Uncovering AGENTS.md: What Testing Guidance Open Source Projects Provide to Coding Agents

Baris Ardic
Delft University of Technology
Delft, The Netherlands
b.ardic@tudelft.nl

Mitchell Olsthoorn
Delft University of Technology
Delft, The Netherlands
m.j.g.olsthoorn@tudelft.nl

Andy Zaidman
Delft University of Technology
Delft, The Netherlands
a.e.zaidman@tudelft.nl

***Abstract*—Agentic coding tools increasingly operate inside software repositories, where their behavior is shaped not only by model capabilities but also by repository context. An emerging form to provide such context is agent context files, also known as AI guidance files (e.g., `AGENTS.md` or `CLAUDE.md`), which can encode project commands, conventions, and development expectations. In this paper, we study how open-source projects organize these guidance files and how they communicate testing expectations to coding agents. We analyze repositories from AIDev-pop, a collection of popular GitHub repositories with observed agent-authored pull requests. We first mine root-level AI guidance files and any Markdown files they explicitly reference to characterize repository-level organization and delegation. We find that root-level guidance is concentrated in generic `AGENTS.md` files and Claude-specific files, and that root guidance is not always self-contained: many repositories refer to additional resources. We then manually analyze testing guidance in a stratified sample of 300 repositories, covering 626 Markdown files and more than 3,300 coded quotations. In our sample, testing guidance is common: 249 of the 300 repositories include test commands, helping agents execute repository-specific tests. Less frequent themes on testing guidance are related to strategy, mentality, mocking, coverage, avoidance, and naming convention expectations. Overall, our results show that AI guidance files are currently mostly used to expose testing infrastructure to coding agents, but are less consistently used to make maintainer expectations about good tests explicit.**

***Index Terms*—Software engineering, agentic coding, AI agents, AI guidance, software testing, open-source software**

I. Introduction

Large Language Models (LLMs) have taken software engineering by storm, as software engineering tasks like generating code, making edits, or writing test cases can now be delegated [1]–[3]. We are now witnessing the next step in this evolution: *agentic software engineering*. In agentic software engineering, agents with varying levels of autonomy perform complete software engineering activities, encompassing code generation, static analysis, testing, debugging, and maintenance, either independently or in cooperation with human developers [4], [5]. These agents use large language models (LLMs) to interpret requirements, create solutions, reason about code, and respond to stakeholders in natural language [6]. They may also autonomously participate in pull-based development [7], [8].

As these agents start to operate with increasing autonomy, their behavior not only depends on model capabilities but also on the contextual information present in the repository [9]. Part of that contextual information is encoded in *agent context files*, such as AGENTS.md or CLAUDE.md, that serve as "READMEs for agents" [10]–[14], specifying architecture, build commands, coding conventions, and other constraints [15]. These AGENTS.md files are version-controlled, encode agent guidance that was previously part of prompts, and are collaboratively maintained configuration artifacts [15]; they have been reported to be adopted by more than 60,000 repositories to date [11]. Although recent work has begun to study agent-facing configuration artifacts [16]–[18], we still know little about how AI guidance is organized within repositories and what development practices it emphasizes.

Software testing is one of the core mechanisms to maintain confidence in software correctness and reliability [19]–[32]. Tests help detect regressions, document expected behavior, and support maintainability as systems evolve [33], [34]. In open-source development, testing is also part of the contribution process: projects can ask contributors to add, update, or run existing tests before submitting changes [35]. Because of the importance of tests in the pull-based development model in open source projects [36], this paper investigates how developers of open source software projects encode guidance with regard to software testing into AGENTS.md files.

Our investigation is led by the following research questions:

RQ1 : How are AI guidance files organized in repositories?

Through RQ1, we aim to increase our understanding of how open source projects structure their AI guidance. In particular, we aim to understand whether guidance is structured in a single root-level file or whether separate guidance files exist for different tasks or tools. Additionally, we investigate whether guidance files also point to auxiliary guidance information in other Markdown files.

RQ2 : How do AI guidance files address testing?

Knowing the importance of testing to safeguard correctness and reliability, we examine what AI guidance developers encode in agent context files. In particular, we aim to understand what testing knowledge is transferred to agents and what expectations are set for AI agents when it comes to testing.

In this study, we investigate repository-level Markdown files intended to guide coding agents. We focus on AIDev-pop, a subset of popular GitHub repositories with observed agent-

authored pull requests [37], [38]. From these repositories, we identify AI guidance files such as `AGENTS.md`, `CLAUDE.md`, and `GEMINI.md`.

Our study contributes an empirical view of repository-level guidance for coding agents. First, we characterize how AI guidance files are organized in popular repositories with agentic pull-request activity. Second, we examine how root guidance files refer to additional Markdown files in these repositories. Third, we qualitatively analyze testing guidance provided to agents in a stratified sample of repositories and identify recurring themes. Our findings show that root-level AI guidance is concentrated in generic and Claude-specific files, but is not always limited to a single self-contained artifact: some root guidance files expose additional project documentation and AI-facing instructions elsewhere in the repository. For testing, guidance most often helps agents run and navigate tests, while guidance about test design, mocking, coverage, avoidance, and naming appears less consistently. This suggests that repositories commonly help agents run tests, but less often communicate what constitutes a good test.

II. Related Work

Recent work has begun to examine repository-level guidance and configuration artifacts for agentic coding tools. Mohsenimofidi et al. [15] study context engineering for AI agents in open-source software and analyze both the presentation and evolution of repository context files. They report that instructions vary along descriptive, prescriptive, prohibitive, explanatory, and conditional styles, and that changes to these files often fine-tune or adjust existing instructions rather than following a single clear evolution pattern.

Chatlatanagulchai et al. [39] provide a Claude-specific view, showing that `CLAUDE.md` files commonly contain practical repository information, including commands, implementation notes, and architectural context. Chatlatanagulchai et al. [40] also performed another study to characterize 2,303 agent context files from 1,925 repositories. Their content analysis leads to 16 instruction types, and their findings indicate that developers prioritize functional context, such as build and run commands, implementation details, and architecture, over non-functional requirements like security and performance.

Galster et al. [16] study configuration mechanisms across tools such as Claude Code, GitHub Copilot, Cursor, Gemini, and Codex. They show that context files are a prominent configuration mechanism, while more advanced mechanisms, such as skills and subagents, are adopted more selectively. A related dataset paper broadens this view by collecting configuration artifacts across tools and repositories [18].

Gloaguen et al. [17] examine the effect of `AGENTS.md` files on agent performance, showing that repository context files can change agent behavior but do not necessarily improve task success.

Treude et al. [41] found that software engineers are embedding guidance related to fairness, accessibility, sustainability, tone, and privacy into repository context files.

Together, these studies establish agent-facing guidance files as an important part of the emerging agentic-development toolchain. Our study extends this body of work by examining how (1) guidance files are organized at the repository level and (2) what testing-related instructions they communicate to coding agents.

Testing guidance has been studied more directly in human-facing documentation. Falcucci et al. [35] analyze what contribution guidelines say about software testing and show that testing instructions are common in contribution documentation, often in the form of commands for running tests. This provides a point of comparison for agent-facing guidance: repositories may communicate similar testing expectations to coding agents, but through different files, structures, and conventions.

III. Methodology

To answer our research questions, we combine repository mining with manual qualitative content analysis. We first identify repositories with root-level AI guidance files and construct a root guidance corpus. We then extend this corpus by retrieving selected files explicitly referred to from these root files. Finally, we manually inspect a balanced subset of repositories to characterize the themes of testing guidance that cannot be captured reliably through keyword matching alone.

A. Repository Selection

We start from AIDev-pop, the popular-repository subset of the AIDev dataset [37]. AIDev is a dataset of GitHub pull requests created by autonomous coding agents, including agents such as Claude Code, GitHub Copilot, OpenAI Codex, Cursor, and Devin. AIDev-pop restricts this dataset to pull requests from repositories with more than 100 GitHub stars. These repositories provide a relevant setting for studying AI guidance files because they are active open-source projects in which agentic development activity has already been observed.

We use AIDev-pop to identify which repositories to study, but we retrieve repository contents directly from GitHub. This allows us to analyze the repository state available at the time of our GitHub scan, conducted on April 1, 2026, rather than relying only on the repository state represented in the original AIDev data. This is important because AI guidance files may evolve rapidly as agent-driven software development continues to mature.

To obtain the repository list, we extract repository identifiers from the pull request metadata. In AIDev, repositories are represented as GitHub API URLs, which we normalize into the standard `owner/repository` format. For each repository, we query the repository root through the GitHub API and record whether the repository was successfully accessed, whether matching AI guidance files were found, or whether the scan failed due to access errors or other API-level issues.

AIDev-pop lists 2,807 unique repositories. Of these, 21 repositories were no longer accessible at the time of our scan, for example, because they had been moved, removed, or otherwise could not be queried through the GitHub API.

B. Identifying Root-Level AI Guidance Files

For each accessible repository, we inspect the repository root for files whose names indicate that they provide instructions to AI coding agents. We focus on root-level files because they are the most likely to define repository-wide guidance and because they can be identified consistently across repositories.

We identify candidate AI guidance files using a predefined filename list. The list includes generic instruction files, such as `AGENTS.md`, `agent.md`, `AI.md`, `LLM.md`, `instructions.md`, and `ai-instructions.md`; agent-specific files, such as `CLAUDE.md`, `claude-code.md`, `gemini.md`, `codex.md`, and `openai.md`; and files associated with coding assistants or AI development tools, including `copilot.md`, `copilot-instructions.md`, `github-copilot.md`, `cursor.md`, `devin.md`, `codeium.md`, and `windsurf.md`. Filename matching is case-insensitive, while preserving the original filename and path for later analysis.

We grouped the files into file families based on their filenames. Generic instruction files are assigned to the *Generic* family. Frequently occurring files explicitly associated with Claude or Gemini are assigned to the *Claude* and *Gemini* families, respectively. Remaining tool-specific and other less frequent guidance filenames are grouped into an *Other* family. This grouping allows us to distinguish general repository guidance from guidance written for a particular agent or tool, while avoiding overly sparse categories for rare filenames.

For each matched root-level file, we retrieve its content and metadata through the GitHub API. We store the repository name, repository-relative path, filename, file family, file content, and basic file-level metadata such as file size and line count. These retrieved files constitute the root guidance corpus. We use this to investigate root-level organization, including which file families and filenames are most common and whether repositories rely on a single guidance file or multiple root-level guidance files.

C. Referred File Retrieval and Extended Corpus Construction

Root-level AI guidance files may not contain all relevant instructions directly. During preliminary inspection, we observed that some root files refer to additional repository documentation through explicit links or path mentions. To capture such delegated guidance, we construct an extended guidance corpus by retrieving selected files that are explicitly referenced from the root guidance files.

We start from the root guidance corpus, consisting of repositories with at least one root-level AI guidance file (e.g., `AGENTS.md`, `CLAUDE.md`). For each root guidance file, we extract candidate references using two mechanisms: (i) standard Markdown links and (ii) plain-text path mentions. We restrict extraction to local repository paths and ignore external resources. To avoid uncontrolled expansion, we retain only Markdown files (`.md`) whose filenames match a predefined allowlist of guidance-oriented names. The allowlist combines two types of files: recognized AI guidance filenames from the root-file detector, which allows root files to delegate to additional or localized agent-facing instructions such as `AGENTS.md`, `CLAUDE.md`, and `SKILL.md` files used by agent skill packages [42]; and established contribution or testing-oriented documentation names that are directly relevant to our study scope, such as `CONTRIBUTING.md`, `TESTING.md`, and `TESTS.md`.

We perform a single expansion step, i.e., one hop from the root guidance file, and do not recursively follow references from retrieved files. This choice reflects a conservative interpretation of delegation: we include only files that are explicitly surfaced by the root guidance file itself. Retrieved files are stored with their repository-relative paths, enabling us to analyze their structural placement, such as whether they appear at the repository root, in documentation directories, in testing-related directories, or in component-specific subdirectories.

We distinguish between repository-available information and explicitly exposed guidance. Documentation that exists elsewhere in the repository is not included unless it is directly referenced from a root guidance file and falls within the filename scope described above. This avoids treating every linked Markdown file, such as general design notes, examples, or unrelated documentation, as part of the guidance corpus. Instead, the extended corpus is intended to capture a conservative set of explicitly exposed files that are likely to contain actionable agent-facing guidance about contribution or testing.

Together with the root guidance files, the successfully retrieved referred files form the extended guidance corpus.

D. Structural Classification of Guidance Files

To answer RQ1, we characterize how AI guidance files are organized at both the repository and file levels. For the root guidance corpus, we analyze the distribution of matched filenames and file families, and we count how many root-level guidance files appear in each repository. We also classify repositories according to whether their root guidance is generic only, tool-specific only, or mixed, i.e., containing both generic and tool-specific guidance files.

To further interpret multi-file organization, we also compare collected guidance files within the same repository for textual overlap. We use normalized exact hashes to identify duplicate files and character n-gram TF–IDF cosine similarity to identify near-duplicate or highly similar files. TF–IDF cosine similarity is a standard lexical document-similarity measure [43], and character n-gram features have been used to detect duplicate software-engineering text artifacts such as bug reports [44]. This gives us a transparent and reproducible way to detect substantial textual overlap among guidance files within the same repository.

For the extended guidance corpus, we additionally analyze file locations using repository-relative paths to characterize their structural placement. We compute each file's path depth and assign it to a path bucket. Root-level files are assigned to *root*; files under documentation directories are assigned to *documentation*; files under test-related directories are assigned to *testing*; files under source or package directories are assigned

to *source/package*; and remaining nested files are assigned to *other nested.*

Finally, we use file location as a proxy for the apparent scope of the guidance. Root-level guidance files are classified as *repo-wide root*; files in broad documentation or project-level directories are classified as *broad area*; and files located under source, package, tool-specific, or otherwise nested component directories are classified as *component- or tool-scoped*. These classifications allow us to distinguish centralized repository-wide guidance from guidance that appears to be delegated to more specific parts of the repository.

E. Detecting Testing-Related Mentions

To identify files that explicitly discuss testing, we apply a keyword-based analysis to both the root guidance corpus and the extended guidance corpus. We search each file for whole-word occurrences of the term `test` and its common forms `tests`, `testing`, and `tested`. Matching is case-insensitive, and whole-word matching is used to avoid partial matches in unrelated words.

For each file, we record whether at least one testing-related term appears, which terms were matched, and the total number of matched occurrences. We use these to characterize explicit testing mentions and to stratify the qualitative analysis sample. The keyword analysis is not used to infer the type or extent of testing guidance; those claims are based on the qualitative coding of the sampled repositories.

F. Qualitative Analysis Sample and Coding Procedure

The aforementioned keyword-based analysis identifies files that explicitly mention testing, but it does not help us understand what kind of testing guidance those files provide. To characterize this content, we perform a manual qualitative analysis on a sample of 300 repositories, considering both the root guidance and referred files.

We select repositories using a stratified sampling strategy based on the prevalence of testing-related keywords in their root and referred guidance files. Specifically, we first compute each repository's total number of testing-related keyword occurrences across all retrieved guidance files. Repositories with no testing-related mentions are retained for full-corpus reporting but excluded from the qualitative analysis sample. The remaining repositories are split into low-, medium-, and high-testing buckets based on their repository-level testing-related keyword counts. This produced thresholds of 11 and 28 testing-related occurrences: repositories with 1–11 occurrences are assigned to the low-testing bucket, repositories with 12–28 occurrences to the medium-testing bucket, and repositories with 29 or more occurrences to the high-testing bucket. We randomly sample 100 repositories from each bucket, resulting in a balanced qualitative analysis sample of 300 repositories.

We organize the review at the repository level rather than treating each file independently, to preserve the local context in which testing instructions appear. The qualitative analysis sample is used to identify and characterize the themes of testing guidance that appear in AI guidance files. We do not use this sample to generalize how common these themes are across all repositories in AIDev-pop.

We analyze the sampled files using an iterative coding process. We first apply open coding [45] to identify recurring testing-related instructions, such as instructions to run tests, add or update tests, follow project-specific test commands, ensure CI or validation checks pass, consult testing documentation, or follow test style conventions. We then use axial coding [45] to group related codes into broader themes. Codes and themes are reviewed and refined through repeated passes over the sampled files.

To assess coding consistency, two coders independently coded a random subset of 30 repositories, corresponding to 10% of the 300-repository qualitative analysis sample. This subset preserves the low-, medium-, and high-testing strata. One coder coded the full sample, while the second coder coded only the 30-repository subset. Following recommended practice for reporting intercoder agreement in qualitative coding, we assessed agreement over section-level code applications [46], [47]. Disagreements were recorded when coders assigned different codes to the same testing-related content or selected different quoted sections for a coding decision. The double-coded subset contained 331 code applications. The coders initially disagreed on 77 applications, yielding an initial agreement of 76.8%. They then discussed all disagreements in a consensus meeting, refined their shared interpretation of the codebook, and reached agreement on the subset [46]. The first coder subsequently revisited the remaining repositories and updated the qualitative coding accordingly.

G. Analysis Overview

Table I summarizes how the three corpora are used to answer our research questions. We use the root guidance corpus for repository-level prevalence and root-level organization analyses. We use the extended guidance corpus to analyze delegated files and the structural placement of guidance beyond the repository root. For testing-related content, we use keyword analysis to identify explicit testing mentions and construct the qualitative analysis sample. We then use the qualitative analysis sample to characterize themes of testing guidance and to report their prevalence within the sampled repositories. We provide a data package containing the frozen guidance corpus, derived metadata, analysis scripts, similarity and symlink diagnostics, and qualitative analysis material used in this study [48].

IV. RQ1: How are AI guidance files organized in repositories?

We answer RQ1 using the two corpora constructed in the methodology. The root guidance corpus, consisting of 975 accessible AIDev-pop repositories with at least one recognized root-level AI guidance file, supports our analysis of root-level organization: filename and file-family distributions, single- versus multi-file setups, and generic, tool-specific, or mixed root patterns. The extended guidance corpus adds Markdown files explicitly referred to from root guidance files, allowing

TABLE I
OVERVIEW OF THE DATA USED IN THE ANALYSIS.

Corpus	Used for	RQ
Root guidance corpus	Prevalence, filename and file-family distributions, root-level organization, and within-repository similarity	RQ1
Extended guidance corpus	Referred files, structural placement beyond root files, and testing-related keyword analysis	RQ1, RQ2
Qualitative analysis sample	Qualitative coding of themes of testing guidance	RQ2

us to examine whether AI guidance remains centralized at the repository root or is delegated to existing project documentation, agent-specific files, or more localized project areas.

Finding 1

Root-level AI guidance files are present in a substantial minority of popular repositories with observed agentic pull-request activity: 975 of 2,786 accessible repositories contained at least one such file.

A. Root-Level Filenames and File Families

We first examine which root-level filenames repositories use for AI guidance. The root guidance corpus contains 975 repositories and 1,325 root-level guidance files. The distribution is concentrated around two filenames: `AGENTS.md` appears in 666 files when capitalization variants are grouped, and `CLAUDE.md` appears in 615 files. Other guidance filenames are much less common: `GEMINI.md` appears 25 times, while the remaining recognized filenames appear only a few times.

Table II summarizes the root-level guidance files by family. Grouped by family, generic guidance files account for 683 files (51.5%). This family consists primarily of `AGENTS.md` files (666), with a small number of other generic filenames: `AGENT.md` (11), `LLM.md` (3), `AI.md` (1), `AI-INSTRUCTIONS.md` (1), and `INSTRUCTIONS.md` (1). Claude-specific files account for 615 files (46.4%), Gemini-specific files account for 25 files (1.9%), and other tool-specific files account for 2 files (0.2%). Thus, root-level AI guidance in our corpus is dominated by generic agent guidance and Claude-specific guidance.

B. Repository-Level Root Guidance Patterns

Most repositories use only one or two root-level guidance files. Among the 975 repositories, 645 repositories contain one root-level guidance file, 312 contain two, 16 contain three, and 2 contain four. This suggests that root-level AI guidance is usually organized as a small set of entry-point files rather than as many parallel files.

Next, we examine how repositories combine different guidance files at the root level. Table III summarizes these repository-level patterns. Repositories are *generic-only* when they contain only generic guidance files, *tool-specific-only* when they contain only Claude, Gemini, or other tool-specific files, and *mixed* when they contain both generic and tool-specific guidance. The three patterns are all common: 353 repositories are generic-only, 295 are tool-specific-only, and 327 are mixed.

To interpret these multi-file repositories, we also checked whether multiple guidance files within the same repository contained distinct text or duplicated instructions. We compared all collected Markdown files within each repository using normalized exact hashes and character n-gram TF–IDF cosine similarity. The analysis considered 3,400 within-repository file pairs across 1,787 usable files, excluding files with fewer than 50 normalized characters. We treated files with identical normalized hashes as exact duplicates. For non-identical pairs, we used cosine similarity as an operational measure of textual overlap. Since clone and near-duplicate detection depend on the selected representation and comparison procedure [49], we use thresholds to define analysis categories for this corpus. We labeled pairs with similarity of at least 0.95 as near duplicates, and pairs between 0.85 and 0.95 as high-similarity candidates for interpreting whether multiple guidance files provide distinct instructions.

High-similarity pairs were concentrated almost entirely among root guidance files. Among the 331 repositories with at least two root guidance files, 143 repositories (43.2%) contained at least one root–root pair with similarity of 0.85 or higher. Using the stricter exact-or-near-duplicate criterion, 136 repositories (41.1%) contained at least one such root–root pair. Most exact duplicates were `AGENTS.md`-`CLAUDE.md` pairs: 118 exact duplicate pairs appeared between these two filenames.

This suggests that some repositories expose similar or identical instructions through multiple root filenames, rather than maintaining fully distinct guidance artifacts for each file. We therefore checked whether exact duplicates represented separate copied files or repository-level aliases implemented as symbolic links. Using GitHub metadata, we identified 161 symlinked guidance files across 141 repositories, of which 157 were root guidance files. These symlinks were concentrated in root-level agent filenames, especially `CLAUDE.md`. In these cases, the symlinked file is not a separate guidance artifact: it is a path within the same repository that points to another guidance file, often exposing the same underlying instructions under an additional agent-specific filename. Of the 161 symlinked guidance files, 121 were named `CLAUDE.md`, compared with 28 named `AGENTS.md` and 6 named `GEMINI.md`. Among the 139 exact-duplicate pairs, 124 involved at least one symlinked path, and 106 were directly explained by one file linking to the other. Thus, most exact duplicates reflect aliases

TABLE II
ROOT-LEVEL AI GUIDANCE FILES BY FAMILY.

Family	Files	Share
Generic	683	51.5%
Claude	615	46.4%
Gemini	25	1.9%
Other tool-specific	2	0.2%

to the same underlying file rather than separate copies of the same guidance text.

Finding 2

Root-level AI guidance is concentrated in generic and Claude-specific files. Repositories commonly use generic-only, tool-specific-only, and mixed root patterns. However, multi-file organization does not always indicate distinct guidance artifacts: most exact duplicates are symlink-based aliases, while other cases reflect copied or highly similar guidance across separate files.

C. Referred Files Extend Root Guidance

Root AI guidance files are not always self-contained. In addition to providing instructions directly, they may point agents to other Markdown files in the repository. Following our methodology, we consider these files as *referred files*: Markdown files collected because they were linked or path-mentioned from root AI guidance files. This definition is independent of path depth. For example, a root-level `CONTRIBUTING.md` file is a referred file when it is referenced from a root guidance file.

Across the corpus, we identify 533 referred Markdown files in 200 repositories. Of these, 103 files are root-level referred files and 430 are located below the repository root. Thus, references from AI guidance files do not only lead to nested, agent-specific instructions. They also connect agents to existing project-level documentation.

To characterize where referred files appear, we group them by their repository-relative paths. Root-level files are those with no directory prefix. Documentation paths include folders such as `docs/` or `documentation/`; testing paths include folders such as `test/`, `tests/`; source or package paths include folders such as `src/`, `lib/`, `packages/`, or `apps/`; agent/tooling paths include folders such as `.claude/`, `.cursor/`, or `.agents/`; and GitHub/tooling paths include folders such as `.github/` or `.devcontainer/`. Remaining hidden or nested paths are grouped separately.

Table IV shows the resulting location categories. The largest group appears in source or package paths, with 176 files across 41 repositories. Root-level referred files form the second largest group, with 103 files across 103 repositories. Other nested paths account for 100 files across 42 repositories, while agent- or tooling-specific folders account for 72 files across 32 repositories. Documentation and testing folders are less frequent, but they still appear as explicit targets of references from root AI guidance.

The filenames of referred files show two recurring uses of these references. First, root guidance often points to established project documentation. When capitalization variants are grouped, `CONTRIBUTING.md` appears 117 times across 115 repositories, with the difference coming from two repositories that include both root-level and documentation-subdirectory `CONTRIBUTING.md` files. Similarly, `TESTING.md` appears 20 times across 20 repositories. Second, root guidance can lead to additional AI-facing instruction files. The most common referred filename is non-root `AGENTS.md`, with 239 files across 58 repositories, followed by `SKILL.md` with 109 files across 30 repositories and `CLAUDE.md` with 46 files across 19 repositories. The path-depth breakdown suggests that referred files are usually close to the repository root. The largest group appears at depth 2, with 168 files, followed by depth 1 with 132 files and depth 3 with 128 files. Root-level referred files account for 103 files. Only three files appear deeper than depth 3. Therefore, references generally add a small number of organizational layers rather than scattering agent-relevant context arbitrarily deep in the repository.

Using the location-based scope proxy defined in the methodology, referred files appear in both repository-level and more localized positions. The clearest repository-level group consists of the 103 root-level referred files. Beyond the root, referred files most often appear in source or package paths (176 files), other nested paths (100 files), and agent/tooling folders (72 files), with smaller numbers in documentation, testing, GitHub/tooling, and other hidden folders. We interpret this classification as an approximation of apparent scope based on structural location, rather than as a manually validated classification of each file's intended scope.

At the repository level, referred-file usage is skewed. The median repository with referred files contains one referred file, while the mean is 2.65 and the maximum is 37. This indicates that most repositories use references sparingly, often to point to a single project document or instruction file, while a smaller number of projects use references more extensively across packages, tooling areas, or documentation structures.

Overall, referred files show that root AI guidance can act as an entry point into a broader documentation structure. In some repositories, guidance remains a single root-level artifact; in others, it connects agents to existing human-facing documents and to additional AI-facing files placed elsewhere in the repository. For RQ1, this indicates that the organization of AI guidance is not limited to root files, but can extend through references to related Markdown files.

TABLE III
REPOSITORY-LEVEL ROOT GUIDANCE PATTERNS.

Pattern	Repositories	Share
Generic only	353	36.2%
Mixed	327	33.5%
Tool-specific only	295	30.3%

TABLE IV
LOCATIONS OF REFERRED MARKDOWN FILES.

Location category	Files	Repositories
Source or package	176	41
Root-level referred file	103	103
Other nested path	100	42
Agent/tooling folder	72	32
Documentation folder	36	32
Testing folder	22	20
GitHub/tooling folder	16	14
Other hidden folder	9	3

TABLE V
THEMES OF TESTING GUIDANCE IDENTIFIED

Theme	# of Repositories
Test commands	249
Test locations and placement	186
Test strategy	156
Test mentality	144
Test framework&environment	142
Testing tips	109
Testing types	74
Test coverage	71
Avoidance guidance	57
Mocking guidance	52
Test-related examples	39
Test naming and style	38

Finding 3

Root AI guidance is not always a single self-contained artifact. In 200 repositories, root guidance referred to additional Markdown files, combining links to existing project documentation with additional AI-facing instructions placed elsewhere in the repository.

V. RQ2: How do AI guidance files address testing?

Using the qualitative analysis sample, we characterize how AI guidance files address testing. Across the 300 reviewed repositories, we coded more than 3,300 quotations and developed the coding through several refinement passes. The resulting codebook contains more than 200 testing-related codes, capturing recurring instructions about tests, commands, tools, files, and expected agent behavior. We consolidated these codes into 12 recurring themes of testing guidance through axial coding. Repositories typically contained multiple themes of testing guidance: the median repository contained four themes (mean=4.39, SD=2.52), and 257 repositories contained at least two. Prominent themes include directly actionable test commands, guidance on how tests should be designed, where tests are located, which testing frameworks are used, and what testing behavior agents are expected to follow.

A. Overview of Themes of Testing Guidance

Table V summarizes these themes at the repository level, counting a theme once per repository if it appears in any reviewed guidance file for that repository. In the following paragraphs, we discuss the 12 themes that emerged from our thematic analysis.

Test commands. Test commands are the most common theme of testing guidance: they appear in 249 of the 300 reviewed repositories (83.0%). This category captures concrete commands for running tests.

Common instances include commands for running the repository's test suite or the default set of tests (230 repositories), commands for running a narrower target such as a package, file, or selected test case (97), unit test commands (46), end-to-end test commands (40), commands that run tests with coverage (37), and integration test commands (33). This guidance is immediately actionable because it tells the agent how to invoke the repository's test infrastructure.

Test locations and placement. Test locations and placement form the second most prevalent theme, appearing in 186 repositories (62.0%). This guidance helps agents understand where testing artifacts are located or where new artifacts should be placed. We distinguish between test locations and placement. Test locations describe where tests or testing resources already exist in the repository, while placement instructions tell the agent where new tests or related artifacts should be added. In practice, this included descriptions of existing test locations (141), placement instructions for new artifacts (69), and references to supporting testing resources such as test fixtures (23) and shared test helpers (9). This distinction separates repository orientation from direct development guidance.

Test strategy. Another common theme is test strategy, which appears in 156 repositories (52.0%). Test strategy captures instructions that describe a coherent approach for designing, selecting, organizing, or executing tests. We use this category when the guidance describes how testing should be approached. These instructions go beyond providing a single command or isolated recommendation. Common instances include broad strategy guidance (138), execution strategies (21), instructions to check existing tests for usage patterns (15), failed-test strategies (11), and test patterns (9). One example of broad strategy guidance comes from a repository that advises agents to focus tests on application behavior and accessibility, test the exposed API through its inputs and outputs rather than implementation details, group tests with a single `describe` block per subject, assert on specific errors, and avoid exporting internal helpers purely for test coverage.

Test mentality. Nearly half of the reviewed repositories include test mentality guidance: 144 repositories (48.0%) contain this theme. Test mentality captures general expectations, habits, or values about testing that agents are expected to carry across tasks. These instructions express what agents should normally or always do during development, framing behavioral expectations. Examples include ensuring all tests pass (27), verifying tests pass before a pull request (15), always running tests (10), including regression tests for bug fixes (10), and creating tests for new logic (8).

Test framework and environment. Guidance about test frameworks and environments appears in 142 repositories (47.3%). This category describes the testing infrastructure of the repository. The most visible examples are testing frameworks used (112), CI/CD configuration (27), test environment information (14), test-suite configuration scripts (10), and environment variables needed for testing (7). We distinguish this theme from test commands as it provides contextual knowledge about the infrastructure needed to run or understand tests, rather than giving an invocation command.

Testing tips. Testing tips are less widespread than strategies, but still appear in 109 repositories (36.3%). These tips capture shorter or more localized pieces of testing advice. They are useful for testing, but they do not amount to a complete strat-

egy. The more specific tips include repository-specific testing tips, such as concurrent testing tips (13), advice about common testing problems (5), and guidance for debugging test failures (2). For example, one repository warns that all tests potentially run in parallel and instructs agents to design tests accordingly. Another provides practical guidance on failure diagnosis, such as using a verification test when failures are caused by a dummy server not running or missing environment variables. We distinguish testing tips from test strategy because tips usually appear as isolated recommendations, while strategies describe a more complete approach to testing.

Testing types. Testing type guidance appears in about a quarter of the reviewed repositories, with 74 repositories (24.7%) mentioning this theme. It identifies the kinds or levels of testing that are relevant in a repository. End-to-end testing was the most common specific testing type (33), followed by database testing (9), integration testing (9), performance testing (8), and snapshot testing (5). These help capture which testing activities are made visible to agents and whether the guidance concerns testing in general or a specific testing mode.

Test coverage. Test coverage appears in 71 of the 300 reviewed repositories (23.7%). This guidance concerns how repositories refer to test coverage as part of testing or validation. Coverage may appear as a command, a quality target, or an expectation. For example, guidance files instruct agents to run tests with coverage or check coverage after changes.

Avoidance guidance. Avoidance guidance appears in 57 repositories (19.0%). This guidance is expressed through prohibitions or warnings about testing practices. The most common examples are general avoidance guidance (43), avoiding mocks (13), and not simulating test results (1). This theme of guidance constrains agent behavior by identifying testing shortcuts or practices that the repository discourages.

Mocking guidance. Mocking guidance appears in 52 repositories (17.3%). This group captures instructions about whether, when, where, or how agents should use mocks and test doubles. It includes guidance about mocking dependencies or external services, avoiding or reducing mocks, mock locations, and mock examples. Because of this, mocking can appear as technical guidance, placement guidance, avoidance guidance, or example guidance.

Test-related examples. Concrete test-related examples appear in 39 repositories (13.0%). These examples show agents what repository-conforming tests look like, rather than only describing them in prose. Most examples were coded as example tests (31), with smaller numbers showing example test structures (10) or mock examples (2). We consider examples as a theme of testing guidance because they make testing expectations concrete for agents.

Test naming and style. Test naming and style guidance is less common, appearing in 38 repositories (12.7%). This guidance concerns how tests should be named, structured, or written. This category is mostly driven by test naming guidance (35), with a smaller number of files describing broader test style expectations. This guidance helps agents produce tests that conform to repository conventions rather than merely producing tests that execute.

Finding 4

Our thematic analysis revealed 12 themes of testing guidance. *Test commands* are the backbone of testing guidance. They appear in 249 of the 300 reviewed repositories (83.0%), making them the most common theme of testing guidance.

B. A Closer Look at Testing Guidance

We take a closer look at the explicit content that we encountered for specific themes in the repositories. Among the 249 repositories with test-command guidance, only 19 repositories (7.6%) contained commands as their only theme. The most common combinations paired commands with locations or placement (167 repositories), framework or environment guidance (134), strategy (134), and test mentality (124). This shows that commands often coexist with other testing guidance, but the most common combinations remain centered on making testing executable, navigable, and actionable.

With regard to test commands, we observe that they are present in AI guidance files in 83% of the repositories. This raises the question of what happens in the other 17% of the repositories. Our analysis reveals four potential situations:

1) running the full test suite may be outside the expected scope of the agent; the agent may only be expected to create changes and submit a pull request
2) it might be that the relevant tests are executed through continuous integration
3) 6 repositories explicitly mention that they only do manual testing
4) the agent might also find test commands in packaging files, CI configurations, or other documentation

Going beyond test commands, if the guidance does not explain what kind of tests maintainers prefer, when mocks are acceptable, whether tests should exercise real behavior, or how coverage should be interpreted, the agent does not have an immediate source of information to rely on. This matters because AI agents might still generate tests, but they will not adhere to the repository's (implicit) testing norms. However, 15 repositories explicitly instruct the agent to check existing tests for determining a testing strategy.

An interesting example of why explicit guidance is important comes from recent work in the area of mocking. Hora and Robbes found that agent-authored test commits are more likely to add mocks than non-agent test commits, and argue that this may reflect tests that are easier to generate automatically but less effective at validating real interactions [50]. They therefore call for mocking best practices and anti-patterns to be included in agent configuration files.

Our results show that some repositories already encode such boundaries, but that this is not yet common. Mocking guidance appears in 52 of the 300 reviewed repositories (17.3%). Several instructions do not merely tell agents which mocking library to use; they express testing principles about

realism and functional validation, such as preferring tests of real behavior or emphasizing that coverage is not itself the goal of testing. These instructions frame mocks as a design decision that affects the evidence provided by a test, rather than as a neutral implementation convenience.

Coverage guidance points in a similar direction. Test coverage appears in 71 repositories (23.7%), but repositories do not always frame coverage as a target to maximize. Some guidance instead warns agents against treating coverage as the goal of testing, emphasizing that tests should validate functionality or behavior. This suggests that maintainers are sometimes also trying to prevent agents from optimizing for superficial indicators of test quality.

Taken together, these cases suggest that root AI guidance files can function as coordination points: maintainers can keep repository-wide expectations discoverable at the root while delegating more specific instructions closer to the relevant project context. For researchers, analyzing only root files may therefore underrepresent the guidance exposed to agents.

Finding 5

Commands, locations, and framework information help agents run or navigate tests; they are the most frequent themes of testing guidance. Less frequent themes are related to strategy, mentality, avoidance, mocking, coverage, and naming conventions; they express what good tests should look like.

VI. Discussion

A. From Root Guidance Files to Guidance Structures

The RQ1 results suggest that root-level AI guidance should not be interpreted only as a count of visible files. Although guidance is concentrated in `AGENTS.md` and `CLAUDE.md`, multiple root files do not always represent distinct artifacts: some repositories expose copied, highly similar, or symlinked guidance through different filenames.

The referred-file results extend this point. Root guidance is not always a complete manual, but can serve as an entry point into existing documentation and additional AI-facing guidance files elsewhere in the repository. This matters for both analysis and practice: relevant instructions may be distributed across root files, human-facing documents such as `CONTRIBUTING.md` or `TESTING.md`, and local instruction files placed in package, source, or tooling directories.

This distributed structure also raises a maintainability issue. Two repositories explicitly instruct agents to keep `AGENTS.md` small, and the distributed structure could be an answer to managing more guidance information. Across the manual sample, 94 repositories contained guidance that points agents to other files or resources, including `TESTING.md` files (26 repositories), subfolder `agent.md` files (17), other Markdown files (14), test-related `SKILL.md` files (11), `CONTRIBUTING.md` files (10), documented architecture (2), and multiple skill or Markdown files (2). Six repositories also included guidance about when to use a `SKILL.md` file.

Taken together, these cases suggest that root AI guidance files can function as coordination points: they keep guidance discoverable at the repository root while delegating more specific or localized instructions to documents maintained closer to the relevant project context.

B. Testing Guidance for Agents and Human Contributors

The RQ2 results show a strong operational emphasis: AI guidance commonly helps agents run and navigate tests, while test expectations are less consistently externalized. This mirrors Falcucci et al.'s findings for human-facing contribution guidelines, where test documentation often explains how to run tests, but less often explains how to write tests or discusses coverage, mocking, and best practices [35].

For agents, this imbalance may be more consequential. Human contributors can learn implicit testing norms through experience, discussion, existing tests, and review feedback. Coding agents rely more directly on written repository context. If expectations about mocks, coverage, regression tests, naming, or testing shortcuts are missing, agents may still produce executable tests, but those tests may not match what maintainers consider good project-specific tests.

A potential implication of missing project-specific expectations in AI guidance files is that, given the speed of agent-assisted development, agents may generate a large number of tests that do not fully meet project maintainers' expectations. This potentially increases the risk of tests with weak assertions, excessive mocking, superficial coverage, or other testing-related technical debt.

This does not mean that agent-facing documentation must replace human-facing documentation. Pointers to `TESTING.md` or `CONTRIBUTING.md` can be a useful way to reuse existing guidance. However, reuse may need adaptation when those documents assume tacit knowledge or human interpretation. Maintainers can use agent-facing guidance to make project-specific expectations explicit: which tests to run for different changes, when to add tests, how to use existing tests as models, when mocks are appropriate, and how to interpret coverage or failures.

C. Maintaining AI Guidance

A few rare cases indicate that AI guidance itself is becoming something repositories must maintain deliberately. Seven repositories instruct agents to update `AGENTS.md` or related files, which may keep guidance current but also lets agents modify the instruction layer that shapes future agent behavior. Five repositories instruct agents to disclose AI involvement, such as through identifiable commit metadata. Eight repositories contain intentional circular pointers from referred guidance back to root guidance, showing that guidance can become self-referential rather than strictly hierarchical. These cases are not common, but they point to an emerging maintenance concern: AI guidance files are not only instructions for agents, but repository artifacts whose scope, evolution, and accountability need to be managed.

VII. Threats to Validity

A. Internal Validity

Subjectivity of manual analysis. The qualitative coding of testing guidance involves interpretive decisions about relevant text, code assignment, and theme construction [46], [51]. To reduce this threat, two coders independently coded a stratified subset of 30 repositories, corresponding to 10% of the qualitative analysis sample. They compared 331 code applications, discussed disagreements, refined their shared interpretation of the codebook, and reached agreement on the subset. The first coder then revisited the remaining repositories and updated the coding accordingly.

Keyword-based testing detection. Our keyword search may miss testing guidance that uses other terms, such as validation, or checks, and may also include occasional irrelevant uses of `test`. In this study, keyword counts are used mainly to stratify repositories for the analysis. The qualitative characterization itself is based on manual inspection. Because repositories without testing-related keyword matches were excluded from the qualitative analysis sample, the RQ2 qualitative findings describe repositories with explicit testing-related guidance.

Scope of AI guidance files. We identify root-level guidance using a predefined set of Markdown filenames associated with generic or tool-specific agent instructions. Repositories may encode agent guidance through other filenames, non-Markdown files, or tool settings outside our allowlist. This is an intentional scope decision: our goal is to study prevalent repository-level Markdown guidance conventions, not all possible ways of configuring AI coding agents. Future work could take a more holistic approach and consider other types of AI guidance.

Boundary of referred-file retrieval. We retrieve referred files only when they are explicitly linked or path-mentioned from root guidance files, and we follow only one expansion step. As a result, we may miss guidance that exists elsewhere in the repository, is not referenced from root guidance, or is reachable only through additional links. This conservative boundary avoids uncontrolled expansion and keeps the extended corpus grounded in guidance that root files explicitly expose to agents.

Operational classifications. Path buckets and apparent scope are based on repository-relative file locations, and therefore approximate the intended scope of guidance rather than manually validating it for every file. Similarly, near-duplicate and high-similarity categories depend on the chosen textual-similarity thresholds. We treat these categories as analytical approximations and base the strongest duplication claims on exact hashes and GitHub symlink metadata.

B. External Validity

Scope of the studied repositories. Our results are based on popular GitHub repositories with observed agent-authored pull requests. The AIDev-pop dataset provides a relevant setting as it contains repositories with known agentic development activity. However, future work should investigate whether smaller, less visible, private, or less active repositories organize AI guidance differently.

Markdown-based guidance. Our results generalize primarily to repository guidance exposed through Markdown files, such as `AGENTS.md`, `CLAUDE.md`, `GEMINI.md`, and referred Markdown documentation. Coding agents may also be configured through prompts, tool settings, workflow files, issue templates, or other non-Markdown mechanisms. Therefore, our results characterize Markdown-based guidance, not the complete configuration environment of coding agents.

Evolving agent-guidance conventions. The agent-guidance ecosystem is evolving rapidly. Filename conventions, tool support, and repository practices around files such as `AGENTS.md` and `CLAUDE.md` may change over time. As such, future work could replicate our investigation to examine how guidance files have evolved.

VIII. Conclusion

Agent-facing guidance files are becoming part of the repository context that shapes how coding agents work. In this paper, we studied how such guidance is organized in popular open-source repositories with observed agent-authored pull requests, and how it communicates testing expectations to agents.

Guidance is concentrated in a small number of root-level AI guidance files, especially generic `AGENTS.md` files and Claude-specific files. At the same time, multiple root files do not always represent distinct guidance artifacts: in many cases, repositories expose the same underlying instructions through duplicate or symlinked files. We also find that root AI guidance files are not always self-contained, as some guidance files point agents toward existing project documentation and additional AI guidance files elsewhere in the repository.

With regard to testing guidance, not all guidance is equal. In particular, test commands are the dominant theme, and they are often accompanied by information about test locations, frameworks, and execution context. This makes testing more actionable for agents. Less frequent themes, however, encode expectations about how tests should be designed, when mocks are acceptable, how coverage should be interpreted, or which testing shortcuts should be avoided. This suggests that many repositories help agents find and run tests, but fewer make explicit what maintainers consider good tests.

These findings point to AI guidance files as a mechanism for shaping agent testing behavior. Future work could examine whether or to what extent explicit testing guidance changes agent behavior in realistic development tasks, for example, by reducing over-mocking, improving alignment with existing test patterns, or discouraging superficial coverage-oriented tests. Another direction is to compare agent-facing guidance with human-facing documentation to understand which expectations are copied, adapted, or omitted when projects write for agents. Finally, longitudinal studies could examine how guidance files evolve as maintainers gain experience with agent-authored contributions.

Acknowledgements

This research is sponsored by the Swiss National Science Foundation (SNSF Grant 200021M 205146), and the Dutch Science Foundation NWO (Grant No. VI.C.182.032).

REFERENCES

[1] A. Roychoudhury, C. Păsăreanu, M. Pradel, and B. Ray, "Agentic AI software engineers: Programming with trust," *Commun. ACM*, vol. 69, no. 5, p. 56–58, Apr. 2026.

[2] A. Roychoudhury, "Agentic AI for software: thoughts from software engineering community," 2025. [Online]. Available: https://arxiv.org/abs/2508.17343

[3] A. Deljouyi, R. Koohestani, M. Izadi, and A. Zaidman, "Leveraging large language models for enhancing the understandability of generated unit tests," in *47th IEEE/ACM International Conference on Software Engineering (ICSE)*. IEEE, 2025, pp. 1449–1461.

[4] N. Otoum and N. Elkhalili, "Methods and techniques of agentic software engineering: A systematic literature review," *IEEE Access*, vol. 14, pp. 7443–7465, 2026.

[5] H. Jin, L. Huang, H. Cai, J. Yan, B. Li, and H. Chen, "From llms to llm-based agents for software engineering: A survey of current, challenges and future," *arXiv preprint arXiv:2408.02479*, 2024.

[6] J. He, C. Treude, and D. Lo, "Llm-based multi-agent systems for software engineering: Literature review, vision, and the road ahead," *ACM Trans. Softw. Eng. Methodol.*, vol. 34, no. 5, May 2025.

[7] H. Li, H. Zhang, and A. E. Hassan, "The rise of AI teammates in software engineering (SE) 3.0: How autonomous coding agents are reshaping software engineering," *arXiv preprint arXiv:2507.15003*, 2025.

[8] M. Watanabe, H. Li, Y. Kashiwa, B. Reid, H. Iida, and A. E. Hassan, "On the use of agentic coding: An empirical study of pull requests on github," *ACM Trans. on Software Engineering and Methodology*, 2025.

[9] J. L. Lulla, S. Mohsenimofidi, M. Galster, J. M. Zhang, S. Baltes, and C. Treude, "On the impact of agents.md files on the efficiency of ai coding agents," 2026. [Online]. Available: https://arxiv.org/abs/2601.20404

[10] R. Robbes, T. Matricon, T. Degueule, A. Hora, and S. Zacchiroli, "Promises, perils, and (timely) heuristics for mining coding agent activity," in *Proceedings of the International Conference on Mining Software Repositories (MSR)*. ACM, 2026, pp. 506–517.

[11] AGENTS.md, "AGENTS.md: A simple, open format for guiding coding agents," https://agents.md/, 2026, accessed: 2026-05-19.

[12] OpenAI, "Custom instructions with AGENTS.md," https://developers.openai.com/codex/guides/agents-md, 2026, accessed: 2026-05-19.

[13] M. Nigh. (2025, Nov.) How to write a great agents.md: Lessons from over 2,500 repositories. GitHub Blog. Updated November 25, 2025. Accessed: 2026-05-19. [Online]. Available: https://github.blog/ai-and-ml/github-copilot/how-to-write-a-great-agents-md-lessons-from-over-2500-repositories/

[14] G. Cardoen, T. Mens, and A. Decan, "Ghagentfiles: A dataset of coding agent file histories in github repositories," in *International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2026.

[15] S. Mohsenimofidi, M. Galster, C. Treude, and S. Baltes, "Context engineering for AI agents in open-source software," in *Proceedings of the International Conference on Mining Software Repositories (MSR)*. ACM, 2026, pp. 194–198.

[16] M. Galster, S. Mohsenimofidi, J. L. Lulla, M. A. Abubakar, C. Treude, and S. Baltes, "Configuring agentic AI coding tools: An exploratory study," in *Proceedings of the 3rd ACM/IEEE International Conference on AI-powered Software (AIware 2026)*, 2026.

[17] T. Gloaguen, N. Mündler, M. Müller, V. Raychev, and M. Vechev, "Evaluating AGENTS.md: Are repository-level context files helpful for coding agents?" *arXiv preprint arXiv:2602.11988*, 2026.

[18] M. Galster, S. Mohsenimofidi, L. Böhme, J. L. Lulla, M. A. Abubakar, C. Treude, and S. Baltes, "A dataset of agentic AI coding tool configurations," in *Proceedings of the 3rd ACM/IEEE International Conference on AI-powered Software (AIware 2026)*, 2026.

[19] Z. Lubsen, A. Zaidman, and M. Pinzger, "Using association rules to study the co-evolution of production test code," in *Proceedings of the Working Conference on Mining Software Repositories (MSR)*. IEEE, 2009, pp. 151–154.

[20] K. E. Haji, C. E. Brandt, and A. Zaidman, "Using github copilot for test generation in python: An empirical study," in *Proceedings of the 5th ACM/IEEE International Conference on Automation of Software Test (AST)*. ACM, 2024, pp. 45–55.

[21] C. E. Brandt, M. Castelluccio, C. Holler, J. Kratzer, A. Zaidman, and A. Bacchelli, "Mind the gap: What working with developers on fuzz tests taught us about coverage gaps," in *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. ACM, 2024, pp. 157–167.

[22] C. E. Brandt and A. Zaidman, "Developer-centric test amplification," *Empir. Softw. Eng.*, vol. 27, no. 4, p. 96, 2022.

[23] P. S. Kochhar, T. F. Bissyandé, D. Lo, and L. Jiang, "An empirical study of adoption of software testing in open source projects," in *2013 13th International Conference on Quality Software*. IEEE, 2013, pp. 103–112.

[24] M. Aniche, C. Treude, and A. Zaidman, "How developers engineer test cases: An observational study," *IEEE Transactions on Software Engineering*, vol. 48, no. 12, pp. 4925–4946, 2021.

[25] B. Ardiç, Q. Le Dilavrec, and A. Zaidman, "How students use generative AI for software testing: An observational study," *Empir. Softw. Eng.*, vol. 31, no. 6, p. 167, 2026.

[26] M. Beller, G. Gousios, A. Panichella, and A. Zaidman, "When, how, and why developers (do not) test in their ides," in *Proceedings of the Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*. ACM, 2015, pp. 179–190.

[27] M. Beller, G. Gousios, A. Panichella, S. Proksch, S. Amann, and A. Zaidman, "Developer testing in the IDE: patterns, beliefs, and behavior," *IEEE Trans. Software Eng.*, vol. 45, no. 3, pp. 261–284, 2019.

[28] M. Swillus, R. Hoda, and A. Zaidman, "On the emergence of testing strategies: A socio-technical grounded theory," *Empir. Softw. Eng.*, vol. 31, no. 5, p. 147, 2026.

[29] M. Swillus and A. Zaidman, "Sentiment overflow in the testing stack: Analyzing software testing posts on stack overflow," *J. Syst. Softw.*, vol. 205, p. 111804, 2023.

[30] A. Khatami and A. Zaidman, "State-of-the-practice in quality assurance in java-based open source software development," *Softw. Pract. Exp.*, vol. 54, no. 8, pp. 1408–1446, 2024.

[31] A. Zaidman, "An inconvenient truth in software engineering? the environmental impact of testing open source java projects," in *Proceedings of the 5th ACM/IEEE International Conference on Automation of Software Test (AST)*. ACM, 2024, pp. 215–219.

[32] V. Hurdugaci and A. Zaidman, "Aiding software developers to maintain developer tests," in *16th European Conference on Software Maintenance and Reengineering, (CSMR)*. IEEE, 2012, pp. 11–20.

[33] A. Zaidman, B. Van Rompaey, A. van Deursen, and S. Demeyer, "Studying the co-evolution of production and test code in open source and industrial developer test processes through repository mining," *Empir. Softw. Eng.*, vol. 16, no. 3, pp. 325–364, 2011.

[34] A. Zaidman, B. Van Rompaey, S. Demeyer, and A. van Deursen, "Mining software repositories to study co-evolution of production & test code," in *First International Conference on Software Testing, Verification, and Validation (ICST)*. IEEE, 2008, pp. 220–229.

[35] B. Falcucci, F. Gomide, and A. Hora, "What do contribution guidelines say about software testing?" in *2025 IEEE/ACM 22nd International Conference on Mining Software Repositories (MSR)*. IEEE, 2025, pp. 434–438.

[36] G. Gousios, A. Zaidman, M. D. Storey, and A. van Deursen, "Work practices and challenges in pull-based development: The integrator's perspective," in *37th IEEE/ACM International Conference on Software Engineering (ICSE)*. IEEE, 2015, pp. 358–368.

[37] H. Li, H. Zhang, and A. E. Hassan, "Aidev: studying ai coding agents on github," *arXiv preprint arXiv:2602.09185*, 2026.

[38] ——, "AIDev: Studying AI coding agents on GitHub dataset," https://huggingface.co/datasets/hao-li/AIDev, 2025, accessed: 2026-05-19.

[39] W. Chatlatanagulchai, K. Thonglek, B. Reid, Y. Kashiwa, P. Leelaprute, A. Rungsawang, B. Manaskasemsak, and H. Iida, "On the use of agentic coding manifests: An empirical study of claude code," in *International Conference on Product-Focused Software Process Improvement*. Springer, 2025, pp. 543–551.

[40] W. Chatlatanagulchai, H. Li, Y. Kashiwa, B. Reid, K. Thonglek, P. Leelaprute, A. Rungsawang, B. Manaskasemsak, B. Adams, A. E. Hassan, and H. Iida, "Agent READMEs: An empirical study of context files for agentic coding," 2025. [Online]. Available: https://arxiv.org/abs/2511.12884

[41] C. Treude, S. Baltes, and M. Cheong, "Operationalizing ethics for AI agents: How developers encode values into repository context files," in *Proceedings of the 3rd ACM/IEEE International Conference on AI-powered Software (AIware 2026)*, Montreal, Canada, 2026.

[42] R. Xu and Y. Yan, "Agent skills for large language models: Architecture, acquisition, security, and the path forward," *arXiv preprint arXiv:2602.12430*, 2026.
[43] C. D. Manning, *Introduction to information retrieval*. Syngress Publishing,, 2008.
[44] A. Sureka and P. Jalote, "Detecting duplicate bug report using character n-gram-based features," in *2010 Asia Pacific software engineering conference*. IEEE, 2010, pp. 366–374.
[45] R. Hoda, *Socio-technical Grounded Theory for Qualitative Data Analysis*. Cham: Springer International Publishing, 2024, pp. 223–267.
[46] C. O'Connor and H. Joffe, "Intercoder reliability in qualitative research: Debates and practical guidelines," *International journal of qualitative methods*, vol. 19, p. 1609406919899220, 2020.
[47] S. N. Halpin, "Inter-coder agreement in qualitative coding: Considerations for its use," *American Journal of Qualitative Research*, vol. 8, no. 3, pp. 23–43, 2024.
[48] B. Ardic, M. Olsthoorn, and A. Zaidman, "Data package for "uncovering agents.md: What testing guidance open source projects provide to coding agents"," Figshare, 2026, https://doi.org/10.6084/m9.figshare.32653524.
[49] C. K. Roy, J. R. Cordy, and R. Koschke, "Comparison and evaluation of code clone detection techniques and tools: A qualitative approach," *Science of computer programming*, vol. 74, no. 7, pp. 470–495, 2009.
[50] A. Hora and R. Robbes, "Are coding agents generating over-mocked tests? an empirical study," in *Proc. of the International Conference on Mining Software Repositories (MSR)*. ACM, 2026, pp. 334–345.
[51] B. Ardic, C. Brandt, A. Khatami, M. Swillus, and A. Zaidman, "The qualitative factor in software testing: A systematic mapping study of qualitative methods," *Journal of Systems and Software*, vol. 227, p. 112447, 2025.