

Docs » Base for Applications Category » Library Skeleton » Code Quality

Code Quality

Code Quality Tools

Anonymous · 08/10/2026

Code Quality Tools

`derafu/library-skeleton` (and `derafu/website-skeleton`, for sites) come with pre-configured code quality tools to ensure consistent code style and identify potential issues early in development. This document explains how to use these tools effectively.

PHP CS Fixer

PHP CS Fixer is a tool that automatically fixes PHP coding standards issues in your code according to rules you define.

Configuration

The configuration is defined in `php-cs-fixer.php` at the root of your project. Key features include:

- PSR-12 coding standards by default.
- Strict type declarations enabled.
- Automatic array syntax conversion to short syntax.
- Ordered imports.
- Modern PHP features optimization (e.g., arrow functions).
- PHPUnit strict assertions.

Usage

Run PHP CS Fixer to check code style issues:

```
vendor/bin/php-cs-fixer fix --dry-run --diff
```

Fix code style issues automatically:

```
vendor/bin/php-cs-fixer fix
```

PHPStan

PHPStan is a static analysis tool that finds bugs in your code without running it. It's focused on finding errors in code logic.

Configuration

The configuration is defined in `phpstan.neon` at the root of your project. By default, it:

- Sets analysis level to 5 (out of 9)
- Analyzes code in the `src` and `tests` directories
- Allows no exceptions: there is no `ignoreErrors` and no `@phpstan-ignore` in the code. If PHPStan reports something, the code is fixed (a type guard, a better type), not silenced.

Levels Explained

- Level 1: Basic checks.
- Level 2: Possibly undefined variables.
- Level 3: Return types, phpdocs.
- Level 4: Type hints.
- Level 5: Basic dead code detection.
- Level 6: Detecting unreachable code.
- Level 7: Union types.
- Level 8: More precise analysis
- Level 9: Mixed type detection

Usage

Run PHPStan analysis:

```
vendor/bin/phpstan analyse
```

PHPUnit

PHPUnit is the standard testing framework for PHP applications.

Configuration

The configuration is defined in `phpunit.xml` at the root of your project. Key features include:

- Test coverage reporting enabled.
- Strict mode enabled: a warning, a notice, a deprecation or a risky test (one that changes the global state or does not assert anything) fails the run, and the run stops at the first failure.

- Every test class says what it covers (`#[CoversClass]`), and a test that covers something else is reported.
- A test is never skipped, excluded or softened to make the run pass: a failing test stays red until the code or the test is fixed.
- Test results output to `var/tests-coverage.txt` and `var/tests-coverage.xml`.
- Test documentation output to `var/tests-testdox.txt`.

Directory Structure

- `tests/src/`: Unit tests for your source code.
- `tests/fixtures/`: Test data and fixtures.

Usage

Run the test suite:

```
vendor/bin/phpunit
```

Generate test coverage reports:

```
vendor/bin/phpunit --coverage-html var/coverage
```

Translations

A package whose exceptions or templates show text to people has a test that audits its messages, with the tools of `Derafu\Translation\Lint` and `Derafu\Twig\Lint` (see [Translation](#)). The test fails when:

- A message has no translation in the Spanish catalogue.
- The catalogue has a message that nothing uses.
- An exception of the package is not translatable.

`derafu/library-skeleton` has the example in `tests/src/Translation/ProjectMessagesTest.php`, and `derafu/website-skeleton` the one of a site in `tests/src/Translation/WebsiteMessagesTest.php`.

Lint Code

Some packages include code whose only purpose is to examine other code and report problems: for example, to find the routes that the templates refer to, or the messages that the code builds to be translated, so a test can check that they exist. That code goes in a namespace named `Lint`, with the package as its prefix: `Derafu\Twig\Lint`, `Derafu\Translation\Lint`, `Derafu\Routing\Lint`, `Derafu\Content\Lint`.

Lint names a role, like `Contract` or `Exception`, and not a subject of the package, so every package can use the same word. The rule is:

- **It is never part of what the package does when it runs.** The runtime code does not call it. It is for tools and tests.
- **It finds, it does not decide.** A lint class extracts facts (what a template refers to) and leaves the verdict to whoever uses it, because whether something is a problem depends on which files belong together, and only the user knows that.
- **It fails loudly.** If it can not read something (a template that does not parse), it raises an error. Returning nothing would look the same as a clean result.

Its tests go in `tests/src/Lint/`.

Integration with CI/CD

These tools are integrated into the CI workflow (`.github/workflows/ci.yml`), which automatically runs on each push to the repository. This ensures that code quality standards are maintained throughout development.

Best Practices

1. **Run tools locally before committing:** This helps catch issues before they enter the codebase.
2. **Gradually increase PHPStan level:** Start with level 5 and work toward higher levels as your project matures.
3. **Aim for high test coverage:** Write tests for all critical code paths.
4. **Update rules as needed:** Customize tool configurations to match your project's specific requirements.

Last updated on 08/10/2026