], [16, p. 14]	scripted testing, exploratory testing, automated testing [16, p. 20]
Test Technique[b]	A "procedure used to create or select a test model, identify test coverage items, and derive corresponding test cases" [16, p. 11] (similar in [14, p. 467]) that "generate evidence that test item requirements have been met or that defects are present in a test item" [22, p. vii]	equivalence partitioning, boundary value analysis, branch testing [16, p. 11]
Test Type	"Testing that is focused on specific quality characteristics" [14, p. 473], [16, p. 15], [22, p. 7]	security testing, usability testing, performance testing [14, p. 473], [16, p. 15]

[a] Also called "test phase" or "test stage" (see relevant synonym flaws in Section IV-B2).
[b] Also called "test design technique" [16, p. 11], [17].

category being replaced with a more specific one, an additional synonym being mentioned, or another source describing an already-documented parent-child relation. If any new information contradicts existing information (or otherwise indicates something is wrong), this is investigated and documented (see Section IV), which may be done in a separate document and/or in the glossary itself. Sometimes, new information does not conflict with existing information, in which case the clearest and most concise version is kept, or they are merged to paint a more complete picture. Finally, we record any other notes, such as questions, prerequisites, and other resources to investigate.

We use similar procedures to track software qualities and supplementary terminology (either shared by multiple approaches or too complicated to explain inline) in separate glossaries with a similar format. The name, definition, and synonym(s) of all terms are tracked, as well as any precedence for a related test type for a given software quality. We use heuristics to guide this process for all three glossaries to increase confidence that all terms are identified, paying special attention to the following when investigating a new source:

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

We apply these heuristics to most investigated sources, especially established standards (see Section III-A1), in their entirety. Some sources, however, are only partially investigated, such as those chosen for a specific area of interest or based on a test approach that was determined to be out-of-scope. These include the following sources as described in Section III-B2: [41]–[48].

During the first pass of data collection, we investigate and record all terminology related to software testing. Some of these terms are less applicable to test case automation—our original motivation—or quite broad, so they will be omitted during future analysis.

1) Derived Test Approaches: Throughout this research, we noticed many groups of test approaches that arise from some underlying area of software (testing) knowledge. The legitimacy of extrapolating new test approaches from these knowledge domains is heavily implied by the literature, but not explicitly stated as a general rule. Regardless, since the field of software is ever-evolving, it is crucial to be able to adapt to, talk about, and understand new developments in software testing. Bases for defining new test approaches suggested by the literature include coverage metrics, software qualities, and attacks. These are meaningful enough to merit analysis and are therefore in scope. Requirements may also imply related test approaches, but this mainly results in test approaches that would be out of scope. Other test approaches found in the literature are derived from programming languages or other orthogonal test approaches, but these are out of scope as this information is better captured by other approaches.

2) 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, many undefined test approaches are given by [7] and [16]. Once we exhaust the standards in Section III-A1, we perform miniature literature reviews on these subsets to "fill in" the missing definitions (along with any relations), essentially "snowballing" on these terms. This process uncovers even more approaches, although some are out of scope, such

as EManations SECurity (EMSEC) testing and aspects of Orthogonal Array Testing (OAT) (see Section I). The following terms (and their respective related terms) were explored in the sources given:

- Assertion Checking: [27], [28], [49]
- Loop Testing[8]: [50]–[53]
- EMSEC Testing: [54], [55]
- Asynchronous Testing: [56]
- Performance(-related) Testing: [57]
- Web Application Testing: [21], [36]
  - HTML Testing: [20], [32], [58]
  - Document Object Model (DOM) Testing: [59]
- Sandwich Testing: [60], [61]
- Orthogonal Array Testing[9]: [62], [63]
- Backup Testing[10]: [37]

Applying our procedure from Section III-B to these sources brings the number of testing approaches from 476 to 563 and the number of undefined terms from 175 to 193. This implies that about 79% of added test approaches are defined, which helps verify that our procedure constructively uncovers new terminology.

In addition to terms with missing definitions, some terms do not appear in the literature at all! While most test approaches arise as a result of our snowballing approach, we each have preexisting knowledge of what test approaches exist (a form of experience-based testing, if you will). As an example, we are surprised that property-based testing is not mentioned in any sources investigated, even using it as a target "stopping point" throughout this process. Test approaches such as these that arise independently of snowballing may serve as starting points for continuing research if they are not mentioned by the literature. The following terms come from previous knowledge, conversations with colleagues, research for other projects, or ad hoc cursory research to see what other test approaches exist:

1) Chaos engineering
2) Chosen-ciphertext attacks
3) Concolic testing
4) Concurrent testing
5) Destructive testing
6) Dogfooding
7) Implementation-based testing
8) Interaction-based testing
9) Lunchtime attacks
10) Parallel testing
11) Property-based testing
12) Pseudo-random bit testing
13) Rubber duck testing
14) Shadow testing
15) Situational testing

C. Tools

To better visualize how test approaches relate to each other, we develop a tool to automatically generate graphs of these relations. We graph all parent-child relations, since they are guaranteed to be visually meaningful, but only graph some synonym relations. For a given synonym pair to be captured by our methodology, at least one term will have its own row in its relevant glossary. We then decide whether to include or exclude the synonym pair from our generated graphs based on the following possible cases:

1. (Excluded)
   Only one synonym has its own row. This is a "typical" synonym relation (see Section II-E) where the terms are interchangeable. We could include the synonym as an alternate name inside the node of its partner, but we do not want to clutter our graphs unnecessarily.
2. (Included)
   Both synonyms have their own row in the glossary. This may indicate that the synonym relation is incorrect, as it equates separate approaches (with their own definitions, nuances, etc.) that should likely be distinct. This could, however, also indicate that the two terms are interchangeable and could be merged into one row, which would result in Case 1 above.
3. (Included)
   Two synonym pairs share a synonym without its own row. This is a transitive extension to the previous case. If two distinct approaches share a synonym, that implies that they are synonyms themselves, resulting in the same possibility of the relation being incorrect.

Since these graphs tend to be large, it is useful to focus on specific subsets of them. We can generate more focused graphs from a given subset of approaches, such as those in a selected approach category (see Section II-D) or those pertaining to recovery or scability; the latter are shown in Figures 3a and 4a, respectively. By specifying sets of approaches and relations to add or remove, we can then update these generated graphs based on our recommendations; applying those given in Sections V-A, V-B, and V-C results in the updated graphs in Figures 3b, 4b, and 5, respectively. We color any added approaches or relations orange to distinguish them. Recommendations can also be inherited; for example, we generate Figure 5 based on the modifications we apply to Figures 3b and 4b and other changes from Section V-C.

IV. Flaws

After gathering all these data[11], we find many flaws. ?? shows the source tiers (see Section III-A) responsible for these flaws, which reveals a lot about software testing literature:

1) Established standards (Section III-A1) aren't actually standardized, since:
   a) other documents disagree with them very frequently and
   b) they are the most internally inconsistent source tier!

[8]References [41] and [42] were used as reference for terms but not fully investigated, [44] and [45] were added as potentially in scope, and [43] and [48] were added as out-of-scope examples.

[9]References [46] and [47] were added as out-of-scope examples.

[10]See Section IV-D.

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

TABLE IV: Breakdown of identified Flaws by Manifestation by Source Tier.

Source Tier	Mistakes		Omissions		Contradictions		Ambiguities		Overlaps		Redundancies[a]		Total
	Exp	Imp	Exp	Imp	Exp	Imp	Exp	Imp	Exp	Imp	Exp	Imp	
Established Standards	7	1	2	0	18	10	4	0	8	0	0	0	50
Terminology Collections	11	0	1	0	35	17	16	3	5	1	2	0	91
Textbooks	7	1	2	0	38	4	5	0	1	0	0	0	58
Papers and Others	10	1	4	0	24	22	9	3	1	1	2	0	77
Total	35	3	9	0	115	53	34	6	15	2	4	0	276

[a]Section omitted for brevity.

TABLE V: Breakdown of identified Flaws by Domain by Source Tier.

Source Tier	Categories		Synonyms		Parents		Definitions		Labels		Traceability		Total
	Exp	Imp	Exp	Imp	Exp	Imp	Exp	Imp	Exp	Imp	Exp	Imp	
Established Standards	10	7	4	2	6	0	14	2	5	0	0	0	50
Terminology Collections	10	14	10	3	10	2	22	0	13	2	5	0	91
Textbooks	2	0	16	0	9	4	19	1	7	0	0	0	58
Papers and Others	9	13	14	9	11	3	8	0	7	2	1	0	77
Total	31	34	44	14	36	9	63	3	32	4	6	0	276

2) Less standardized (or "trustworthy"; see Section II-C) documents, such as terminology collections and textbooks (Sections III-A2 and III-A3, 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-A. 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 trusted" one (i.e., a previous row in the table; see Sections II-C and III-A). We also group these flaws by their rigidity (defined in Section II-B) by counting (Exp)licit and (Imp)licit flaws separately. Since we give each flaw a manifestation and a domain, the totals per source and grand totals in these tables are equal. As an example, the structure of tours can be defined as either quite general [16, p. 34] or "organized around a special focus" [17]. This is a contradictory definition, so it appears in Sections IV-A3 and IV-B4. Within these sections, we omit "less significant" flaws for brevity, then sort these flaws by their source tier (see Section III-A).

We only list flaws we automatically uncover based on their domain in their corresponding domain section for clarity, although they still contribute to the counts in Table IV. Moreover, certain "subsets" of testing contain many interconnected flaws, which we present in subsections as a "third view". This keeps related information together but causes a further apparent mismatch between the counts in Tables IV and V and the number of flaws in Sections IV-A and IV-B; we still include these flaws in the manifestation and domain counts. These subsets include functional testing (Section IV-C), recovery testing (Section IV-D), scalability testing (Section IV-E), and compatibility testing (Section IV-F).

A. Flaws by Manifestation

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

1) Mistakes: The following are cases where information is incorrect; this includes wrong definitions (see Section IV-B4), untrue claims about citations (see Section IV-B6), and simple typos:

- Since errors are distinct from defects/faults [11, p. 12-3], [9, pp. 128, 140], [10, pp. 399–400], error guessing should instead be called "defect guessing" if it is based on a "checklist of potential defects" [22, p. 29] or "fault guessing" if it is a "fault-based technique" [13, p. 4-9] that "anticipate[s] the most plausible faults in each SUT" [11, p. 5-13]. One (or both) of these proposed terms may be useful in tandem with "error guessing", which would focus on errors as traditionally defined; this would be a subapproach of error-based testing (implied by [10, p. 399]).
- Similarly, "fault seeding" is not a synonym of "error seeding" as claimed by [14, p. 165] and [10, p. 427]. The term "error seeding", also used by [7, p. 34], should be abandoned in favour of "fault seeding", as it is defined as the "process of intentionally adding known faults to those already in a computer program … [to] estimat[e] the number of faults remaining" [14,

p. 165] based on the ratio between the number of new faults and the number of introduced faults that were discovered [10, p. 427].
- Reference [17] classifies ML model testing as a test level, which it defines as "a specific instantiation of a test process": a vague definition that does not match the one in Table III.
- The terms "acceleration tolerance testing" and "acoustic tolerance testing" seem to only refer to software testing in [7, p. 56]; elsewhere, they seem to refer to testing the acoustic tolerance of rats [64] or the acceleration tolerance of astronauts [65, p. 11], aviators [66, pp. 27, 42], or catalysts [67, p. 1463], which don't exactly seem relevant…
- The differences between the terms "error", "failure", "fault", "defect" are significant and meaningful [9, pp. 128, 139–140], [10, pp. 399–400], [11, p. 12-3], but [15, 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"!
- Reference [1, p. 447] claims that "structural testing subsumes white box testing" but the two terms seem to describe the same thing: it says "structure tests are aimed at exercising the internal logic of a software system" and "in white box testing …, using detailed knowledge of code, one creates a battery of tests in such a way that they exercise all components of the code (say, statements, branches, paths)" on the same page!
- Reference [21, p. 46] says that the goal of negative testing is "showing that a component or system does not work" which is not true; if robustness is an important quality for the system, then testing the system "in a way for which it was not intended to be used" [17] (i.e., negative testing) is one way to help test this!
- Reference [21, p. 42] says "See boundary value analysis," for the glossary entry of "boundary value testing" but does not include "boundary value analysis" in the glossary.

2) Omissions: The following are cases where information (usually definitions; see Section IV-B4) should be included but is not:

- Integration testing, system testing, and system integration testing are all listed as "common test levels" [16, p. 12], [22, p. 6], but the latter two are not defined. This makes the relations between these three terms unclear; system integration testing is listed as a child of both integration testing [17] and system testing [7, p. 23].
- Similarly, component testing, integration testing, and component integration testing are all mentioned by [14], but the latter is only defined as "testing of groups of related components" [p. 82]. As before, the relations between these three terms are unclear; component integration testing is only listed as a child of integration testing [17].

3) Contradictions: The following are cases where multiple sources of information (sometimes within the same document!) disagree:

- Regression testing and retesting are sometimes given as two distinct approaches [16, p. 8], [7, p. 34], but sometimes regression testing is defined as a form of "selective retesting" [14, p. 372], [11, pp. 5-8, 6-5, 7-5 to 7-6], [68, p. 3]. Moreover, the two possible variations of regression testing given by [10, p. 411] are "retest-all" and "selective retest", which is possibly the source of the above misconception. This creates a cyclic relation between regression testing and selective retesting.
- "Software testing" is often defined to exclude static testing [1, p. 439], [7, p. 13], [69, p. 222], restricting "testing" to mean dynamic validation [11, p. 5-1] or verification "in which a system or component is executed" [14, p. 427]. However, "terminology is not uniform among different communities, and some use the term 'testing' to refer to static techniques[12] as well" [11, p. 5-2]. This is done by [16, pp. 16–17] and [24, pp. 8–9]; the authors of the former even explicitly exclude static testing in another document [14, p. 440]!
- A component is an "entity with discrete structure … within a system considered at a particular level of analysis" [70] and "the terms module, component, and unit [sic] are often used interchangeably or defined to be subelements of one another in different ways depending upon the context" with no standardized relationship [14, p. 82]. For example, [17] defines them as synonyms while [71, p. 107] says "components differ from classical modules for being re-used in different contexts independently of their development". Additionally, since components are structurally, functionally, or logically discrete [14, p. 419] and "can be tested in isolation" [17], "unit/component/module testing" could refer to the testing of both a module and a specific function in a module, introducing a further level of ambiguity.
- Performance testing and security testing are given as subtypes of reliability testing by [72], but these are all listed separately by [7, p. 53].
- Similarly, random testing is a subtechnique of specification-based testing [16, pp. 7, 22], [17], [11, p. 5-12], [22, pp. 5, 20, Fig. 2] but is listed separately by [7, p. 46].

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

- Path testing "aims to execute all entry-to-exit control flow paths in a SUT's control flow graph" [11, p. 5-13] (similar in [15, p. 119]), but [14, p. 316] adds that it can also be "designed to execute … selected paths."
- The structure of tours can be defined as either quite general [16, p. 34] or "organized around a special focus" [17].
- Alpha testing is performed by "users within the organization developing the software" [14, p. 17], "a small, selected group of potential users" [11, p. 5-8], or "roles outside the development organization" conducted "in the developer's test environment" [17].
- "Use case testing" is given as a synonym of "scenario testing" by [17] but listed separately by [16, Fig. 2] and described as a "common form of scenario testing" in [22, p. 20]. This implies that use case testing may instead be a child of user scenario testing (see Table VII).
- The terms "test level" and "test stage" are given as synonyms ([17]; implied by [24, p. 9]), but [11, p. 5-6] says "[test] levels can be distinguished based on the object of testing, the target, or on the purpose or objective" and calls the former "test stages", giving the term a child relation (see Section II-F) to "test level" instead. However, the examples listed—unit testing, integration testing, system testing, and acceptance testing [11, pp. 5-6 to 5-7]—are commonly categorized as "test levels" (see Section II-D).
- "Operational acceptance testing" and "production acceptance testing" are given as synonyms by [17] but listed separately by [7, p. 30].
- While [15, p. 120] implies that condition testing is a subtechnique of path testing, [10, Fig. 13.17] says that multiple condition coverage (which seems to be a synonym of condition coverage [p. 422]) does not subsume and is not subsumed by path coverage.
- Load testing is performed with loads "between anticipated conditions of low, typical, and peak usage" [16, p. 5] or loads that are as large as possible [15, p. 86].
- State testing requires that "all states in the state model … [are] 'visited' " in [22, p. 19] which is only one of its possible criteria in [15, pp. 82-83].
- Reference [14, p. 456] says system testing is "conducted on a complete, integrated system" (which [1, Tab. 12.3] and [10, p. 439] agree with), while [15, p. 109] says it can also be done on "at least a major portion" of the product.
- "Walkthroughs" and "structured walkthroughs" are given as synonyms by [17] but [1, p. 484] implies that they are different, saying a more structured walkthrough may have specific roles.
- Reference [15, p. 92] says that reviews are "the process[es] under which static white-box testing is performed" but correctness proofs are given as another example by [10, pp. 418–419].
- Reference [21, p. 46] says "negative testing is related to the testers' attitude rather than a specific test approach or test design technique"; while [22] seems to support this idea of negative testing being at a "higher" level than other approaches, it also implies that it is a test technique [pp. 10, 14].

4) Ambiguities: The following are cases where information (usually definitions or distinctions between terms; see Sections IV-B4 and IV-B5, respectively) is unclear:

- The distinctions between development testing [14, p. 136], developmental testing [7, p. 30], and developer testing [7, p. 39], [24, p. 11] are unclear and seem miniscule.
- Reference [17] defines "Machine Learning (ML) model testing" and "ML functional performance" in terms of "ML functional performance criteria", which is defined in terms of "ML functional performance metrics", which is defined as "a set of measures that relate to the functional correctness of an ML system". The use of "performance" (or "correctness") in these definitions is at best ambiguous and at worst incorrect.
- "Installability testing" is given as a test type [16, p. 22], [22, p. 38], [14, p. 228], while "installation testing" is given as a test level [10, p. 439]. Since "installation testing" is not given as an example of a test level throughout the sources that describe them (see Section II-D), it is likely that the term "installability testing" with all its related information should be used instead.
- Reference [17] claims that code inspections are related to peer reviews but [15, pp. 94–95] makes them quite distinct.

5) Overlaps: The following are cases where information overlaps, such as nonatomic definitions and terms (see Sections IV-B4 and IV-B5, respectively):

- Reference [16, p. 34] gives the "landmark tour" as an example of "a tour used for exploratory testing", but they also use the analogy of "a tour guide lead[ing] a tourist through the landmarks of a big city" to describe tours in general. Is the distinction between them the fact that landmark tours are pre-planned and follow a decided-upon sequence [p. 34]?
- ISO/IEC and IEEE say that "test level" and "test phase" are synonyms, both meaning a "specific instantiation of [a] test sub-process" [14, pp. 469, 470], [73, p. 9], but they have other definitions as well. "Test level" can also refer to the scope of a test process; for example, "across the whole organization" or only "to specific projects" [16, p. 24] and "test phase" can also refer to the "period of time in the software life cycle" when testing occurs [14, p. 470], usually after the implementation phase [pp. 420, 509], [23, p. 56].
- ISO/IEC and IEEE define "error" as "a human action that produces an incorrect result", but also as "an incorrect result" itself [9, p. 128]. Since faults

are inserted when a developer makes an error [11, p. 12-3], [9, pp. 128, 140], [10, pp. 399–400], this means that they are "incorrect results", making "error" and "fault" synonyms and the distinction between them less useful.
- Additionally, "error" can also be defined as "the difference between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition" [9, p. 128] (similar in [11, pp. 17-18 to 17-19, 18-7 to 18-8]). While this is a widely used definition, particularly in mathematics, it makes some test approaches ambiguous; for example, back-to-back testing is "testing in which two or more variants of a program are executed with the same inputs, the outputs are compared, and errors are analyzed in case of discrepancies" [9, p. 30] (similar in [17]), which seems to refer to this definition of "error".
- The SWEBOK Guide V4 defines "privacy testing" as testing that "assess[es] the security and privacy of users' personal data to prevent local attacks" [11, p. 5-10]; this seems to overlap (both in scope and name) with the definition of "security testing" in [16, p. 7]: testing "conducted to evaluate the degree to which a test item, and associated data and information, [sic] are protected so that" only "authorized persons or systems" can use them as intended.
- "Orthogonal array testing" [11, pp. 5-1, 5-11] and "operational acceptance testing" [7, p. 30] have the same acronym ("OAT").

B. Flaws by Domain

The following sections list observed flaws grouped by what information is flawed as presented in Section II-A2. This includes test approach categories (Section IV-B1), synonym relations (Section IV-B2), parent-child relations (Section IV-B3), definitions (Section IV-B4), labels (Section IV-B5), and traceability information (Section IV-B6).

1) Approach Category Flaws: While the IEEE categorization of testing approaches described in Table III 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 [14, 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" [22, p. 8]. For example, "the test coverage items derived by applying equivalence partitioning can be used to identify the input parameters of test cases derived for scenario testing" [p. 8]. Even the categories themselves are not consistently defined, and some approaches are categorized differently by different sources; these differences are tracked so they can be analyzed more systematically.

- Reference [17] classifies ML model testing as a test level, which it defines as "a specific instantiation of a test process": a vague definition that does not match the one in Table III.
- Reference [21, p. 46] says "negative testing is related to the testers' attitude rather than a specific test approach or test design technique"; while [22] seems to support this idea of negative testing being at a "higher" level than other approaches, it also implies that it is a test technique [pp. 10, 14].

Some category flaws can be detected automatically, such as test approaches with more than one category. These are given in Table VI and include experience-based testing, which is of particular note. ISO/IEC and IEEE categorize experience-based testing as both a test design technique and a test practice on the same page—twice [16, Fig. 2, p. 34]! These authors say "experience-based testing practices like exploratory testing … are not … techniques for designing test cases", although they "can use … test techniques" [22, p. viii], which they support in [16, p. 33] along with scripted testing. This implies that "experience-based test design techniques" are used by the practice of experience-based testing which is not itself a test technique (and similarly with scripted testing). If this is the case, it blurs the line between "practice" and "technique", which may explain why experience-based testing is categorized inconsistently in the literature.

Subapproaches of experience-based tesing, such as error guessing and exploratory testing, are also categorized ambiguously, causing confusion on how categories and parent-child relations (see Section II-F) interact. Reference [16, p. 34] says that a previous standard [22] "describes the experience-based test design technique of error guessing. Other experience-based test practices include (but are not limited to) exploratory testing …, tours, attacks, and checklist-based testing". This seems to imply that error guessing is both a technique and a practice, which does not make sense if these categories are orthogonal. These kinds of inconsistencies between parent and child test approach categorizations may indicate that categories are not transitive or that more thought must be given to them.

2) Synonym Relation Flaws: As mentioned in Section II-E, synonyms do not inherently signify a flaw. Unfortunately, there are many instances of incorrect or ambiguous synonyms, such as the following:

- A component is an "entity with discrete structure … within a system considered at a particular level of analysis" [70] and "the terms module, component, and unit [sic] are often used interchangeably or defined to be subelements of one another in different ways depending upon the context" with no standardized relationship [14, p. 82]. For example, [17] defines them as synonyms while [71, p. 107] says "components differ from classical modules for being re-used in different contexts independently of their development". Additionally, since components are structurally, func-

TABLE VI: Test approaches with more than one category.

Approach	Category 1	Category 2
Capacity Testing	Technique [22, p. 38]	Type [16, p. 22], [7, p. 53], [73, p. 2]
Checklist-based Testing	Practice [16, p. 34]	Technique [17]
Data-driven Testing	Practice [16, p. 22]	Technique [21, p. 43]
End-to-end Testing	Type [17]	Technique [7, p. 47], [60, pp. 601, 603, 605–606]
Endurance Testing	Technique [22, p. 38]	Type [73, p. 2]
Experience-based Testing	Practice [16, pp. 22, 34], [22, p. viii]	Technique [16, pp. 4, 22], [17], [11, p. 5-13], [7, pp. 46, 50], [22, p. 4]
Exploratory Testing	Practice [16, pp. 20, 22, 34], [22, p. viii]	Technique [16, p. 34], [11, p. 5-14], [7, p. 50]
Load Testing	Technique [22, p. 38]	Type [16, pp. 5, 20, 22], [17], [14, p. 253]
Model-based Testing	Practice [16, p. 22], [22, p. viii]	Technique [21, p. 4]
Mutation Testing	Methodology [14, p. 286]	Technique [11, p. 5-15], [10, pp. 428–429]
Performance Testing	Technique [22, p. 38]	Type [16, pp. 7, 22, 26–27], [22, p. 7]
Stress Testing	Technique [22, p. 38]	Type [16, pp. 9, 22], [14, p. 442]

tionally, or logically discrete [14, p. 419] and "can be tested in isolation" [17], "unit/component/module testing" could refer to the testing of both a module and a specific function in a module, introducing a further level of ambiguity.

- ISO/IEC and IEEE say that "test level" and "test phase" are synonyms, both meaning a "specific instantiation of [a] test sub-process" [14, pp. 469, 470], [73, p. 9], but they have other definitions as well. "Test level" can also refer to the scope of a test process; for example, "across the whole organization" or only "to specific projects" [16, p. 24] and "test phase" can also refer to the "period of time in the software life cycle" when testing occurs [14, p. 470], usually after the implementation phase [pp. 420, 509], [23, p. 56].
- "Use case testing" is given as a synonym of "scenario testing" by [17] but listed separately by [16, Fig. 2] and described as a "common form of scenario testing" in [22, p. 20]. This implies that use case testing may instead be a child of user scenario testing (see Table VII).
- The terms "test level" and "test stage" are given as synonyms ([17]; implied by [24, p. 9]), but [11, p. 5-6] says "[test] levels can be distinguished based on the object of testing, the target, or on the purpose or objective" and calls the former "test stages", giving the term a child relation (see Section II-F) to "test level" instead. However, the examples listed—unit testing, integration testing, system testing, and acceptance testing [11, pp. 5-6 to 5-7]—are commonly categorized as "test levels" (see Section II-D).
- "Operational acceptance testing" and "production acceptance testing" are given as synonyms by [17] but listed separately by [7, p. 30].
- The differences between the terms "error", "failure", "fault", "defect" are significant and meaningful [9, pp. 128, 139–140], [10, pp. 399–400], [11, p. 12-3], but [15, 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"!
- Reference [17] claims that code inspections are related to peer reviews but [15, pp. 94–95] makes them quite distinct.
- "Walkthroughs" and "structured walkthroughs" are given as synonyms by [17] but [1, p. 484] implies that they are different, saying a more structured walkthrough may have specific roles.
- Reference [1, p. 447] claims that "structural testing subsumes white box testing" but the two terms seem to describe the same thing: it says "structure tests are aimed at exercising the internal logic of a software system" and "in white box testing …, using detailed knowledge of code, one creates a battery of tests in such a way that they exercise all components of the code (say, statements, branches, paths)" on the same page!

There are also cases in which a term is given as a synonym to two (or more) terms that are not synonyms themselves. Sometimes, these terms are synonyms; for example, [17] says "use case testing", "user scenario testing", and "scenario testing" are all synonyms (although there may be a slight distinction; see Table VII and Section IV-B2). However, this does not always make sense. We identify nine such cases through automatic analysis of the generated graphs. The following three are the most prominent examples:

1) Invalid Testing:
   - Error Tolerance Testing [21, p. 45]
   - Negative Testing [17] (implied by [22, p. 10])
2) Soak Testing:
   - Endurance Testing [22, p. 39]
   - Reliability Testing[13] [24, Tab. 2], [32, Tab. 1, p. 26]
3) Link Testing:
   - Branch Testing (implied by [22, p. 24])

[13]Endurance testing is given as a child of reliability testing by [7, p. 55], although the terms are not synonyms.

TABLE VII: 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	[22, p. 19]	[21, p. 15]
Co-existence Testing	→	Compatibility Testing	[16, p. 3], [22, Tab. A.1], [72]	[22, p. 37]
Fault Tolerance Testing	→	Robustness Testing[a]	[7, p. 56]	[17]
Functional Testing	→	Specification-based Testing[b]	[22, p. 38]	[14, p. 196], [10, p. 399], [21, p. 44]
Orthogonal Array Testing	→	Pairwise Testing	[62, p. 1055]	[11, p. 5-11], [63, p. 473]
Path Testing	→	Exhaustive Testing	[1, pp. 466-467, 476]	[10, p. 421]
Performance Testing	→	Performance-related Testing	[16, p. 22], [22, p. 38]	[57, p. 1187]
Static Analysis	→	Static Testing	[16, pp. 9, 17, 25, 28], [17]	[1, p. 438]
Structural Testing	→	Structure-based Testing	[15, pp. 105-121]	[16, p. 9], [17], [14, pp. 443-444]
Use Case Testing	→	Scenario Testing[c]	[22, p. 20]	[17], [21, pp. 47-49]

[a]Fault tolerance testing may also be a subapproach of reliability testing [14, p. 375], [11, p. 7-10], which is distinct from robustness testing [7, p. 53].
[b]See Section IV-C1.
[c]See Section IV-B2.

- Component Integration Testing [21, p. 45]
- Integration Testing (implied by [24, p. 13])

3) Parent-Child Relation Flaws: Parent-child relations (defined in Section II-F) are also not immune to flaws; for example, performance testing and security testing are given as subtypes of reliability testing by [72], but these are all listed separately by [7, p. 53]. Additionally, some self-referential definitions imply that a test approach is a parent of itself. Since these are by nature self-contained within a given source, these are counted once as explicit flaws within their sources in Tables IV and V. For example, performance and usability testing are both given as subapproaches of themselves [24, Tab. 2], [32, Tab. 1].

There are also pairs of synonyms where one is described as a subapproach of the other, abusing the meaning of "synonym" and causing confusion. We identify 16 of these pairs through automatic analysis of the generated graphs, with the most prominent given in Table VII. 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"[14], this overlooks the effects of input data [15, p. 121], [22, p. 129], [1, p. 467] and implementation issues [p. 476] on the code's behaviour. Exhaustive testing requires "all combinations of input values and preconditions … [to be] tested" [16, p. 4] (similar in [17], [15, p. 121]).

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

- Reference [16, p. 34] gives the "landmark tour" as an example of "a tour used for exploratory testing", but they also use the analogy of "a tour guide lead[ing] a tourist through the landmarks of a big city" to describe tours in general. Is the distinction between them the fact that landmark tours are pre-planned and follow a decided-upon sequence [p. 34]?
- Integration testing, system testing, and system integration testing are all listed as "common test levels" [16, p. 12], [22, p. 6], but the latter two are not defined. This makes the relations between these three terms unclear; system integration testing is listed as a child of both integration testing [17] and system testing [7, p. 23].
- Similarly, component testing, integration testing, and component integration testing are all mentioned by [14], but the latter is only defined as "testing of groups of related components" [p. 82]. As before, the relations between these three terms are unclear; component integration testing is only listed as a child of integration testing [17].
- "Software testing" is often defined to exclude static testing [1, p. 439], [7, p. 13], [69, p. 222], restricting "testing" to mean dynamic validation [11, p. 5-1] or verification "in which a system or component is executed" [14, p. 427]. However, "terminology is not uniform among different communities, and some use the term 'testing' to refer to static techniques[15] as well" [11, p. 5-2]. This is done by [16, pp. 16–17] and [24, pp. 8–9]; the authors of the former even explicitly exclude static testing in another document [14, p. 440]!
- ISO/IEC and IEEE define "error" as "a human action that produces an incorrect result", but also as "an incorrect result" itself [9, p. 128]. Since faults are inserted when a developer makes an error [11, p. 12-3], [9, pp. 128, 140], [10, pp. 399–400], this means that they are "incorrect results", making "error" and "fault" synonyms and the distinction between them less useful.
- Additionally, "error" can also be defined as "the

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

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

difference between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition" [9, p. 128] (similar in [11, pp. 17-18 to 17-19, 18-7 to 18-8]). While this is a widely used definition, particularly in mathematics, it makes some test approaches ambiguous; for example, back-to-back testing is "testing in which two or more variants of a program are executed with the same inputs, the outputs are compared, and errors are analyzed in case of discrepancies" [9, p. 30] (similar in [17]), which seems to refer to this definition of "error".

- The SWEBOK Guide V4 defines "privacy testing" as testing that "assess[es] the security and privacy of users' personal data to prevent local attacks" [11, p. 5-10]; this seems to overlap (both in scope and name) with the definition of "security testing" in [16, p. 7]: testing "conducted to evaluate the degree to which a test item, and associated data and information, [sic] are protected so that" only "authorized persons or systems" can use them as intended.
- Path testing "aims to execute all entry-to-exit control flow paths in a SUT's control flow graph" [11, p. 5-13] (similar in [15, p. 119]), but [14, p. 316] adds that it can also be "designed to execute … selected paths."
- The structure of tours can be defined as either quite general [16, p. 34] or "organized around a special focus" [17].
- Alpha testing is performed by "users within the organization developing the software" [14, p. 17], "a small, selected group of potential users" [11, p. 5-8], or "roles outside the development organization" conducted "in the developer's test environment" [17].
- Reference [17] defines "Machine Learning (ML) model testing" and "ML functional performance" in terms of "ML functional performance criteria", which is defined in terms of "ML functional performance metrics", which is defined as "a set of measures that relate to the functional correctness of an ML system". The use of "performance" (or "correctness") in these definitions is at best ambiguous and at worst incorrect.
- Load testing is performed with loads "between anticipated conditions of low, typical, and peak usage" [16, p. 5] or loads that are as large as possible [15, p. 86].
- State testing requires that "all states in the state model … [are] 'visited' " in [22, p. 19] which is only one of its possible criteria in [15, pp. 82-83].
- Reference [14, p. 456] says system testing is "conducted on a complete, integrated system" (which [1, Tab. 12.3] and [10, p. 439] agree with), while [15, p. 109] says it can also be done on "at least a major portion" of the product.
- Reference [15, p. 92] says that reviews are "the process[es] under which static white-box testing is performed" but correctness proofs are given as another example by [10, pp. 418–419].
- Reference [21, p. 46] says that the goal of negative testing is "showing that a component or system does not work" which is not true; if robustness is an important quality for the system, then testing the system "in a way for which it was not intended to be used" [17] (i.e., negative testing) is one way to help test this!

5) Label Flaws: While some flaws exist because the definition of a term is wrong, others exist because a term's name or label is wrong! Terms can be thought of as definition-label pairs, but there is a meaningful distinction between definition flaws and label flaws, which we define in Section II-A2. We observe the following label flaws:

- Since errors are distinct from defects/faults [11, p. 12-3], [9, pp. 128, 140], [10, pp. 399–400], error guessing should instead be called "defect guessing" if it is based on a "checklist of potential defects" [22, p. 29] or "fault guessing" if it is a "fault-based technique" [13, p. 4-9] that "anticipate[s] the most plausible faults in each SUT" [11, p. 5-13]. One (or both) of these proposed terms may be useful in tandem with "error guessing", which would focus on errors as traditionally defined; this would be a subapproach of error-based testing (implied by [10, p. 399]).
- Similarly, "fault seeding" is not a synonym of "error seeding" as claimed by [14, p. 165] and [10, p. 427]. The term "error seeding", also used by [7, p. 34], should be abandoned in favour of "fault seeding", as it is defined as the "process of intentionally adding known faults to those already in a computer program … [to] estimat[e] the number of faults remaining" [14, p. 165] based on the ratio between the number of new faults and the number of introduced faults that were discovered [10, p. 427].
- The distinctions between development testing [14, p. 136], developmental testing [7, p. 30], and developer testing [7, p. 39], [24, p. 11] are unclear and seem miniscule.
- The terms "acceleration tolerance testing" and "acoustic tolerance testing" seem to only refer to software testing in [7, p. 56]; elsewhere, they seem to refer to testing the acoustic tolerance of rats [64] or the acceleration tolerance of astronauts [65, p. 11], aviators [66, pp. 27, 42], or catalysts [67, p. 1463], which don't exactly seem relevant…
- "Orthogonal array testing" [11, pp. 5-1, 5-11] and "operational acceptance testing" [7, p. 30] have the same acronym ("OAT").
- "Installability testing" is given as a test type [16, p. 22], [22, p. 38], [14, p. 228], while "installation testing" is given as a test level [10, p. 439]. Since "installation testing" is not given as an example of a test level throughout the sources that describe them (see Section II-D), it is likely that the term

"installability testing" with all its related information should be used instead.

6) Traceability Flaws: Sometimes a document cites another for a piece of information that does not appear! For example, [36, p. 184] claims that [31] defines "prime path coverage", but it does not.

C. Functional Testing

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

1) Specification-based Testing: This is defined as "testing in which the principal test basis is the external inputs and outputs of the test item" [16, p. 9]. This agrees with a definition of "functional testing": "testing that … focuses solely on the outputs generated in response to selected inputs and execution conditions" [14, p. 196]. Notably, [14] lists both as synonyms of "black-box testing" [pp. 431, 196, respectively], despite them sometimes being defined separately. For example, the International Software Testing Qualifications Board (ISTQB) defines "specification-based testing" as "testing based on an analysis of the specification of the component or system" and "functional testing" as "testing performed to evaluate if a component or system satisfies functional requirements" [17]. Overall, specification-based testing [16, pp. 2-4, 6-9, 22] is a test design technique used to "derive corresponding test cases" [16, p. 11] from "selected inputs and execution conditions" [14, p. 196].

2) Correctness Testing: Reference [11, p. 5-7] says "test cases can be designed to check that the functional specifications are correctly implemented, which is variously referred to in the literature as conformance testing, correctness testing or functional testing"; this mirrors previous definitions of "functional testing" [16, p. 21], [14, p. 196] but groups it with "correctness testing". Since "correctness" is a software quality [14, p. 104], [11, p. 3-13] which is what defines a "test type" [16, p. 15], it seems consistent to label "functional testing" as a "test type" [16, pp. 15, 20, 22], [22, pp. 7, 38, Tab. A.1], [26, p. 4]. However, this conflicts with its categorization as a "technique" if considered a synonym of specification-based testing (see Section IV-C1). Additionally, "correctness testing" is listed separately from "functionality testing" by [7, p. 53].

3) Conformance Testing: Testing that ensures "that the functional specifications are correctly implemented", and can be called "conformance testing" or "functional testing" [11, p. 5-7]. "Conformance testing" is later defined as testing used "to verify that the SUT conforms to standards, rules, specifications, requirements, design, processes, or practices" [11, p. 5-7]. This definition seems to be a superset of testing methods mentioned earlier as the latter includes "standards, rules, requirements, design, processes, … [and]" practices in addition to specifications!

A complicating factor is that "compliance testing" is also (plausibly) given as a synonym of "conformance testing" [21, p. 43]. However, "conformance testing" can also be defined as testing that evaluates the degree to which "results … fall within the limits that define acceptable variation for a quality requirement" [14, p. 93], which seems to describe something different.

4) Functional Suitability Testing: Procedure testing is called a "type of functional suitability testing" [16, p. 7] but no definition of that term is given. "Functional suitability" is the "capability of a product to provide functions that meet stated and implied needs of intended users when it is used under specified conditions", including meeting "the functional specification" [72]. This seems to align with the definition of "functional testing" as related to "black-box/specification-based testing". "Functional correctness", a child of "functional suitability", is the "capability of a product to provide accurate results when used by intended users" [72] and seems to align with the quality/ies that would be tested by "correctness" testing.

5) Functionality Testing: "Functionality" is defined as the "capabilities of the various … features provided by a product" [14, p. 196] and is said to be a synonym of "functional suitability" [17], although it seems like it should really be a synonym of "functional completeness" based on [72], which would make "functional suitability" a subapproach. Its associated test type is implied to be a subapproach of build verification testing [17] and made distinct from "functional testing" [24, Tab. 2]. "Functionality testing" is listed separately from "correctness testing" by [7, p. 53].

D. 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" [72] (similar in [11, p. 7-10]) so that the system "can perform required functions" [14, p. 370]. It is also called "recoverability testing" [21, p. 47] and potentially "restart & recovery (testing)" [24, Fig. 5]. The following terms, along with "recovery testing" itself [16, p. 22] are all classified as test types, and the relations between them can be found in Figure 3a.

- Recoverability Testing: Testing "how well a system or software can recover data during an interruption or failure" [11, p. 7-10] (similar in [72]) and "re-establish the desired state of the system" [72]. Synonym for "recovery testing" in [21, p. 47].
- Disaster/Recovery Testing serves to evaluate if a system can "return to normal operation after a hardware or software failure" [14, 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" [22, p. 37]. These two definitions seem to describe different aspects of the system, where the first is intrinsic to the hardware/software and the second might not be.

- Backup and Recovery Testing "measures the degree to which system state can be restored from backup within specified parameters of time, cost, completeness, and accuracy in the event of failure" [73, p. 2]. This may be what is meant by "recovery testing" in the context of performance-related testing and seems to correspond to the definition of "disaster/recovery testing" in [14, p. 140].
- Backup/Recovery Testing: Testing that determines the ability "to restor[e] from back-up memory in the event of failure, without transfer[ing] to a different operating site or back-up system" [22, p. 37]. This seems to correspond to the definition of "disaster/recovery testing" in [22, p. 37]. It is also given as a subtype of "disaster/recovery testing", even though that tests if "operation of the test item can be transferred to a different operating site" [p. 37]. It also seems to overlap with "backup and recovery testing", which adds confusion.
- Failover/Recovery Testing: Testing that determines the ability "to mov[e] to a back-up system in the event of failure, without transfer[ing] to a different operating site" [22, p. 37]. This is given as a subtype of "disaster/recovery testing", even though that tests if "operation of the test item can be transferred to a different operating site" [p. 37].
- Failover Testing: Testing that "validates the SUT's ability to manage heavy loads or unexpected failure to continue typical operations" [11, p. 5-9] by entering a "backup operational mode in which [these responsibilities] … are assumed by a secondary system" [17]. While not explicitly related to recovery, "failover/recovery testing" also describes the idea of "failover", and [7, p. 56] uses the term "failover and recovery testing", which could be a synonym of both of these terms.

E. Scalability Testing

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

1) ISO/IEC and IEEE give "scalability testing" as a synonym of "capacity testing" [22, p. 39] while other sources differentiate between the two [7, p. 53], [37, pp. 22-23]
2) ISO/IEC and IEEE give the external modification of the system as part of "scalability" [22, p. 39], while [72] implies that it is limited to the system itself
3) The SWEBOK Guide V4's definition of "scalability testing" [11, p. 5-9] is really a definition of usability testing!

F. Compatibility Testing

"Compatibility testing" is defined as "testing that measures the degree to which a test item can function satisfactorily alongside other independent products in a shared environment (co-existence), and where necessary, exchanges information with other systems or components (interoperability)" [16, p. 3]. This definition is nonatomic as it combines the ideas of "co-existence" and "interoperability". The term "interoperability testing" is not defined, but is used three times [16, pp. 22, 43] (although the third usage seems like it should be "portability testing"). This implies that "co-existence testing" and "interoperability testing" should be defined as their own terms, which is supported by definitions of "co-existence" and "interoperability" often being separate [14, pp. 73, 237], [17], the definition of "interoperability testing" from [14, p. 238], and the decomposition of "compatibility" into "co-existence" and "interoperability" by [72]! The "interoperability" element of "compatibility testing" is explicitly excluded by [22, p. 37], (incorrectly) implying that "compatibility testing" and "co-existence testing" are synonyms. Furthermore, the definition of "compatibility testing" in [21, p. 43] unhelpfully says "See interoperability testing", adding another layer of confusion to the direction of their relationship.

V. Recommendations

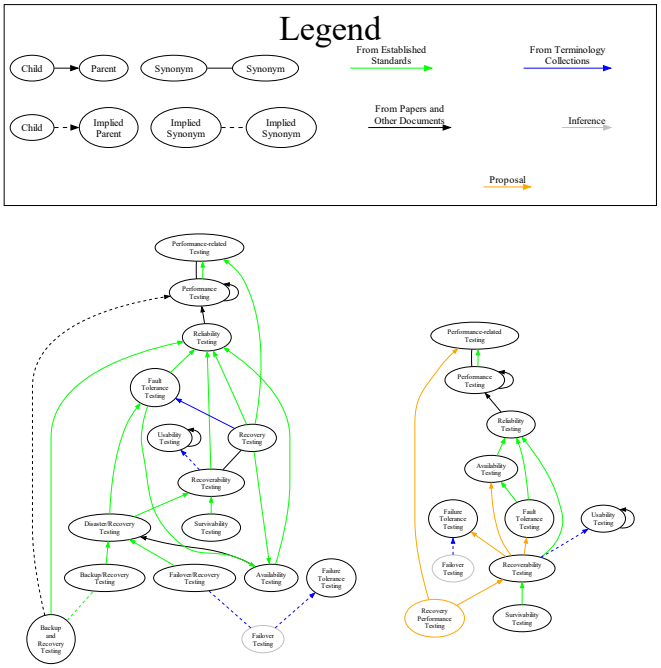
As we have shown in Section IV, "testing is a mess" [Mosser, 2023, priv. comm.]! It will take a lot of time, effort, expertise, and training to organize these terms (and their relations) logically. However, the hardest step is often the first one, so we attempt to give some examples of how this "rationalization" 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 (Section IV-B2) 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), and performance-related testing (Section V-C). Graphical representations (described in Section III-C) of these subsets are given in Figures 3 to 5, in which arrows



(a) Graph of current relations. (b) Graph of proposed relations.

Fig. 3: Graphs of relations between terms related to recovery testing.

representing relations between approaches are coloured based on the source tier (see Section III-A) that defines them. Any added approaches or relations are colored orange.

A. Recovery Testing

The following terms should be used in place of the current terminology to more clearly distinguish between different recovery-related test approaches. The result of the proposed terminology, along with their relations, is demonstrated in Figure 3b.

- Recoverability Testing: "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" [72] (similar in [11, p. 7-10]) so that the system "can perform required functions" [14, p. 370]. "Recovery testing" will be a synonym, as in [21, p. 47], since it is the more prevalent term throughout various sources, although "recoverability testing" is preferred to indicate that this explicitly focuses on the ability to recover, not the performance of recovering.
- Failover Testing: Testing that "validates the SUT's ability to manage heavy loads or unexpected failure to continue typical operations" [11, p. 5-9] by entering a "backup operational mode in which [these responsibilities] … are assumed by a secondary system" [17]. This will replace "failover/recovery testing", since it is more clear, and since this is one way that a system can recover from failure, it will be a subset of "recovery testing".
- Transfer Recovery Testing: Testing to evaluate if, in the case of a failure, "operation of the test item can be transferred to a different operating site and … be transferred back again once the failure has been resolved" [22, p. 37]. This replaces the second definition of "disaster/recovery testing", since the first is just a description of "recovery testing", and could potentially be considered as a kind of failover testing. This may not be intrinsic to the hardware/software (e.g., may be the responsibility of humans/processes).
- Backup Recovery Testing: Testing that determines the ability "to restor[e] from back-up memory in the event of failure" [22, p. 37]. The qualification that this occurs "without transfer[ing] to a different operating site or back-up system" [p. 37] could be made explicit, but this is implied since it is separate from transfer recovery testing and failover testing, respectively.
- Recovery Performance Testing: Testing "how well a system or software can recover … [from] an interruption or failure" [11, p. 7-10] (similar in [72]) "within specified parameters of time, cost, completeness, and accuracy" [73, p. 2]. The distinction between the performance-related elements of recovery testing seemed to be meaningful, but was not captured consistently by the literature. This will be a subset of "performance-related testing" as "recovery testing" is in [16, p. 22]. This could also be extended into testing the performance of specific elements of recovery (e.g., failover performance testing), but this be too fine-grained and may better be captured as an orthogonally derived test approach.

B. Scalability Testing

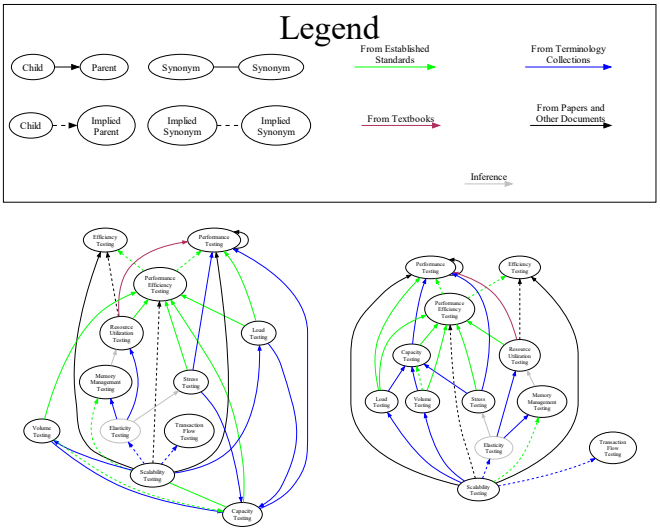
The ambiguity around scalability testing found in the literature is resolved and/or explained by other sources! [22, p. 39] gives "scalability testing" as a synonym of "capacity testing", defined as the testing of a system's ability to "perform under conditions that may need to be supported in the future", which "may include assessing what level of additional resources (e.g. memory, disk capacity, network bandwidth) will be required to support anticipated future loads". This focus on "the future" is supported by [17], which defines "scalability" as "the degree to which a component or system can be adjusted for changing capacity". In contrast, capacity testing focuses on the system's present state, evaluating the "capability of a product to meet requirements for the maximum limits of a product parameter", such as the number of concurrent users, transaction throughput, or database size [72]. Because of this nuance, it makes more sense to consider these terms separate and not synonyms, as done by [7, p. 53] and [37, pp. 22-23].

Unfortunately, only focusing on future capacity requirements still leaves room for ambiguity. While the previous definition of "scalability testing" includes the external modification of the system, [72] describes it as testing the "capability of a product to handle growing or shrinking workloads or to adapt its capacity to handle variability", implying that this is done by the system itself. The potential reason for this is implied by [11, p. 5-9]'s claim that one objective of elasticity testing is "to evaluate scalability": [72]'s notion of "scalability" likely refers more accurately to "elasticity"! This also makes sense in the context of other definitions provided by [11]:

- Scalability: "the software's ability to increase and scale up on its nonfunctional requirements, such as load, number of transactions, and volume of data" [p. 5-5]. Based on this definition, scalability testing is then a subtype of load testing and volume testing, as well as potentially transaction flow testing.
- Elasticity Testing[16]: testing that "assesses the ability of the SUT … to rapidly expand or shrink compute, memory, and storage resources without compromising the capacity to meet peak utilization" [p. 5-9]. Based on this definition, elasticity testing is then a subtype of memory management testing (with both being a subtype of resource utilization testing) and stress testing.

This distinction is also consistent with how the terms are used in industry: [39] says that scalability is the ability to "increase … performance or efficiency as demand increases over time", while elasticity allows a system to "tackle changes in the workload [that] occur for a short period".

To make things even more confusing, the SWEBOK Guide V4 says "scalability testing evaluates the capability to use and learn the system and the user documentation" and "focuses on the system's effectiveness in supporting user tasks and the ability to recover from user errors" [11, p. 5-9]. This seems to define "usability testing" with elements of functional and recovery testing, which is completely separate from the definitions of "scalability", "capacity", and "elasticity testing"! This definition should simply be disregarded, since it is inconsistent with the rest of the literature. The removal of the previous two synonym relations is demonstrated in Figure 4b.



(a) Graph of current relations.

(b) Graph of proposed relations.

Fig. 4: Graphs of relations between terms related to scalability testing.

C. Performance(-related) Testing

"Performance testing" is defined as testing "conducted to evaluate the degree to which a test item accomplishes its designated functions" [14, p. 320], [16, p. 7] (similar in [22, pp. 38-39], [57, p. 1187]). It does this by "measuring the performance metrics" [57, p. 1187] (similar in [17]) (such as the "system's capacity for growth" [32, 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" [14, p. 320], [16, p. 7] (similar in [57, p. 1187]), or
2) "under a 'typical' load" [22, 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' " [22, p. 38], along with other approaches like load and capacity testing [16, 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" [14, p. 320], [16, p. 7] (similar in [57, p. 1187]), and "performance testing" is a subapproach of this performed "under a 'typical' load" [22, 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" [14, p. 253], [17], [16, p. 5],

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

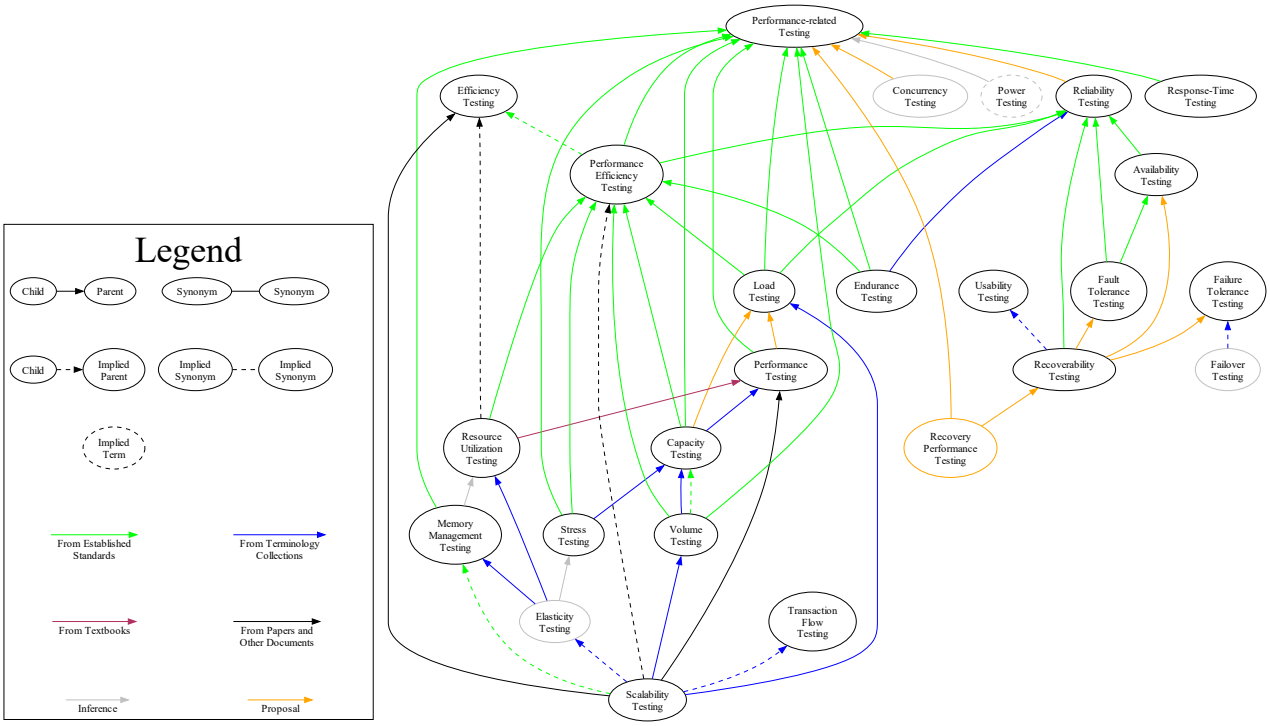


Fig. 5: Proposed relations between rationalized "performance-related testing" terms.

[22, 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 "self-loops" mentioned in Section IV-B3 provide no new information and can be removed. Similarly, the term "soak testing" can be removed. Since it is given as a synonym to both "endurance testing" and "reliability testing" (see Section IV-B2), 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 made implicitly) result in the relations shown in Figure 5.

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. Finally, Dr. Spencer Smith and Dr. Jacques Carette have been great supervisors and valuable sources of guidance and feedback.

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. [Online]. Available: https://waseda.app.box.com/v/SWEBOK4-book

[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] 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

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

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

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

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

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

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

[20] 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

[21] 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

[22] ISO/IEC and IEEE, "ISO/IEC/IEEE International Standard - Software and systems engineering –Software testing –Part 4: Test techniques," ISO/IEC/IEEE 29119-4:2021(E), Oct. 2021.

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

[24] 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

[25] 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.

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

[27] P. Chalin, J. R. Kiniry, G. T. Leavens, and E. Poll, "Beyond Assertions: Advanced Specification and Verification with JML and ESC/Java2," in Formal Methods for Components and Objects, F. S. de Boer, M. M. Bonsangue, S. Graf, and W.-P. de Roever, Eds. Berlin, Heidelberg: Springer, 2006, pp. 342–363.

[28] S. K. Lahiri, K. L. McMillan, R. Sharma, and C. Hawblitzel, "Differential Assertion Checking," in Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, ser. ESEC/FSE 2013. New York, NY, USA: Association for Computing Machinery, Aug. 2013, pp. 345–355. [Online]. Available: https://dl.acm.org/doi/10.1145/2491411.2491452

[29] ChatGPT (GPT-4o), "Defect Clustering Testing," Nov. 2024. [Online]. Available: https://chatgpt.com/share/67463dd1-d0a8-8012-937b-4a3db0824dcf

[30] 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

[31] 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

[32] 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

[33] 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

[34] 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

[35] H. Washizaki, "Software Engineering Body of Knowledge (SWEBOK)," Feb. 2025. [Online]. Available: https://www.computer.org/education/bodies-of-knowledge/software-engineering/

[36] 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

[37] M. Bas, "Data Backup and Archiving," Bachelor Thesis, Czech University of Life Sciences Prague, Praha-Suchdol, Czechia, Mar. 2024. [Online]. Available: https://theses.cz/id/60licg/zaverecna_prace_Archive.pdf

[38] LambdaTest, "What is Operational Testing: Quick Guide With

Examples," 2024. [Online]. Available: https://www.lambdatest.com/learning-hub/operational-testing

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

[40] 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

[41] 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

[42] ——, "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

[43] 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

[44] 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.

[45] 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.

[46] 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

[47] 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

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

[49] J. Berdine, C. Calcagno, and P. W. O'Hearn, "Smallfoot: Modular Automatic Assertion Checking with Separation Logic," in Formal Methods for Components and Objects, F. S. de Boer, M. M. Bonsangue, S. Graf, and W.-P. de Roever, Eds. Berlin, Heidelberg: Springer, 2006, pp. 115–137.

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

[51] P. Godefroid and D. Luchaup, "Automatic Partial Loop Summarization in Dynamic Test Generation," in Proceedings of the 2011 International Symposium on Software Testing and Analysis, ser. ISSTA '11. New York, NY, USA: Association for Computing Machinery, Jul. 2011, pp. 23–33. [Online]. Available: https://dl.acm.org/doi/10.1145/2001420.2001424

[52] S. Preuße, H.-C. Lapp, and H.-M. Hanisch, "Closed-loop System Modeling, Validation, and Verification," in Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies & Factory Automation (ETFA 2012). Krakow, Poland: IEEE, 2012, pp. 1–8. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6489679

[53] P. Forsyth, T. Maguire, and R. Kuffel, "Real Time Digital Simulation for Control and Protection System Testing," in 2004 IEEE 35th Annual Power Electronics Specialists Conference (IEEE Cat. No.04CH37551), vol. 1. Aachen, Germany: IEEE, 2004, pp. 329–335.

[54] C. Zhou, Q. Yu, and L. Wang, "Investigation of the Risk of Electromagnetic Security on Computer Systems," International Journal of Computer and Electrical Engineering, vol. 4, no. 1, p. 92, Feb. 2012, publisher: IACSIT Press. [Online]. Available: http://ijcee.org/papers/457-JE504.pdf

[55] ISO, "ISO 21384-2:2021 - Unmanned aircraft systems –Part 2: UAS components," ISO 21384-2:2021, Dec. 2021. [Online]. Available: https://www.iso.org/obp/ui#iso:std:iso:21384:-2:ed-1:v1:en

[56] C. Jard, T. Jéron, L. Tanguy, and C. Viho, "Remote testing can be as powerful as local testing," in Formal Methods for Protocol Engineering and Distributed Systems: Forte XII / PSTV XIX'99, ser. IFIP Advances in Information and Communication Technology, J. Wu, S. T. Chanson, and Q. Gao, Eds., vol. 28. Beijing, China: Springer, Oct. 1999, pp. 25–40. [Online]. Available: https://doi.org/10.1007/978-0-387-35578-8_2

[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] S. R. Choudhary, H. Versee, and A. Orso, "A Cross-browser Web Application Testing Tool," in 2010 IEEE International Conference on Software Maintenance. Timisoara, Romania: IEEE, Sep. 2010, pp. 1–6, iSSN: 1063-6773. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/5609728

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

[60] 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

[61] R. S. Sangwan and P. A. LaPlante, "Test-Driven Development in Large Projects," IT Professional, vol. 8, no. 5, pp. 25–29, Oct. 2006. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1717338

[62] 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

[63] 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

[64] D. C. Holley, G. D. Mele, and S. Naidu, "NASA Rat Acoustic Tolerance Test 1994-1995: 8 kHz, 16 kHz, 32 kHz Experiments," San Jose State University, San Jose, CA, USA, Tech. Rep. NASA-CR-202117, Jan. 1996. [Online]. Available: https://ntrs.nasa.gov/api/citations/19960047530/downloads/19960047530.pdf

[65] V. V. Morgun, L. I. Voronin, R. R. Kaspransky, S. L. Pool, M. R. Barratt, and O. L. Novinkov, "The Russian-US Experience with Development Joint Medical Support Procedures for Before and After Long-Duration Space Flights," NASA, Houston, TX, USA, Tech. Rep., 1999. [Online]. Available: https://ntrs.nasa.gov/api/citations/20000085877/downloads/20000085877.pdf

[66] R. B. Howe and R. Johnson, "Research Protocol for the Evaluation of Medical Waiver Requirements for the Use of Lisinopril in USAF Aircrew," Air Force Materiel Command, Brooks Air Force Base, TX, USA, Interim Technical Report AL/AO-TR-1995-0116, Nov. 1995. [Online]. Available: https://apps.dtic.mil/sti/tr/pdf/ADA303379.pdf

[67] D. Liu, S. Tian, Y. Zhang, C. Hu, H. Liu, D. Chen, L. Xu, and J. Yang, "Ultrafine SnPd nanoalloys promise high-efficiency electrocatalysis for ethanol oxidation and oxygen reduction," ACS Applied Energy Materials, vol. 6, no. 3, pp. 1459–1466, Jan. 2023, publisher: ACS Publications. [Online]. Available: https://pubs.acs.org/doi/pdf/10.1021/acsaem.2c03355?casa_token=ItHfKxeQNbsAAAAA:8zEdU5hi2HfHsSony3ku-lbH902jkHpA-JZw8jIeODzUvFtSdQRdbYhmVq47aX22igR52o2S22mnC88Mxw
[68] E. F. Barbosa, E. Y. Nakagawa, and J. C. Maldonado, "Towards the Establishment of an Ontology of Software Testing," vol. 6, San Francisco, CA, USA, Jan. 2006, pp. 522–525.
[69] P. Ammann and J. Offutt, Introduction to Software Testing, 2nd ed. Cambridge, United Kingdom: Cambridge University Press, 2017. [Online]. Available: https://eopcw.com/find/downloadFiles/11
[70] ISO/IEC, "ISO/IEC 25019:2023 - Systems and software engineering –Systems and software Quality Requirements and Evaluation (SQuaRE) –Quality-in-use model," ISO/IEC 25019:2023, Nov. 2023. [Online]. Available: https://www.iso.org/obp/ui/en/#iso:std:iso-iec:25019:ed-1:v1:en
[71] L. Baresi and M. Pezzè, "An Introduction to Software Testing," Electronic Notes in Theoretical Computer Science, vol. 148, no. 1, pp. 89–111, Feb. 2006. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1571066106000442
[72] 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
[73] 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.