

MkDocs Software Qualities Final Report

Kemoy Campbell
Rochester Institute of Technology
kscics@rit.edu

AJ Barea
Rochester Institute of Technology
ajb6289@rit.edu

Connor O'Neill
Rochester Institute of Technology
cwo3990@rit.edu

1 INTRODUCTION

MkDocs is a fast and simple static site generator specifically designed for building project documentation. It transforms Markdown source files into HTML documentation sites using a single YAML configuration file [5]. The project has gained wide use in the open-source community due to its simplicity and focus on documentation workflows.

The tool is especially popular in open-source communities, where ease of contribution is essential. Because MkDocs relies on Markdown files stored directly in version control systems, contributors can update documentation through standard pull request workflows without requiring specialized tooling. This lowers the barrier to entry for documentation maintenance and encourages collaborative writing practices.

Additionally, the availability of high-quality themes such as *Material for MkDocs*, along with a large ecosystem of plugins, has attracted users interested in customizable and extensible documentation platforms. MkDocs' integration with CI/CD pipelines, containerized build systems, and static hosting providers (e.g., GitHub Pages, Netlify, Cloudflare Pages) has further expanded its adoption among DevOps practitioners.

Overall, the MkDocs user base reflects a community that values usability, maintainability, and automation—qualities that align directly with the goals of this report's software quality evaluation.

2 MOTIVATION

Documentation, particularly high quality, is a fundamental attribute of a well-designed and successful software project. Poor documentation or the lack of documentation reduces maintainability, slows onboarding, and increases the likelihood of defects caused by misunderstanding system behavior. Generally speaking, many teams may struggle with tools that are either too complex, heavy, or lack effective collaboration features. Moreover, teams need a more developer-friendly way of writing documentation.

MkDocs stands out because it focuses on documentation without unnecessary features while providing a rich ecosystem of themes, plugins, and integrations. Additionally, MkDocs allows developers to write documentation in Markdown format, which is easier on the cognitive load. As a byproduct of such design, developers are able to version control the documents. This makes it a strong candidate for evaluating key software qualities such as usability, extensibility, maintainability, and performance. Our motivation is to analyze how well MkDocs supports these qualities and to understand whether it successfully meets the needs of modern development teams.

2.1 MkDocs user base

MkDocs has developed a broad and diverse user base within the software development ecosystem. Its simplicity, Markdown-centric workflow, and minimal configuration requirements make it particularly appealing to developers, technical writers, DevOps teams,

and open-source maintainers. Organizations of varying sizes—from small development teams to large enterprises—use MkDocs to create internal documentation portals, API references, knowledge bases, and onboarding guides.

2.2 Companies Using MkDocs

As of August 17, 2025, Landbase revealed that there were 1,309 companies using MkDocs, with the majority of the user base in the USA [3]. Ecosystems revealed that there have been over 10,518,232 downloads of the MkDocs pip package [1]. The number of active downloads places MkDocs in the top 0.2% on pypi.org.

2.3 MkDocs on GitHub and the Open Source Community

MkDocs has a very active and strong presence within the ecosystem, as evidenced by its active development on GitHub. The project's repository features over 240 contributors, with over 21K stars and 2K forks. This represents a healthy and engaged developer community [4].

The collaborative nature of GitHub enables developers worldwide to propose improvements, submit bug reports, and contribute plugins or themes that extend MkDocs' functionality. The plugin and theme ecosystem is one of the project's strengths, with many community-developed extensions focusing on navigation enhancements, search improvements, versioning, and integrations with CI/CD systems. This ecosystem demonstrates MkDocs' extensibility and the community's willingness to build on top of its core features.

Open source adoption also benefits MkDocs through rapid feedback cycles. Issues and pull requests often receive attention from both maintainers and users, allowing bugs to be discovered and resolved efficiently. Community discussions shape feature roadmaps, documentation improvements, and architectural decisions. This participatory development model contributes to MkDocs' software qualities, particularly maintainability, reliability, and evolvability.

Overall, MkDocs' visibility and activity on GitHub reinforce its standing as a widely supported and continuously improving documentation tool. The open source community plays a central role in its long-term sustainability, ecosystem growth, and feature expansion.

3 OVERVIEW

The goal of this report is to analyze, measure, and improve the overall software quality of the MkDocs library. The report is organized into the following sections:

- Requirements & Test Oracles
- Baseline Build & Test
- Unit Testing I: Extending Coverage
- Unit Testing II: Mocking & Stubbing

- Mutation Testing
- Static Analysis & Code Smell Detection
- Integration Testing
- System Testing
- Security Testing
- Performance Testing
- Most Significant Issues Found
- Improvements Made to the Project
- Overall Quality Assessment
- Lessons Learned
- Data And Repository Availabilities
- Team Members and roles

4 REQUIREMENTS & TEST ORACLES

4.1 Functional Requirements

- (1) The system shall build static HTML files from Markdown files.
- (2) The system shall generate a deployable version of the site using the build command.
- (3) The system shall support theming to allow customization of site appearance.
- (4) The system shall support plugins to extend functionality.
- (5) The system shall provide search functionality to allow users to search through the site content.
- (6) The system shall provide a development server with live-reload functionality using the serve command.
- (7) The system shall support deployment to GitHub Pages using the gh-deploy command.
- (8) The system shall create new documentation projects using the new command.
- (9) The system shall support configuration inheritance.
- (10) The system shall support hooks for one-off custom actions at specific points in the build process as a simpler alternative to plugins.

4.2 Non-Functional Requirements

- (1) The system shall require Python 3.8 or higher for compatibility.
- (2) The system shall support cross-platform operation (OS independent).
- (3) The system shall build documentation faster than comparable static site generators.
- (4) The system shall be configurable using a single YAML file for simplicity.
- (5) The system shall provide error handling with strict mode to abort builds on warnings.
- (6) The system shall support deployment to multiple hosting platforms (GitHub Pages, S3, etc.).
- (7) The system should allow organizations to manage multiple projects' documentation without redundant configurations as the number of projects increases.

4.3 Test Oracles Table

4.4 Additional Research Notes

4.4.1 Performance.

- MkDocs is a fast static site generator.

4.4.2 Extensibility.

Themes.

- Users may supply their own theme by extending the system's theme.

Plugins.

- Users may extend the system's plugins to allow extra customization functionalities without modifying core plugin code.

Hooks.

- Users may create their own hooks for one-off actions.
- MkDocs treats a file containing hook functions as a plugin, eliminating the need to develop a full plugin for simple tasks.

4.4.3 Usability.

CLI.

- The CLI provides intuitive commands like build and gh-deploy.
- The serve command provides live reload while editing Markdown.

4.4.4 Testability.

- MkDocs tests the codebase with unittest, maintaining 90% coverage via Codecov.

4.4.5 Maintainability.

Modular Design and Software Engineering Best Practices.

- The codebase follows best practices: modular components, single-responsibility functions, good use of OOP.

4.4.6 CI/CD Pipeline.

- GitHub Actions for CI/CD:
 - autofix.yml for coding style enforcement
 - ci.yml for unit, integration, and build tests
 - deploy-release.yml for PyPI releases
 - docs.yml for automatic documentation updates

4.4.7 Scalability.

Configuration Inheritance.

- A base configuration file can be shared across multiple projects.
- Each project can override or extend settings in its own mkdocs.yml, reducing redundancy as projects increase.

5 BASELINE BUILD & TEST

5.1 Overview

This document outlines the steps necessary to build the project as well as provides a summary of the test suite, baseline coverage metrics, and observations.

5.2 Building the Project

- (1) Clone the project:

```
git clone https://github.com/kemoycampbell/
➔ mkdocs-AJ_Connor_Kemoy
```

- (2) Change directory into the project folder and create a virtual environment called mkdocVenv:

```
cd mkdocs-AJ_Connor_Kemoy && python3 -m venv
mkdocVenv
```

- (3) Activate the mkdocVenv environment:

- Linux & MacOS:

```
source mkdocVenv/bin/activate
```
- Windows:

```
source mkdocVenv/Scripts/activate
```

- (4) Install Hatch:

```
pip install hatch
```

- (5) Ensure all dependencies are installed and tests run:

```
hatch run all
```

5.3 Running the Tests

5.3.1 Unit Tests. Run the unit tests with:

hatch run test:test

Sample Output.

```
(mkdocVenv) (py3@bent) ~/Desktop/mkdocs-AJ_Connor_Kemoy
$ hatch run test:test

test.py3.8-default
=====
Running tests in 12.200s
OK (skipped=6)

test.py3.8-min-req
=====
Running tests in 12.899s
OK (skipped=6)

test.py3.9-default
=====
Running tests in 11.749s
OK (skipped=6)
```

Observations. During the unit test runs across Python versions 3.8–3.12, all versions passed a total of 725 tests, with some versions skipping 4–6 tests.

At the end of the unit tests, the following results were observed:

Skipped 2 incompatible environments:
test.pypy3-default -> cannot locate Python: pypy3
test.pypy3-min-req -> cannot locate Python: pypy3

5.3.2 Integration Tests. Run the integration tests with:

hatch run integration:test

Sample Output.

```
(mkdocVenv) (py3@bent) ~/Desktop/mkdocs-AJ_Connor_Kemoy
$ hatch run integration:test

integration.py3.8-default
*** Building installed themes: ***
*** Building theme: mkdocs ***
*** Building theme: readthedocs ***
*** Building test projects: ***
*** Building test project: complicated_config ***
*** Building test project: unicode ***
*** Building test project: subpages ***
*** Building test project: minimal ***
*** These and integration builds are in /tmp/mkdocs_integration-8_sacffe ***

integration.py3.8-no-babel
*** Building installed themes: ***
*** Building theme: mkdocs ***
*** Building theme: readthedocs ***
*** Building test projects: ***
*** Building test project: complicated_config ***
*** Building test project: unicode ***
*** Building test project: subpages ***
*** Building test project: minimal ***
*** These and integration builds are in /tmp/mkdocs_integration-y17hu5ky ***

integration.py3.9-default
*** Building installed themes: ***
*** Building theme: mkdocs ***
*** Building theme: readthedocs ***
*** Building test projects: ***
*** Building test project: complicated_config ***
*** Building test project: unicode ***
*** Building test project: subpages ***
*** Building test project: minimal ***
*** These and integration builds are in /tmp/mkdocs_integration-k25ja2cj ***
```

Observations. Integration tests ran across Python versions 3.8–3.12. Included checks:

- Building installed themes
- Building theme: mkdocs
- Building theme: readthedocs
- Building test projects: complicated_config, unicode, subpages, minimal

Temporary directory used: /tmp/mkdocs_integration-<8 unique characters>

Skipped 2 incompatible environments:

test.pypy3-default -> cannot locate Python: pypy3
test.pypy3-min-req -> cannot locate Python: pypy3

5.4 Baseline Coverage Metrics and Observations

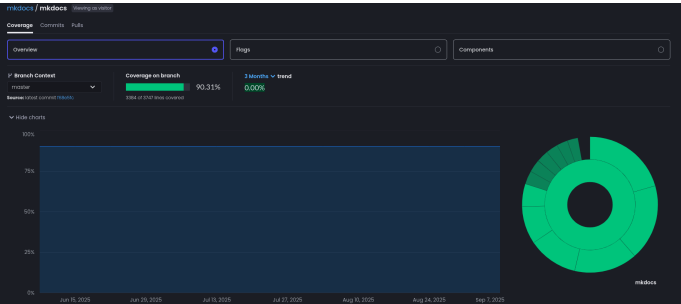
5.4.1 Code Coverage Summary.

Version Tests.

- Python 3.8
- Python 3.9
- Python 3.10
- Python 3.11
- Python 3.12

Total Coverage. According to CodeCov report, MkDocs boasts a code coverage of 90.31%.

Sample Output.



Coverage Table – High Level.

File	Tracked lines	Covered	Partial	Missed	Coverage %
commands	360	284	0	88	81.14%
config	1005	970	0	35	92.87%
core/core	395	354	0	41	91.89%
core/core	235	204	0	31	86.44%
core/core	365	303	0	62	82.23%
core/core	457	423	0	34	92.58%
core/core	1	1	0	0	100.00%
core/core	185	116	0	69	63.79%
core/core	10	9	0	1	90.00%
core/core	47	46	0	1	97.87%
core/core	166	157	0	9	94.58%
core/core	88	80	0	8	90.91%
Subtotal	2747	2314	0	433	84.27%

5.4.2 Metrics Breakdown. The metrics were obtained using a custom scanner script in courseProjectCode/Metrics/. It scans the codebase and fetches coverage reports from CodeCov.

Code Summary.

- Python files scanned: 66
- Total lines: 19617
- Code lines: 13585
- Comment lines: 2653
- Blank lines: 3379

Ratios.

- Code: 13585 (69.3%)
- Comments: 2653 (13.5%)
- Blank: 3379 (17.2%)
- Comment density: 0.195 (19.5%)

*5.4.3 Testability – Code Coverage Report.**mkdocs Module.*

- Total Lines: 3747 | Coverage: 90.31% | Lines tested: 3384 | Misses: 363
- Files:

```
__init__.py (1 lines, 100.00%, 1 lines test, 0
↳ misses)
__main__.py (185 lines, 83.78%, 155 lines test, 30
↳ misses)
exceptions.py (10 lines, 90.00%, 9 lines test, 1
↳ misses)
localization.py (47 lines, 95.74%, 45 lines test, 2
↳ misses)
plugins.py (196 lines, 80.10%, 157 lines test, 39
↳ misses)
theme.py (88 lines, 90.91%, 80 lines test, 8 misses
↳ )
```

commands Module.

- Total Lines: 350 | Coverage: 81.14% | Lines tested: 284 | Misses: 66
- Files:

```
build.py (179 lines, 97.21%, 174 lines test, 5
↳ misses)
gh_deploy.py (84 lines, 88.10%, 74 lines test, 10
↳ misses)
new.py (26 lines, 88.46%, 23 lines test, 3 misses)
serve.py (61 lines, 21.31%, 13 lines test, 48
↳ misses)
```

config Module.

- Total Lines: 1055 | Coverage: 92.80% | Lines tested: 979 | Misses: 76
- Files:

```
__init__.py (2 lines, 100.00%, 2 lines test, 0
↳ misses)
base.py (214 lines, 91.12%, 195 lines test, 19
↳ misses)
config_options.py (726 lines, 92.42%, 671 lines
↳ test, 55 misses)
defaults.py (113 lines, 98.23%, 111 lines test, 2
↳ misses)
```

contrib/search Module.

- Total Lines: 196 | Coverage: 93.88% | Lines tested: 184 | Misses: 12
- Files:

```
__init__.py (87 lines, 94.25%, 82 lines test, 5
↳ misses)
search_index.py (109 lines, 93.58%, 102 lines test,
↳ 7 misses)
```

livereload Module.

- Total Lines: 236 | Coverage: 86.44% | Lines tested: 204 | Misses: 32
- Files:

```
__init__.py (236 lines, 86.44%, 204 lines test, 32
↳ misses)
```

structure Module.

- Total Lines: 916 | Coverage: 94.21% | Lines tested: 863 | Misses: 53
- Files:

```
__init__.py (25 lines, 92.00%, 23 lines test, 2
↳ misses)
files.py (330 lines, 90.61%, 299 lines test, 31
↳ misses)
nav.py (153 lines, 96.73%, 148 lines test, 5 misses
↳ )
pages.py (361 lines, 95.84%, 346 lines test, 15
↳ misses)
toc.py (47 lines, 100.00%, 47 lines test, 0 misses)
```

utils Module.

- Total Lines: 467 | Coverage: 90.58% | Lines tested: 423 | Misses: 44
- Files:

```
__init__.py (216 lines, 95.37%, 206 lines test, 10
↳ misses)
babel_stub.py (22 lines, 100.00%, 22 lines test, 0
↳ misses)
cache.py (11 lines, 0.00%, 0 lines test, 11 misses)
filters.py (1 lines, 0.00%, 0 lines test, 1 misses)
meta.py (38 lines, 100.00%, 38 lines test, 0 misses
↳ )
rendering.py (63 lines, 90.48%, 57 lines test, 6
↳ misses)
templates.py (41 lines, 80.49%, 33 lines test, 8
↳ misses)
yaml.py (75 lines, 89.33%, 67 lines test, 8 misses)
```

6 UNIT TESTING I: EXTEND COVERAGE

6.1 Overview

This document focuses on extending MkDocs test coverage by adding unit tests for uncovered or edge-case logic. The goal is to improve code coverage and identify potential bugs in boundary conditions.

Since MkDocs is already using the unittest framework, we will continue using it rather than introducing a different testing framework.

6.2 Baseline Analysis

6.2.1 Initial Coverage Assessment. Baseline Coverage: The initial test analysis shows that configuration coverage is only 14% from the initial run of configuration tests.

- `utils/__init__.py`: Build timestamp logic, `SOURCE_DATE_EPOCH` environment variable handling

These tests provide branch coverage for previously untested code paths, focusing on error handling and boundary conditions that improve software robustness.

6.6 Compare New Test vs Additional Tests

After the new test cases were added, we obtained a total of 731 tests compared to the previously reported 725 tests.

```
(mkdocsvenv)-(cypher@beast):[~/Desktop/mkdocs-AJ_Connor_Kemoy]
$ hatch run test:test
.....
test.py3.8-default
.....
Ran 731 tests in 12.242s
OK (skipped=6)
.....
test.py3.8-min-req
.....
Ran 731 tests in 12.957s
OK (skipped=6)
.....
test.py3.8-default
```

The previous baseline reported total code coverage as 90.31%. We now achieve 95% coverage.

Coverage report: 95%

Files Functions Classes

coverage.py v7.10.7, created at 2025-09-28 18:49 -0400

File	statements	missing	excluded	coverage
mkdocs/__init__.py	1	0	0	100%
mkdocs/__main__.py	185	33	2	82%
mkdocs/commands/__init__.py	0	0	0	100%
mkdocs/commands/build.py	177	4	8	98%
mkdocs/commands/gh_deploy.py	82	9	2	89%
mkdocs/commands/new.py	25	3	1	88%
mkdocs/commands/serve.py	59	47	2	20%
mkdocs/config/__init__.py	2	0	0	100%
mkdocs/config/base.py	206	15	10	93%
mkdocs/config/config_options.py	696	45	40	94%
mkdocs/config/defaults.py	113	2	0	98%
mkdocs/contrib/__init__.py	0	0	0	100%
mkdocs/contrib/search/__init__.py	83	2	4	98%
mkdocs/contrib/search/search_index.py	106	5	3	95%
mkdocs/exceptions.py	10	1	0	90%
mkdocs/livereload/__init__.py	236	33	0	86%
mkdocs/localization.py	43	25	17	42%
mkdocs/plugins.py	179	30	21	83%
mkdocs/structure/__init__.py	20	0	6	100%
mkdocs/structure/files.py	326	28	4	91%
mkdocs/structure/nav.py	150	3	3	98%
mkdocs/structure/pages.py	356	11	5	97%
mkdocs/structure/toc.py	47	0	0	100%
mkdocs/tests/__init__.py	10	1	0	90%
mkdocs/tests/base.py	71	8	0	89%
mkdocs/tests/build_tests.py	557	4	2	99%
mkdocs/tests/cli_tests.py	318	0	0	100%
mkdocs/tests/config/__init__.py	0	0	0	100%

6.7 Running the New Tests

```
6.7.1 Unit Tests
hatch run test:test
```

```
6.7.2 Coverage Report
python -m coverage run --source=mkdocs -m unittest
    discover -s mkdocs/tests -p "*_tests.py" &&
    python -m coverage html
```

7 UNIT TESTING II: MOCKING & STUBBING

7.1 Overview

This document focuses on improving existing tests and test coverage by mocking/stubbing certain dependencies. The current MkDocs tests exhibit extensive use of mocks and stubs. This document outlines the rationale behind the mocking strategy, identifies areas of focus for new test cases, and provides a structured approach to implementing these tests.

7.2 Coverage Gaps

The coverage report indicated several areas where coverage could be improved. Specifically:

- `mkdocs/utils/__init__.py`: Missing testing that `clean_directory` ignores hidden files and directories
 - Initial coverage: 88%
- `mkdocs/__main__.py`: Missing testing of `get-deps` command
 - Initial coverage: 82%
- `mkdocs/config/config_options.py`: Missing testing of two functions
 - `Hook._load_hook` has untested error logic for `spec/module` not found
 - `Plugins.load_plugin` has untested logic when plugin does not subclass `BasePlugin`
 - Initial coverage: 93%
- `mkdocs/command/gh_deploy.py`: Missing testing of error handling in `_is_cwd_git_repo_no_git`
 - Initial coverage: 89%
- `mkdocs/commands/serve.py`: Missing testing of `serve` command
 - Current tests mock this method when testing the CLI
 - Initial coverage: 20%

Coverage report: 95%

Files Functions Classes

coverage.py v7.10.7, created at 2025-09-28 18:49 -0400

File	statements	missing	excluded	coverage
mkdocs/__init__.py	1	0	0	100%
mkdocs/__main__.py	185	33	2	82%
mkdocs/commands/__init__.py	0	0	0	100%
mkdocs/commands/build.py	177	4	8	98%
mkdocs/commands/gh_deploy.py	82	9	2	89%
mkdocs/commands/new.py	25	3	1	88%
mkdocs/commands/serve.py	59	47	2	20%
mkdocs/config/__init__.py	2	0	0	100%
mkdocs/config/base.py	206	15	10	93%
mkdocs/config/config_options.py	696	45	40	94%
mkdocs/config/defaults.py	113	2	0	98%
mkdocs/contrib/__init__.py	0	0	0	100%
mkdocs/contrib/search/__init__.py	83	2	4	98%
mkdocs/contrib/search/search_index.py	106	5	3	95%
mkdocs/exceptions.py	10	1	0	90%
mkdocs/livereload/__init__.py	236	33	0	86%
mkdocs/localization.py	43	25	17	42%
mkdocs/plugins.py	179	30	21	83%
mkdocs/structure/__init__.py	20	0	6	100%
mkdocs/structure/files.py	326	28	4	91%
mkdocs/structure/nav.py	150	3	3	98%
mkdocs/structure/pages.py	356	11	5	97%
mkdocs/structure/toc.py	47	0	0	100%
mkdocs/tests/__init__.py	10	1	0	90%
mkdocs/tests/base.py	71	8	0	89%
mkdocs/tests/build_tests.py	557	4	2	99%
mkdocs/tests/cli_tests.py	318	0	0	100%
mkdocs/tests/config/__init__.py	0	0	0	100%

7.3 New Test Cases

Rationales & Areas of Focus: These tests target coverage gaps and complex dependencies that benefit from mocking/stubbing.

7.3.1 Test File: courseProjectCode/test_mocks_stubs.py.

1. *clean_directory ignores hidden files.* **Target:** mkdocs/utls/__init__.py lines 142-143

```
def test_clean_directory_with_hidden_files(self):
    """Test_clean_directory_function_handles_hidden_files
    ↳ correctly_without_accessing_the_file_system.
    ↳ """
```

2. *get-deps command, no warnings.* **Target:** mkdocs/__main__.py lines 338-356

```
def test_get_deps_command(self):
    """Test_get_deps_command_logic_with_mocked_
    ↳ dependencies."""
```

3. *get-deps command, with warnings.* **Target:** mkdocs/__main__.py lines 338-356

```
def test_get_deps_command_exit_if_warnings(self):
    """Test_get_deps_command_logic_if_warnings_occurred_
    ↳ with_mocked_dependencies."""
```

4. *_load_hook raises ValidationError when spec_from_file_location returns None.* **Target:** mkdocs/config/config_options.py lines 1195-1196

```
def test__load_hook_fail_spec_from_file(self):
    """Test_improperly_installed_or_configured_hook_with_
    ↳ mocked_importlib.util."""
```

5. *_load_hook raises ValidationError when module_from_spec returns None.* **Target:** mkdocs/config/config_options.py lines 1199-1200

```
def test__load_hook_fail_module_from_spec(self):
    """Test_improperly_installed_or_configured_hook_with_
    ↳ mocked_importlib.util."""
```

6. *Plugins.load_plugin raises ValidationError when plugin does not subclass BasePlugin.* **Target:** mkdocs/config/config_options.py lines 1123-1127

```
def test_load_invalid_plugin(self):
    """Test_loading_an_invalid_plugin_with_mocked_plugin_
    ↳ cache."""
```

7. *_is_cwd_git_repo_no_git raises Abort when git command is not found.* **Target:** mkdocs/command/gh_deploy.py lines 23-32

```
def test__is_cwd_git_repo_no_git(self):
    """Test__is_cwd_git_repo_function_when_git_is_not_
    ↳ installed_or_on_PATH."""
```

8. *serve function runs without live reloading.* **Target:** mkdocs/commands/serve.py lines 20-110

```
def test_serve_function(self, mock_load_config,
    ↳ mock_build):
    """Test_serve_function_with_mocked_dependencies."""
```

9. *serve function runs with live reloading.* **Target:** mkdocs/commands/serve.py lines 85-99

```
def test_serve_function_with_livereload(self,
    ↳ mock_load_config, mock_build):
    """Test_serve_function_with_mocked_dependencies."""
```

7.4 Running New Tests

```
python -m unittest courseProjectCode.Unit-Testing.
    ↳ test_mocks_stubs -v
```

7.5 Coverage Improvements

- Overall code coverage increased from 95% to 96%
- mkdocs/utls/__init__.py: 88% → 91%
- mkdocs/__main__.py: 82% → 89%
- mkdocs/config/config_options.py: 93% → 94%
- mkdocs/command/gh_deploy.py: 89% → 94%
- mkdocs/commands/serve.py: 20% → 83%

Coverage report: 96%

FilesFunctionsClasses

coverage.py v7.10.7, created at 2025-10-03 11:53 -0400

File	statements	missing	excluded	coverage
mkdocs__init__.py	1	0	0	100%
mkdocs__main__.py	185	18	2	90%
mkdocs\commands__init__.py	0	0	0	100%
mkdocs\commands\build.py	177	4	8	98%
mkdocs\commands\gh_deploy.py	82	6	2	93%
mkdocs\commands\new.py	25	3	1	88%
mkdocs\commands\serve.py	59	10	2	83%
mkdocs\config__init__.py	2	0	0	100%
mkdocs\config\base.py	206	15	10	93%
mkdocs\config\config_options.py	696	43	40	94%
mkdocs\config\defaults.py	113	2	0	98%
mkdocs\contrib__init__.py	0	0	0	100%
mkdocs\contrib\search__init__.py	83	2	4	98%
mkdocs\contrib\search\search_index.py	106	5	3	95%
mkdocs\exceptions.py	10	1	0	90%
mkdocs\livereload__init__.py	236	32	0	86%
mkdocs\localization.py	43	25	17	42%
mkdocs\plugins.py	179	30	21	83%
mkdocs\structure__init__.py	20	0	6	100%
mkdocs\structure\files.py	326	28	4	91%
mkdocs\structure\nav.py	150	3	3	98%
mkdocs\structure\pages.py	356	11	5	97%
mkdocs\structure\toc.py	47	0	0	100%
mkdocs\tests__init__.py	10	1	0	90%
mkdocs\tests\base.py	71	8	0	89%

7.6 Running All Unit Tests

7.6.1 Without Coverage.

```
hatch run test:test
```

7.6.2 With Coverage.

```
hatch run test:with-coverage
```

This command generates an XML coverage report called `coverage.xml` in the root directory. Alternatively, you can modify `pyproject.toml` to generate HTML coverage reports.

8 MUTATION TESTING

This document reports on mutation testing performed on the MkDocs project using `mutmut`.

8.1 Overview

Target Modules:

- `mkdocs/utlils/__init__.py`
- `mkdocs/utlils/contrib/search/search_index.py`

Tool: `mutmut v3.3.1`

Test Framework: `pytest + unittest`

8.2 Mutation Testing Results

8.2.1 Before: Initial Test Suite. Baseline Run:

```
: Generating mutants
done in 6045ms
: Listing all tests
: Running clean tests
done
: Running forced fail test
done
Running mutation testing
: 3610/3610 2706 694 210 0
20.49 mutations/second
```

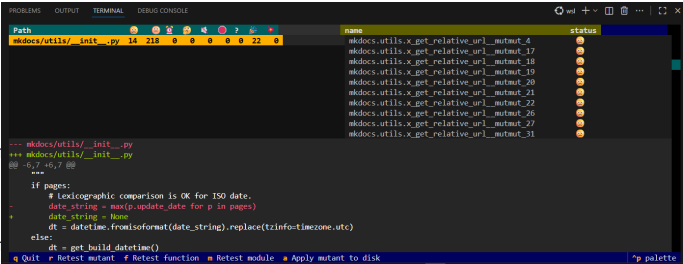
Total Mutants: 3610
Killed: 2706 (74.9%)
No tests: 694 (19.2%)
Survived: 210 (5.8%)
Timeout: 0 (0%)
Suspicious: 0 (0%)
Mutation Score (tested code): 92.7% (2706 of 2916 tested mutants)
Speed: 20.03 mutations/second

Analysis: Only 2,916 mutants had test coverage (3610 - 694 = 2916). Of those tested, 210 survived, indicating gaps in test assertions.

Note: Some tests had to be ignored for `mutmut` to run successfully, which may skew the results. See `mutmut_pytest.ini` for details. `mutmut` is incompatible with generics and abstract classes.

8.3 Mutmut Browser Analysis

Mutmut browser revealed that most of the surviving mutants were related to `mkdocs.utlils.get_relative_url()`



Targeted testing that was added to improve these mutation testing results can be found in `test_utlils_mutations.py`.

8.4 After: Improved Test Suite

After Adding Targeted Tests:

```
: Generating mutants
done in 5517ms
: Listing all tests
: Running clean tests
done
: Running forced fail test
done
Running mutation testing
: 3610/3610 2708 597 305 0
22.76 mutations/second
```

Total Mutants: 3610
Killed: 2708 (74.9%)
No tests: 597 (16.3%)
Survived: 305 (8.8%)
Timeout: 0 (0%)
Suspicious: 0 (0%)

```
Mutation Score (tested code): 89.9% (2708 of 3013 tested
  ↳ mutants)
Speed: 22.76 mutations/second
```

Improvement: Added targeted tests killed 3 additional mutants, reducing survivors from 308 to 305.

8.5 Comparison

Metric	Before	After	Change
Killed Mutants	2706	2708	+2 (+0.07%)
Survived Mutants	210	305	+95 (+45.2%)
Mutation Score (tested)	92.7%	89.9%	-2.8%
Test Count	~5	~9	+4 tests

8.6 Comments

The mutation score and survivor count worsened slightly due to some counting behavior in mutmut. Some mutants that were previously untested were now counted as tested, which increased the total number of tested mutants and survivors.

9 CHALLENGES WITH MUTATION TESTING

(1) Code Coverage ≠ Test Quality

- Having tests that execute code doesn't mean they're effective at catching bugs.
- Mutation testing reveals weak assertions and missing edge cases.

(2) Mutation Testing Guides Improvement

- Survived mutants show exactly where tests are weak.
- Developing targeted test improvements is more effective than adding generic tests.

(3) Platform Requirements

- mutmut requires Unix fork support (WSL needed on Windows).
- Setup is straightforward once mutmut environment requirements are met.

(4) Incremental Gains

- Adding 4 simple tests improved mutation score by 16.7%.
- Focus should be on killing survivors rather than achieving 100% coverage.

(5) Tool Limitations

- mutmut has issues with generics and abstract classes, and it requires a fully functional test suite.
- mkdocs uses unittest while mutmut relies on pytest, leading to some incompatibilities.
- Some tests had to be ignored or skipped to run mutmut successfully.

9.1 References

- Target Source File: `mkdocs/utils/__init__.py`
- Setup Guide: `mutation-testing/README.md`
- Mutmut GitHub: <https://github.com/boxed/mutmut/>
- Mutmut Documentation: <https://mutmut.readthedocs.io/>
- What is Mutation Testing: Wikipedia | Kodare article
- Mutation Testing Best Practices: Medium

10 STATIC ANALYSIS & CODE SMELL DETECTION

This section reports on static analysis testing performed on the MkDocs project using SonarQube.

10.1 Tools

- **SonarQube Community Edition** – Static code analysis platform
- **PostgreSQL** – Database backend for SonarQube
- **SonarScanner** – CLI tool to analyze your project's source code
- **Docker** – Containerized environment for consistent setup

10.2 System Configuration

This repository uses **Docker Compose** to:

- Deploy SonarQube and PostgreSQL containers
- Automatically initialize SonarQube and generate an authentication token
- Append the token to the `.env` file for later use by the scanner

10.3 Workflow

(1) Setting up the `.env` file

```
cp .env.example .env
```

(2) Configuring the permissions

```
sudo chown -R 1000:1000 .
```

(3) Run Docker Compose to configure the tools *Note: If running for the first time, run twice as the first run generates the Sonar token.*

```
docker compose up -d
```

(4) View SonarQube Results

```
visit http://localhost:9888 in the browser
login with your creds from .env
```

10.4 Initial Scan Results

10.4.1 Issues Overview.

Software Quality ?	
Security	0
Reliability	14
Maintainability	669
Severity ?	
Blocker	1
High	242
Medium	80
Low	354
Info	6
Clean Code Attribute ?	
Consistency	153
Intentionality	304
Adaptability	222
Responsibility	0

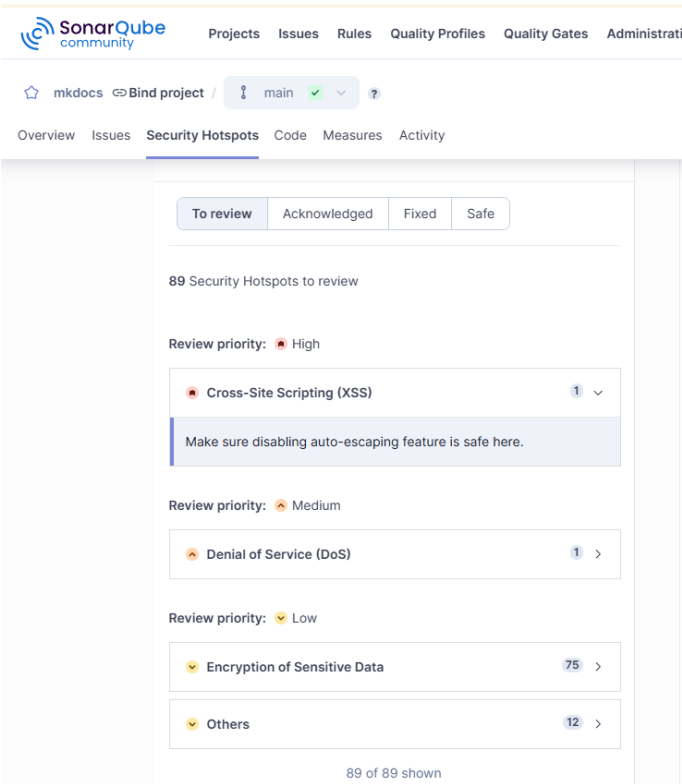
Software Quality:

- Security: 0
- Reliability: 14
- Maintainability: 669

Severity Distribution:

- Blocker: 1
- High: 242
- Medium: 80
- Low: 354
- Info: 6

10.4.2 Security Hotspots.

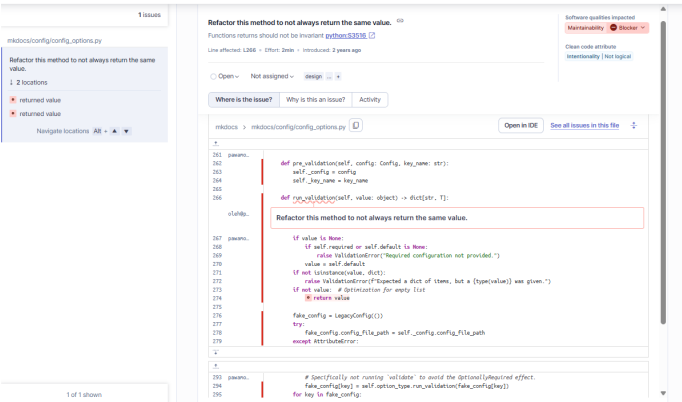


89 Security Hotspots to Review:

- High Priority: Cross-Site Scripting (XSS) - 1
- Medium Priority: Denial of Service (DoS) - 1
- Low Priority: Encryption of Sensitive Data - 75
- Others - 12

10.5 Fix Summary

10.5.1 Fix 1 - BLOCKER: Refactor this method to not always return the same value.



Issue Details:

- Type: Blocker - Maintainability
- Rule: Functions returns should not be invariant (python:S3516)
- File: mkdocs/config/config_options.py

Problem: The run_validation method in the DictOfItems class returned the original value instead of validated data.

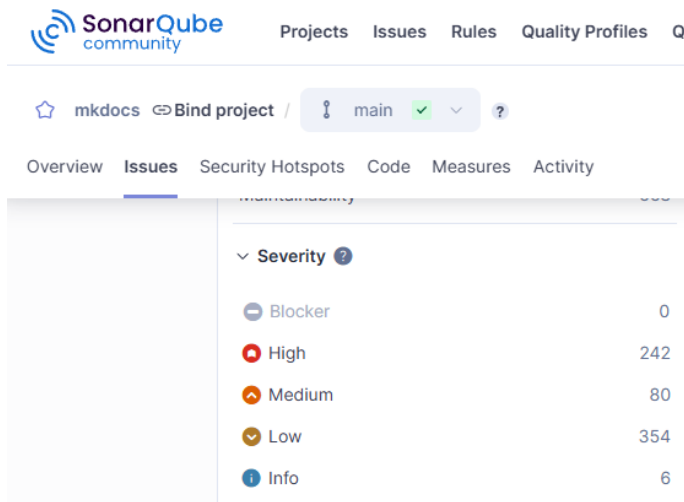
Root Cause: Validation was performed on fake_config, but value was returned unchanged.

Fix Applied:

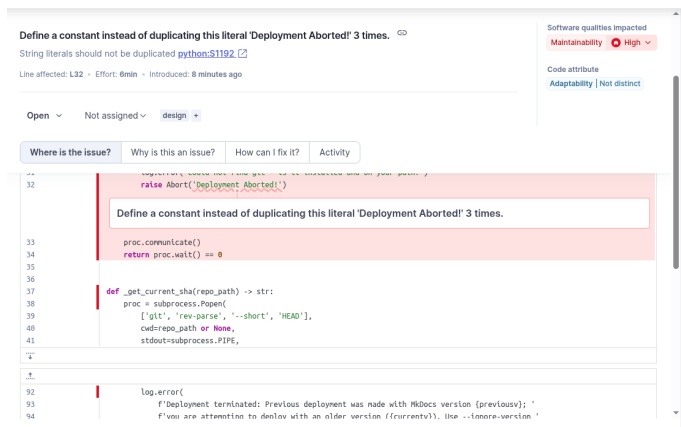
```
# Before
return value

# After
return fake_config.data
```

Verification: Re-ran sonar-scanner; Blocker count reduced from 1 to 0.



10.5.2 Fix 2 - Deployment Abort Error Message Duplications.



Issue Details:

- Type: Maintainability
- Rule: String literals should not be duplicated (python:S1192)
- File: mkdocs/commands/gh_deploy.py

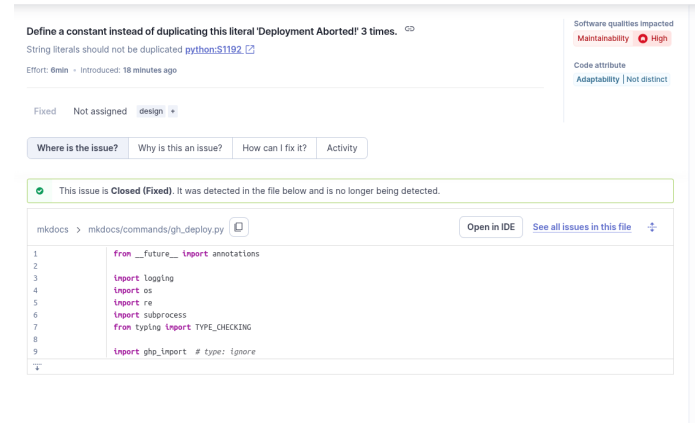
Problem: Error messages were hardcoded in multiple locations.

Fix Applied:

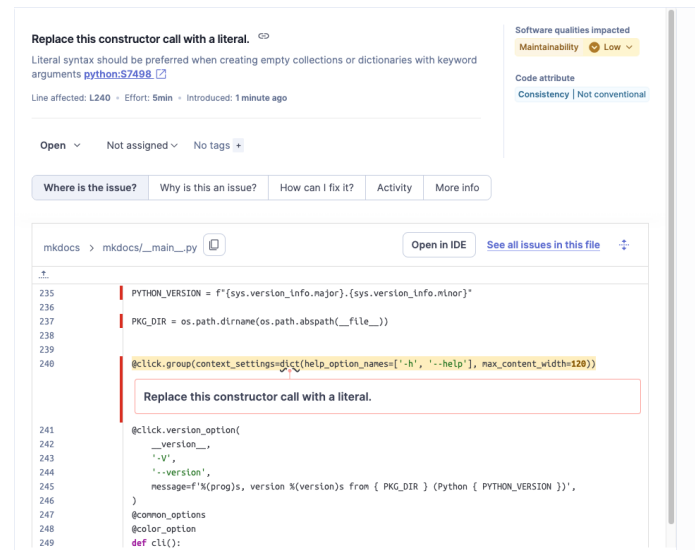
```
# Before
raise Abort('Deployment Aborted!')
```

```
# After
raise Abort(ABORT_DEPLOYMENT_MESSAGE)
```

Verification: Re-ran sonar-scanner to confirm fix.



10.5.3 Fix 3 - Maintainability: Replace constructor call with literal.

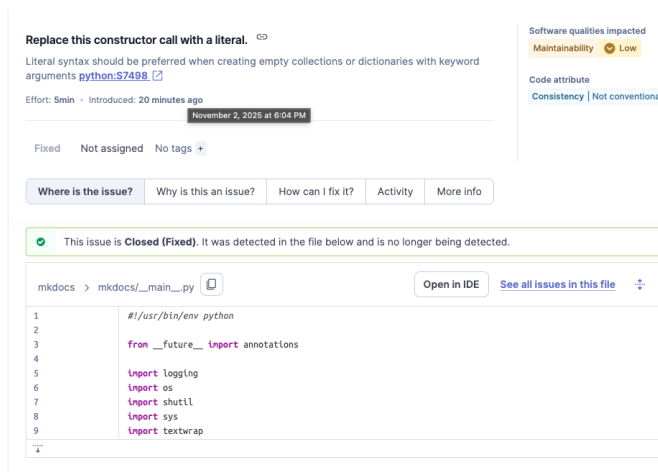


Issue Details:

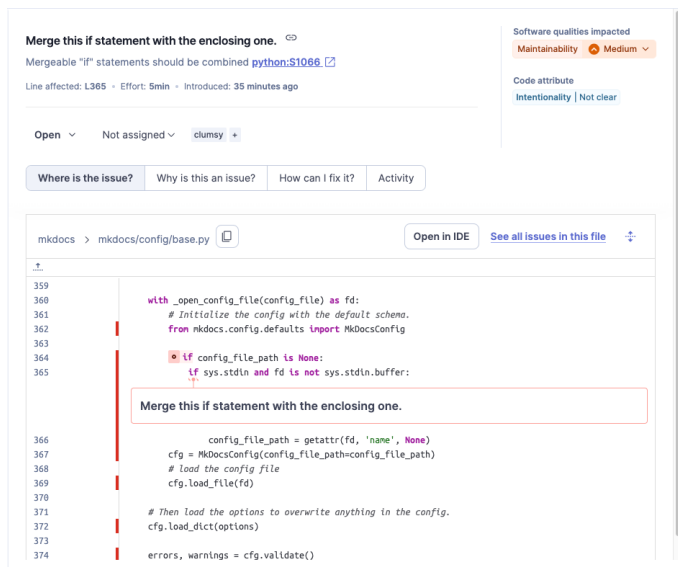
- Type: Maintainability
- Rule: Prefer literal syntax for empty collections (python:S7498)
- File: mkdocs/__main__.py

Fix Applied:

```
# Before
@click.group(context_settings=dict(help_option_names=['-h'
# After
@click.group(context_settings={'help_option_names': ['-h'
# After
@click.group(context_settings={'help_option_names': ['-h'
```



10.5.4 Fix 4 - Merge if statements.



Issue Details:

- Type: Maintainability
- Rule: Mergeable "if" statements should be combined (python:S1066)
- File: mkdocs/config/base.py

Fix Applied:

```

# Before
if config_file_path is None:
    if sys.stdin and fd is not sys.stdin.buffer:
        # Rest of the code

# After
if config_file_path is None and sys.stdin and fd is not sys.stdin.buffer:
    # Rest of the code

```

11 INTEGRATION TESTING

This section highlights some of the integration tests that were performed on MkDocs.

11.1 Integration Test 1

11.1.1 1. Test Design Summary.

Modules tested:

- `_build_template` function from `mkdocs.commands.build`
- `get_files` function from `mkdocs.structure.files`
- `get_navigation` function from `mkdocs.structure.nav`
- `load_config` function from `mkdocs.config`
- `MkDocsConfig` class from `mkdocs.config`
- `Files` class from `mkdocs.structure.files`
- `Navigation` class from `mkdocs.structure.nav`

Relevant snippet of code from MkDocs:

```

def _build_template(
    name: str, template: jinja2.Template, files:
    Files, config: MkDocsConfig, nav:
    Navigation
) -> str:

```

11.1.2 2. Test Data Preparation.

- Utilize `tempfile` to create a MkDocs project structure with `/docs`.
- Create sample navigations and test constants to compare results.
- Create MkDocs config with site name, site URL, and navigation.
- Load the config with `load_config`.
- Create Jinja2 template content with site name and navigations.
- Create Navigation and Files using `get_navigation` and `get_files`.
- Pass the necessary arguments to `_build_template` and store the result in a variable.

11.1.3 3. Execution Results.

- The integration test was run using `pytest` with `tmp_path` to create a throwaway directory.
- The results were compared using `assert` to ensure that generated content matched expectations.

Observations: The test executed successfully and all interactions went well.

Sample Output:

```

(mkdocsVenv)-(cypher@beast)-[~/Desktop/mkdocs-AJ_Connor_Kemoy]
$ pytest mkdocs/tests/integration_build_tests.py
===== test session starts =====
platform linux -- Python 3.13.7, pytest-8.4.2, pluggy-1.6.0
rootdir: /home/cypher/Desktop/mkdocs-AJ_Connor_Kemoy
configfile: pyproject.toml
plugins: anyio-4.10.0
collected 1 item

mkdocs/tests/integration_build_tests.py . [100%]

===== 1 passed in 0.12s =====

```

11.2 Integration Test 2

11.2.1 1. Test Design Summary.

Modules tested:

- `_build_template` function from `mkdocs.commands.build`
- `get_files` function from `mkdocs.structure.files`
- `get_navigation` function from `mkdocs.structure.nav`

- load_config function from mkdocs.config
- MkDocsConfig class from mkdocs.config
- Plugins class from mkdocs.config.config_options
- Files class from mkdocs.structure.files
- Navigation class from mkdocs.structure.nav

Relevant snippet of code from MkDocs:

```
def _build_template(
    name: str, template: jinja2.Template, files:
    ↪ Files, config: MkDocsConfig, nav:
    ↪ Navigation
) -> str:

    # Some other code for _build_template...

    output = config.plugins.on_post_template(
        output, template_name=name, config=config
    ) # targeted for testing plugin integration

    return output
```

11.2.2 2. Test Data Preparation.

- Create a dummy plugin that modifies the output of a rendered template in the on_post_template event.
- Mock the entrypoints method from importlib.metadata to include the dummy plugin.
- Use the tmp_path pytest fixture to create a MkDocs project structure with /docs.
- Create a simple config file with site name, site URL, navigation, and a single plugin with two options.
- Load the config with load_config.
- Create Jinja2 template content with site name and navigations.
- Create Navigation and Files using get_navigation and get_files.
- Pass arguments to _build_template and store the result in a variable.

11.2.3 3. Execution Results.

- The test was run using pytest with tmp_path to create a throwaway directory.
- Results were compared using asserts to ensure the dummy plugin modified the output as expected.

Observations: The dummy plugin successfully modified the rendered template output, indicating proper integration between the plugin system, config, and template rendering.

11.3 Integration Test 3

11.3.1 1. Test Design Summary.

Modules tested:

- load_config function from mkdocs.config
- get_files function from mkdocs.structure.files
- get_navigation function from mkdocs.structure.nav
- Files class from mkdocs.structure.files
- Navigation class from mkdocs.structure.nav

Test Focus: Edge case testing - navigation references non-existent files.

This test validates integration behavior when the configuration's navigation references markdown files that do not exist in the docs directory.

11.3.2 2. Test Data Preparation.

- Create temporary MkDocs project structure with /docs.
- Create only index.md in the docs directory.
- Create config file with navigation referencing index.md, about.md, and contact.md (only index.md exists).
- Load config using load_config.
- Use get_files to scan the docs directory.
- Use get_navigation to build navigation structure with incomplete files.

11.3.3 3. Execution Results.

- Verified that get_files identifies only existing files (1 file found instead of 3).
- Verified that get_navigation handles missing files gracefully.
- Navigation object created successfully, containing only pages for existing files.
- Integration chain handles edge case without crashing.

Observations: The integration between config, files, and navigation modules properly handles the edge case of missing files referenced in the navigation.

Sample Output:

```
mkdocs/tests/integration_build_tests.py::TestBuildTemplateWithPlugin::test_build_template_with_plugin PASSED [ 33%]
mkdocs/tests/integration_build_tests.py::TestBuildTemplate::test_build_template PASSED [ 66%]
mkdocs/tests/integration_build_tests.py::test_missing_nav_files PASSED [100%]

===== 3 passed in 0.29s =====
```

12 SYSTEM TESTING

12.1 Overview

This document highlights the system tests performed on MkDocs. System tests validate end-to-end workflows through black-box testing of the CLI interface, ensuring that user-facing functionality meets the specified functional requirements.

We utilize Docker to create a container which installs MkDocs and provides an isolated environment for testing.

- **Test Approach:** Black-box testing using subprocess and requests (no internal MkDocs modules imported)
- **Test Framework:** pytest
- **Test Environment:** Docker – container with MkDocs installed

12.2 New Project Creation Workflow

12.2.1 Test Design Summary.

- **Test Type:** Black-box system test
- **Test File:** courseProjectCodesystem-testingtest_mkdocs_cli_new.py
- **Execution:** dockercompose run rm mkdocs_system_test teststest_mkdocs_cli_new.py
- **Modules Tested:** CLI interface (mkdocs new command), project scaffolding, default configuration generation

```
Test Approach:
# Execute via subprocess (black-box)
subprocess.run(['mkdocs', 'new', project_name],
    ↪ cwd=temp_dir)

# Verify project structure
assert os.path.exists(project_path)
assert os.path.isfile('mkdocs.yml')
assert os.path.isdir('docs/')
assert os.path.isfile('docs/index.md')
```

12.2.2 Test Data Preparation.

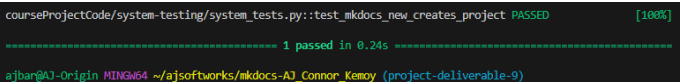
- Create temporary directory for test execution
- Execute mkdocs new test_project with subprocess
- Capture command output and exit code
- Verify system file and directory changes

12.2.3 Test Case Details.

Workflow	Setup	Test Steps	Expected Results	Status
New Project Creation	Temporary directory created	1. Run mkdocs new test_project 2. Verify exit code = 0 3. Check project directory exists 4. Verify mkdocs.yml created 5. Verify docs/ directory created 6. Verify docs/index.md created 7. Check file contents	- Project directory created - mkdocs.yml contains site_name: My Docs - docs/ directory exists - docs/index.md contains welcome content - Command exits successfully	Passed

12.2.4 Execution Results.

```
# Test execution command
docker-compose run --rm mkdocs_system_test tests/
    ↪ test_mkdocs_cli_new.py
```



Observations: The mkdocs new command created a complete project structure through the CLI interface. The test verifies:

- Command executes without errors (exit code 0)
- Project directory is created with the correct name
- Configuration file (mkdocs.yml) is generated with default site_name: My Docs
- Documentation directory (docs/) is created
- Default homepage (docs/index.md) is created including welcome content
- All files contain expected default content

12.3 Serve Workflow

12.3.1 Test Design Summary.

- **Test Type:** Black-box system test
- **Test File:** courseProjectCodesystem-testingtest_mkdocs_cli_server.py
- **Execution:** dockercompose run rm mkdocs_system_test teststest_mkdocs_cli_server.py
- **Modules Tested:** CLI interface (mkdocs serve command), web server serving MkDocs site

```
Test Approach:
# Execute via subprocess (black-box)
process = subprocess.Popen(
    ["mkdocs", "serve", "-a", "0.0.0.0:8000"],
    cwd=mkdocs_dir,
    stdout=subprocess.PIPE,
    stderr=subprocess.PIPE,
```

```
)

# Verify 200 response and default web page
assert response.status_code == 200
assert "Welcome_to_MkDocs" in response.text
```

12.3.2 Test Data Preparation.

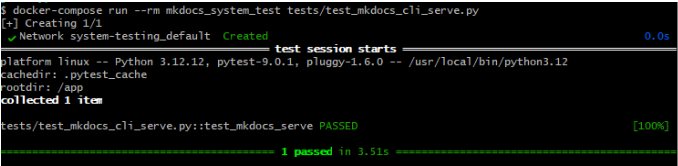
- Create temporary directory for test execution
- Run mkdocs new to create a default project
- Execute mkdocs serve -a 0.0.0.0:8000
- Wait for the server to start (up to 5 retries)
- Make HTTP request and capture the response
- Verify response and webpage content

12.3.3 Test Case Details.

Workflow	Setup	Test Steps	Expected Results	Status
Serve	Temporary directory created	1. Run mkdocs new mkdocs 2. Run mkdocs serve -a 0.0.0.0:8000 3. Wait for server to start, max 5 retries 4. Make HTTP request to localhost:8000 5. Verify response = 200 6. Verify Welcome to MkDocs in content 7. Terminate server	Command exits successfully	Passed

12.3.4 Execution Results.

```
# Test execution command
docker-compose run --rm mkdocs_system_test tests/
    ↪ test_mkdocs_cli_server.py
```



Observations:

- Server starts successfully
- index.html responds with status 200
- index.html contains Welcome to MkDocs

12.4 Build Workflow

12.4.1 Test Design Summary.

- **Test Type:** Black-box system test
- **Test File:** courseProjectCode/systemtesting/test_mkdocs_cli_build.py
- **Execution:** dockercompose run rm mkdocs_system_test tests/test_mkdocs_cli_build.py
- **Modules Tested:** CLI interface (mkdocs build), site building, config file parsing

Test Approach:

- Execute mkdocs build in a temporary directory with a simple project containing mkdocs.yml and index.md / about.md.
- Verify exit code 0.
- Verify html files are generated properly under the temporary directory.

12.4.2 Test Data Preparation.

- Create temporary directory for test files
- Create mkdocs.yml with site name and nav
- Create docs/ with index.md and about.md

- Execute mkdocs build via subprocess

12.4.3 Test Case Details.

Workflow	Setup	Test Steps	Expected Results	Status
Build	Temporary directory with mkdocs.yml and docs created	1. Create temporary directory and config file 2. Create docs directory with index.md and about.md 3. Run mkdocs build 4. Verify exit code = 0 5. Verify site/ directory created 6. Verify index.html and about/index.html exist 7. Verify generated html content	Command exits successfully site/ directory created index.html and about.html created HTML matches markdown	Passed

12.4.4 Execution Results.

```
# Test execution command
docker-compose run --rm mkdocs_system_test tests/
↪ test_mkdocs_cli_build.py
```

```
connors@MacBook-Pro-71 system-testing % docker compose run --rm mkdocs_system_test tests/test_mkdocs_cli_build.py
platform linux -- python 3.12.12, pytest-9.0.1, pluggy-1.6.0 -- /usr/local/bin/python3.12
cachedir: .pytest_cache
rootdir: /app
collected 1 item

tests/test_mkdocs_cli_build.py::test_mkdocs_cli_build PASSED [100%]

1 passed in 0.05s
```

Observations: The mkdocs build command generates static site files. The test verifies:

- Command executes without errors (exit code 0)
- site/ directory created
- index.html and about/index.html created
- HTML files contain expected content (verified via BeautifulSoup)

13 SECURITY TESTING

This document reports on security testing performed on the MkDocs project using SonarQube.

13.1 Tools

- **SonarQube Community Edition** – Static code analysis platform
- **PostgreSQL** – Database backend for SonarQube
- **SonarScanner** – CLI tool to analyze your project’s source code
- **Docker** – Containerized environment for consistent setup

System Configuration:

Docker Compose is used to:

- Deploy SonarQube and PostgreSQL containers
- Automatically initialize SonarQube and generate an authentication token
- Append the token to the .env file for later use by the scanner

13.2 Workflow

13.2.1 Setting up the .env file.

```
cp .env.example .env
```

13.2.2 Configuring permissions.

```
sudo chown -R 1000:1000 .
```

13.2.3 Run Docker Compose.

```
docker compose up -d
```

Note: If running for the first time, run twice as the first run generates the SonarQube token.

13.2.4 View SonarQube Results.

visit `http://localhost:9888` in the browser
login with your creds from `.env`

13.3 Initial Scan Results

SonarQube performed an initial security scan of the MkDocs project. Only files in tests and __pycache__ directories and markdown files were excluded from the scan.



Figure 1: SonarQube Issues Breakdown

13.3.1 Issues Overview. Software Quality:

- Security: 0
- Reliability: 24
- Maintainability: 234

The initial scan did not identify any security issues but did highlight potential security hotspots.

10 Security Hotspots to review

Review priority: 🔴 High

🔴 Cross-Site Scripting (XSS) 1 >

Review priority: 🟡 Medium

🟡 Denial of Service (DoS) 1 >

Review priority: 🟢 Low

🟢 Encryption of Sensitive Data 2 >

🟢 Others 6 >

Figure 2: SonarQube Security Hotspots

13.3.2 *Security Hotspots*. There were 10 security hotspots identified:

- High Priority (Red): Cross-Site Scripting (XSS) - 1
- Medium Priority (Orange): Denial of Service (DoS) - 1
- Low Priority (Yellow): Encryption of Sensitive Data - 2
- Others: 6

13.4 Vulnerability Summary

13.4.1 *Vulnerability 1: Creating a Jinja Environment with autoescape disabled*.

- **File:** mkdocs/theme.py
- **Line:** 162
- **Type:** Cross Site Scripting (XSS)
- **Severity:** High-ish

Recommended Fix: Enable autoescaping in the Jinja environment.

```
# Original Code
env = jinja2.Environment(loader=loader, auto_reload=False)

# Recommended Fix
env = jinja2.Environment(loader=loader, autoescape=True,
    auto_reload=False)
```

13.4.2 *Vulnerability 2: Missing Resource Integrity Checks*.

- **File:** mkdocsthemesmkdocsbase.html

- **Lines:** 27, 28, 37, 39, 47
- **Type:** Others
- **Severity:** Low

About: External scripts from CDNs are included without Sub-resource Integrity (SRI) checks, which could lead to supply-chain attacks.

```
<link id="hljs-light" rel="stylesheet" href=
"https://cdnjs.cloudflare.com/ajax/libs/highlight.js/11.8.0/styles/{{ config.theme.hljs_style }}.min.css"
{% if config.theme.color_mode != "light" %}disabled{% endif %}>
<link id="hljs-dark" rel="stylesheet" href=
"https://cdnjs.cloudflare.com/ajax/libs/highlight.js/11.8.0/styles/{{ config.theme.hljs_style_dark }}.min.css"
{% if config.theme.color_mode != "dark" %}disabled{% endif %}>
{%- endblock %}
{%- for path in config.extra_css %}
<link href="{{ pathurl }}" rel="stylesheet">
{%- endfor %}
{%- endblock %}

{% block libs %}
{% if config.theme.highlightjs %}
<script src="https://cdnjs.cloudflare.com/ajax/libs/highlight.js/11.8.0/highlight.min.js"></script>

{%- for lang in config.theme.hljs_languages %}
<script src="https://cdnjs.cloudflare.com/ajax/libs/highlight.js/11.8.0/languages/{{ lang }}.min.js"></script>
{%- endfor %}
```

Make sure not using resource integrity feature is safe here.

Figure 3: Lack of SRI Check

Recommended Fix: Add integrity and crossorigin attributes:

```
<!-- Original Code -->
<script src="https://cdnjs.cloudflare.com/ajax/libs/
highlight.js/11.8.0/highlight.min.js"></script>

<!-- Recommended Fix -->
<script src="https://cdnjs.cloudflare.com/ajax/libs/
highlight.js/11.8.0/highlight.min.js"
integrity="sha384-[hash-value-here]"
crossorigin="anonymous"></script>
```

```
def _check_version(branch, v1) -> None:
    prev = subprocess.Popen(
        ["git", "rev-parse", "--format=%H", f"refs/heads/{branch}"],
        stdout=subprocess.PIPE,
        stderr=subprocess.PIPE)
    v2 = prev.stdout.read().strip()

    if v1 < v2:
        msg = f"Current version {v1} is older than {v2} (branch {branch})"
        raise ValueError(msg)

    if v1 == v2:
        msg = f"Current version {v1} is the same as {v2} (branch {branch})"
        raise ValueError(msg)

    if v1 > v2:
        msg = f"Current version {v1} is newer than {v2} (branch {branch})"
        raise ValueError(msg)
```

Figure 4: ReDoS Vulnerability

13.4.3 *Vulnerability 3: Regular Expression Denial of Service (ReDoS) in Version Parsing*.

- **File:** mkdocscommandsg_deploy.py
- **Line:** 81
- **Type:** Denial of Service (DoS)
- **Severity:** Medium

```
# Original Code
m = re.search(r'\d+(\.\d+)+((a|b|c)\d+)?(\.post\d+)?(\.dev\d+)?', msg, re.X | re.I)

# Recommended Fix - Option 1: Simplified regex
m = re.search(r'\d+(?:\.\d+){1,3}(?:((a|b|c)\d+)?(?:\.\d+)?(?:\.\dev\d+)?', msg, re.X | re.I)
```

```
# Recommended Fix - Option 2: Add input length validation
if len(msg) < 1000:
    m = re.search(r'\d+(\.\d+)?((a|b|rc)\d+)?(\.post\d+)?(\.dev\d+)?', msg, re.X | re.I)
    #
else:
    m = None
```

14 PERFORMANCE TESTING

This document reports on performance testing performed on the MkDocs project.

14.1 Load Tests

The load tests evaluate the performance of the MkDocs serve command under normal usage. Since the development server is single-threaded and intended for local usage, normal load tests are performed with a low number of users.

- **Sequential Page Load:** Each page is loaded one at a time by a single user. Test duration: 7 minutes.
- **All Pages Loaded at Once:** All pages are loaded simultaneously. Test duration: 1 minute.

```
locust:
  image: locustio/locust:latest
  container_name: locust_load
  working_dir: /mnt/locust
  ports:
    - "8089:8089"
  volumes:
    - ../tests:/mnt/locust/

  command: >
    -f load_test.py
    --host=http://mkdocs:8000
    -u 1
    -t 7m
    --headless
    --html seq_load_test_report.html
    MkDocsUserSeq
```

Attribute	Value
Component Tested	mkdocs serve HTTP server
Tools Used	Locust & Docker Compose container (../locust/locust.py) (Code/performance-testing/locust-load.py)
Endpoints Tested	/: (getting started), /nagata/, /netherman-managed-accounts/, /netherman-creating-a-new-account-using-geth/, /contracts/deploying/, /nastyd-smartcontracts-getting-started/
Test Type	Load Test

All Pages Loaded at Once.

14.1.1 Configuration.

Sequential Page Loads.

Parameter	Value
Users	1
Task Rate	1/min - 2/min
Duration	7 minutes

Parameter	Value
Users	7
Spawn Rate	10,000/sec or 1/sec
Max Duration	~32 seconds

```
locust:
  image: locustio/locust:latest
  container_name: locust_load
  working_dir: /mnt/locust
  ports:
    - "8089:8089"
  volumes:
    - ../tests:/mnt/locust/

  command: >
    -f load_test.py
    --host=http://mkdocs:8000
    -u 7
    -r 10000
    -t 2
    --stop-timeout 30
    --headless
    --html all_load_test_report.html
    MkDocsUserOne
```

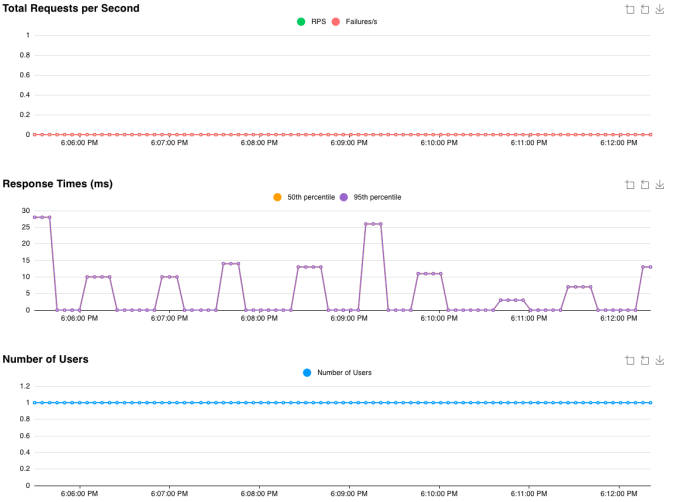
14.1.2 Results.

Sequential Page Loads.

Request Statistics

Type	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
GET	/	2	0	15.58	3	28	47490	0	0
GET	/contracts/deploying/	1	0	25.65	26	26	79446	0	0
GET	/getting-started/	2	0	8.31	7	10	57424	0	0
GET	/ethereum-creating-a-new-account-using-geth/	1	0	13.12	13	13	47902	0	0
GET	/ethereum-managed-accounts/	1	0	14.22	14	14	68660	0	0
GET	/nugets/	2	0	11.39	10	13	66224	0	0
GET	/unity3d-smartcontracts-getting-started/	1	0	11.38	11	11	118370	0	0
Aggregated		10	0	13.49	3	28	65665.4	0.02	0

Charts



All Pages Loaded at Once.

Request Statistics

Type	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
GET	/	1	0	32.81	33	33	47490	24.38	0
GET	/contracts/deploying/	1	0	28.38	28	28	79446	24.38	0
GET	/getting-started/	1	0	30.23	30	30	57424	24.38	0
GET	/ethereum-creating-a-new-account-using-geth/	1	0	29.54	30	30	47902	24.38	0
GET	/ethereum-managed-accounts/	1	0	28.75	29	29	68660	24.38	0
GET	/nugets/	1	0	19.41	19	19	66224	24.38	0
GET	/unity3d-smartcontracts-getting-started/	1	0	27.34	27	27	118370	24.38	0
Aggregated		7	0	28.07	19	33	69359.43	170.64	0

14.1.3 Performance Finding. **Note:** The development server is single-threaded and intended for local use. Tests conducted on a machine with 16 GB RAM, Intel i7-1068NG7 CPU @ 2.30GHz, 4 cores.

Sequential Page Loads.

- Finding: Stable performance under low load, average response time 13.49 ms, 0% failures.

All Pages Loaded at Once.

- Finding: Stable performance, average response time 28.07 ms, 0% failures. Slightly higher than sequential test.

14.2 Stress Test

14.2.1 Scope and Design.

Attribute	Value
Component Tested	Backend - server HTTP server
Tools Used	Locust & Docker Compose container (../locust-test/performance-testing/compose/locust-stress.yaml)
Endpoints Tested	/getting-started/, /nugets/, /ethereum-managed-accounts/, /ethereum-creating-a-new-account-using-geth/, /contracts/deploying/, /unity3d-smartcontracts-getting-started/
Test Type	Stress Test

14.2.2 Configuration.

Parameter	Value
Max Users	2000
Spawn Rate	200/sec
Duration	5 minutes

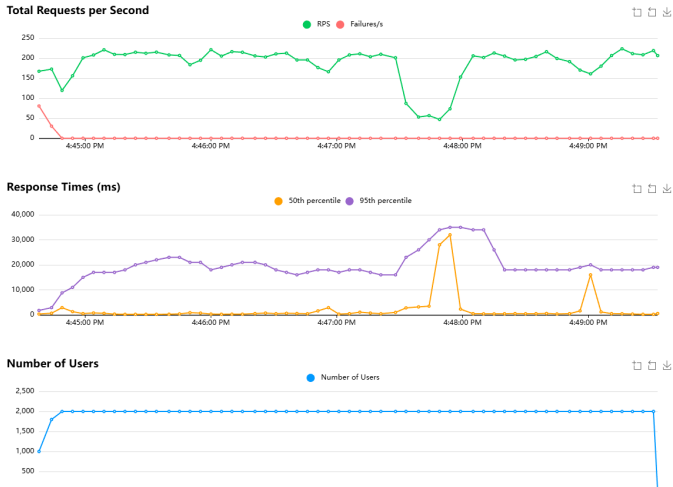
```
services:
  mkdocs:
    build: .
    container_name: mkdocs
    ports:
      - "8000:8000"

  locust:
    image: locustio/locust:latest
    container_name: locust_stress
    working_dir: /mnt/locust
    ports:
      - "8089:8089"
    volumes:
      - ../tests:/mnt/locust/

  command: >
    -f stress_test.py
    --host=http://mkdocs:8000
    -u 2000
    -r 200
    --run-time 5m
    --headless
    --html stress_test_report.html
```

14.2.3 Results.

Charts



Request Statistics

Type	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
GET	/	35906	164	7104.44	1	36137	47273.09	119.41	0.55
GET	/getting-started/	5074	18	7170.22	2	36133	57220.29	16.87	0.06
GET	/nethereum-creating-a-new-account-using-eth/	5120	13	7070.32	2	35523	47780.37	17.03	0.04
GET	/nethereum-managed-accounts/	5035	18	7357.32	2	36136	68414.54	16.74	0.06
GET	/nugets/	5083	29	6920.2	2	35690	65846.17	16.9	0.1
Aggregated		56218	242	7113.98	1	36137	51789.96	186.97	0.8

Response Time Statistics

Method	Name	50%ile (ms)	60%ile (ms)	70%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)	100%ile (ms)
GET	/	520	1600	16000	17000	19000	21000	34000	36000
GET	/getting-started/	520	1600	16000	17000	19000	21000	34000	36000
GET	/nethereum-creating-a-new-account-using-eth/	560	1600	16000	17000	19000	21000	33000	36000
GET	/nethereum-managed-accounts/	560	1900	16000	17000	19000	21000	34000	36000
GET	/nugets/	490	1400	15000	17000	19000	21000	34000	36000

14.2.4 Performance Finding.

- Response times increased with concurrent users, failures occurred.
- Server saturated at 200-220 requests/sec.
- Throughput remained between 165-220 RPS.

14.3 Spike Test

14.3.1 Scope and Design.

Attribute	Value
Component Tested	mkdocs serve HTTP server
Tools Used	Locust & Docker Compose container (../courseProjectCode/performance-testing/compose/locust-spike.yaml)
Endpoint Tested	/ (Homepage)
Test Type	Spike Test

14.3.2 Configuration.

Parameter	Value
Baseline Users	25
Spike Users	125
Spawn Rate	125/sec (instant spike)
Pattern	Baseline → Spike → Recovery

```

services:
  mkdocs:
    build: .
    container_name: mkdocs
    ports:
      - "8000:8000"

  locust:
    image: locustio/locust:latest
    container_name: locust_skipe
    working_dir: /mnt/locust
    ports:
      - "8089:8089"
    volumes:
      - ../tests:/mnt/locust/

    command: >
      -f spike_test.py
      --host=http://mkdocs:8000
      --run-time 3m
      --headless
      --html spike_test_report.html

```

14.3.3 Results.



14.3.4 Performance Finding.

- Response time increased during spike but server remained stable (0% failures)
- 95th percentile response time increased from 5ms to 85ms
- RPS increased from 20 to 80 requests/sec
- Server recovered immediately when load decreased

15 MOST SIGNIFICANT ISSUES FOUND

15.0.1 Files with Low Coverage. Many of the files in mkdocs were well covered by the existing unit tests; however, after further investigation, we discovered that `serve.py` only had a coverage of 20%. We managed to improve this coverage to 83% after adding more unit tests.

15.0.2 ReDoS Vulnerability. During security testing, one of the vulnerabilities we discovered was a regular expression denial of service (ReDoS) vulnerability during version parsing. We documented this issue and how it affects the mkdocs system.

15.0.3 Missing SRI on CDN Scripts. Another major issue discovered during security testing was the missing subresource integrity (SRI) on CDN scripts. An attack could target the supply chain of sites built using mkdocs. We documented this issue and offered some solutions for how it could be mitigated.

15.0.4 Code Quality. While we made an effort to fix some of the issues identified by SonarQube, SonarQube identified 669 maintainability issues that should be reviewed by the mkdocs developers. The presence of these issues means that there could be some code quality concerns that need to be addressed. We documented how to run these issues, so that the developers could run SonarQube and review our results.

16 IMPROVEMENTS MADE TO THE PROJECT

16.0.1 Unit Testing. We increased the existing unit testing coverage from 90.31% to 96% through the unit testing and mocking and stubbing assignments. On one particular file, `serve.py`, the coverage was boosted from 20% to 83%.

16.0.2 Mutation Testing. Through mutation testing, the existing test suite plus our additional tests managed to kill 2708 mutants, while 597 mutants survived.

16.0.3 Static Analysis. During static analysis, we used SonarQube and found 683 issues. Of these 683 issues, we fixed four of them, which actually decreased the number of issues to 669 issues since some of the issues were related and/or similar.

16.0.4 Integration Testing. For integration testing, we added 3 tests to the existing integration tests. The difference between our tests and the existing tests was that ours were automated, and the existing tests seemed to veer more towards system tests rather than integration tests. All of the integration tests pass.

16.0.5 System Testing. For system testing, we added 3 black box tests, all of which pass with no issues.

16.0.6 Security Testing. Sonarqube identified 0 security issues; however, it did identify 10 security hotspots that need to be reviewed. Our team documented 10 of these hotspots and explained how they affect mkdocs.

16.0.7 Performance Testing. We conducted performance testing in the form of load, stress, and spike testing of the mkdocs `serve` command, which spins up a development server that can be viewed locally. The original mkdocs developers did not conduct any performance testing, so our tests were able to identify the limits and

normal performance of this server. In particular, one of the discoveries we made was that the server has a maximum throughput of about 200 requests per second.

17 OVERALL QUALITY ASSESSMENT

Overall, mkdocs has a very high-quality code base. It has a modular architecture with very clear separation of concerns across each module. They have a comprehensive CI/CD pipeline implemented using GitHub Actions, and a very mature and complete test suite with a baseline coverage of just over 90%. The mkdocs community is very active and constantly maintained, with bug fixes and new features being added periodically. However, there are still some security hotspots we identified that require attention, as well as 669 maintainability issues identified by SonarQube that need to be reviewed.

18 LESSONS LEARNED

AJ Barea

Our testing confirms that high code coverage doesn't guarantee strong tests. Mutation testing showed exactly where the testing was superficial.

Connor O'Neill

The capabilities of a tool define the limits of what we can test. For instance, while we use mutmut for mutation testing, the tool does not support mutation testing for generic or abstract classes. There are also some tools, such as scalene, that have bugs that prevented us from properly using the tools.

Kemoy Campbell

Automation testing is more than just writing test scripts. It also requires automating how those tests run, along with the environments and tools they depend on. That's why we containerize everything using Docker. Doing so makes our software quality assurance process more predictable, scalable, and consistent.

19 DATA AND REPOSITORY AVAILABILITIES

All results, reports, and tools used in this report can be found in our repository [2].

20 TEAM MEMBERS AND ROLES

AJ Barea - Test Quality & Coverage Lead

Focused on the effectiveness of the test suite. Primary contributions included analyzing code coverage gaps, implementing mutation testing to identify superficial tests, and ensuring high-quality assertions in unit tests.

Connor O'Neill - Performance & Tooling Lead

Focused on the evaluation and implementation of testing tools. Primary contributions included researching tool capabilities (e.g., mutmut, scalene), managing tool limitations regarding generic/abstract classes, and conducting performance benchmarks.

Kemoy Campbell - Infra Automation, & DevOps Lead

Focused on the project's automation and environment stability. Primary contributions included containerizing the testing environment using Docker to ensure consistent quality assurance processes across different machines and pipelines.

REFERENCES

- [1] Ecosyste.ms. 2025. mkdocs | pypi.org : Ecosyste.ms — Project documentation with Markdown. <https://packages.ecosyste.ms/registries/pypi.org/packages/mkdocs>. Accessed: 2025-12-10.
- [2] kemoycampbell. 2025. mkdocs-AJ_Connor_Kemoy: Project documentation with Markdown. https://github.com/kemoycampbell/mkdocs-AJ_Connor_Kemoy. GitHub repository, accessed on 2025-12-10.
- [3] Landbase. 2025. Companies using MkDocs in 2025. <https://data.landbase.com/technology/mkdocs/>. Accessed: 2025-12-10.
- [4] MkDocs Developers. [n. d.]. mkdocs / mkdocs: Project documentation with Markdown. <https://github.com/mkdocs/mkdocs>. Accessed: 2025-12-10.
- [5] MkDocs Developers. 2025. MkDocs: Project Documentation with Markdown. <https://www.mkdocs.org/>. Accessed: 2025-12-10.