Package ‘muttest’

May 30, 2025

Type Package

Title Mutation Testing

Version 0.1.0

Description Measure quality of your tests.
'muttest' introduces small changes (mutations) to your code
and runs your tests to check if they catch the changes.
If they do, your tests are good.
If not, your assertions are not specific enough.
'muttest' gives you percent score of how often your tests catch the changes.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.2

Depends R (>= 4.1.0)

Imports checkmate, cli, dplyr, fs, purrr, R6, rlang, testthat, tibble,
treesitter, treesitter.r, withr

Config/testthat/edition 3

URL `https://jakubsob.github.io/muttest/`

Suggests box, covr, cucumber (>= 2.1.0), ggplot2, shiny

Config/Needs/website rmarkdown

NeedsCompilation no

Author Jakub Sobolewski [aut, cre]

Maintainer Jakub Sobolewski <`jakupsob@gmail.com`>

Repository CRAN

Date/Publication 2025-05-30 09:40:05 UTC

Contents

`CopyStrategy` *CopyStrategy interface*

Description

Extend this class to implement a custom copy strategy.

Methods

Public methods:

- `CopyStrategy$execute()`
- `CopyStrategy$clone()`

Method `execute()`**:** Copy project files according to the strategy

Usage:

`CopyStrategy$execute(original_dir)`

Arguments:

`original_dir` The original directory to copy from
`plan` The current test plan

Returns: The path to the temporary directory

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

`CopyStrategy$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

See Also

Other CopyStrategy: `PackageCopyStrategy`, `default_copy_strategy()`

`default_copy_strategy` *Create a default project copy strategy*

Description

Create a default project copy strategy

Usage

```
default_copy_strategy(...)
```

Arguments

`...`	Arguments passed to the `?PackageCopyStrategy` constructor.

Value

A `?CopyStrategy` object

See Also

Other CopyStrategy: `CopyStrategy`, `PackageCopyStrategy`

`default_reporter` *Create a default reporter*

Description

Create a default reporter

Usage

```
default_reporter(...)
```

Arguments

`...`	Arguments passed to the `?ProgressMutationReporter` constructor.

See Also

Other MutationReporter: `MutationReporter`, `ProgressMutationReporter`

`default_test_strategy` *Create a default run strategy*

Description

Create a default run strategy

Usage

```
default_test_strategy(...)
```

Arguments

`...`	Arguments passed to the `?FullTestStrategy` constructor.

Value

A `?TestStrategy` object

See Also

Other TestStrategy: `FileTestStrategy`, `FullTestStrategy`, `TestStrategy`

`FileTestStrategy` *Run tests matching the mutated source file name*

Description

This strategy tells if a mutant is caught by a test matching the source file name.

For example, if the source file name is `foo.R`, and there are test files named `test-foo.R` or `test-bar.R`, only `test-foo.R` will be run.

This strategy should give faster results than `?FullTestStrategy`, especially for big codebases, but the score might be less accurate.

Super class

`muttest::TestStrategy` -> `FileTestStrategy`

Methods

Public methods:

- `FileTestStrategy$new()`
- `FileTestStrategy$execute()`
- `FileTestStrategy$clone()`

Method `new()`**:** Initialize the FileTestStrategy

Usage:

```
FileTestStrategy$new(
  load_helpers = TRUE,
  load_package = c("source", "none", "installed")
)
```

Arguments:

`load_helpers` Whether to load test helpers

`load_package` The package loading strategy

Method `execute()`**:** Execute the test strategy

Usage:

```
FileTestStrategy$execute(path, plan, reporter)
```

Arguments:

`path` The path to the test directory

`plan` The current mutation plan. See `plan()`.

`reporter` The reporter to use for test results

Returns: The test results

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
FileTestStrategy$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other TestStrategy: `FullTestStrategy`, `TestStrategy`, `default_test_strategy()`

`FullTestStrategy` *Run all tests for a mutant*

Description

This test strategy tells if a mutant is caught by any test.

To get faster results, especially for big codebases, use `?FileTestStrategy` instead.

Super class

`muttest::TestStrategy` -> `FullTestStrategy`

Methods

Public methods:

- `FullTestStrategy$new()`
- `FullTestStrategy$execute()`
- `FullTestStrategy$clone()`

Method `new()`**:** Initialize

Usage:

```
FullTestStrategy$new(
  load_helpers = TRUE,
  load_package = c("source", "none", "installed")
)
```

Arguments:

`load_helpers` Whether to load test helpers

`load_package` The package loading strategy

Method `execute()`**:** Execute the test strategy

Usage:

```
FullTestStrategy$execute(path, plan, reporter)
```

Arguments:

`path` The path to the test directory

`plan` The current mutation plan. See `plan()`.

`reporter` The reporter to use for test results

Returns: The test results

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
FullTestStrategy$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other TestStrategy: `FileTestStrategy`, `TestStrategy`, `default_test_strategy()`

`MutationReporter` *Reporter for Mutation Testing*

Description

The job of a mutation reporter is to aggregate and display the results of mutation tests. It tracks each mutation attempt, reporting on whether the tests killed the mutation or the mutation survived.

Public fields

`test_reporter`	Reporter to use for the testthat::test_dir function
`out`	Output destination for reporter messages
`width`	Width of the console in characters
`unicode`	Whether Unicode output is supported
`crayon`	Whether colored output is supported
`rstudio`	Whether running in RStudio
`hyperlinks`	Whether terminal hyperlinks are supported
`current_file`	Path of the file currently being mutated
`current_mutator`	Mutator currently being applied
`plan`	Complete mutation plan for the test run
`results`	List of mutation test results, indexed by file path
`current_score`	Current score of the mutation tests

Methods

Public methods:

- `MutationReporter$new()`
- `MutationReporter$start_reporter()`
- `MutationReporter$start_file()`
- `MutationReporter$start_mutator()`
- `MutationReporter$add_result()`
- `MutationReporter$end_mutator()`
- `MutationReporter$end_file()`
- `MutationReporter$end_reporter()`
- `MutationReporter$get_score()`
- `MutationReporter$cat_tight()`
- `MutationReporter$cat_line()`
- `MutationReporter$rule()`

- `MutationReporter$clone()`

Method `new()`**:** Initialize a new reporter

Usage:

```
MutationReporter$new(test_reporter = "silent", file = stdout())
```

Arguments:

`test_reporter` Reporter to use for the testthat::test_dir function

`file` Output destination (default: stdout)

Method `start_reporter()`**:** Start reporter

Usage:

```
MutationReporter$start_reporter(plan = NULL)
```

Arguments:

`plan` The complete mutation plan

`temp_dir` Path to the temporary directory for testing

Method `start_file()`**:** Start testing a file

Usage:

```
MutationReporter$start_file(filename)
```

Arguments:

`filename` Path to the file being mutated

Method `start_mutator()`**:** Start testing with a specific mutator

Usage:

```
MutationReporter$start_mutator(mutator)
```

Arguments:

`mutator` The mutator being applied

Method `add_result()`**:** Add a mutation test result

Usage:

```
MutationReporter$add_result(plan, killed, survived, errors)
```

Arguments:

`plan` Current testing plan. See `plan()`.

`killed` Whether the mutation was killed by tests

`survived` Number of survived mutations

`errors` Number of errors encountered

Method `end_mutator()`**:** End testing with current mutator

Usage:

```
MutationReporter$end_mutator()
```

Method `end_file()`**:** End testing current file

Usage:

```
MutationReporter$end_file()
```

Method `end_reporter()`**:** End reporter and show summary

Usage:

```
MutationReporter$end_reporter()
```

Method `get_score()`**:** Get the current score

Usage:

```
MutationReporter$get_score()
```

Method `cat_tight()`**:** Print a message to the output

Usage:

```
MutationReporter$cat_tight(...)
```

Arguments:

`...` Message to print

Method `cat_line()`**:** Print a message to the output

Usage:

```
MutationReporter$cat_line(...)
```

Arguments:

`...` Message to print

Method `rule()`**:** Print a message to the output with a rule

Usage:

```
MutationReporter$rule(...)
```

Arguments:

`...` Message to print

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
MutationReporter$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other MutationReporter: `ProgressMutationReporter`, `default_reporter()`

muttest — *Run a mutation test*

Description

Run a mutation test

Usage

```
muttest(
  plan,
  path = "tests/testthat",
  reporter = default_reporter(),
  test_strategy = default_test_strategy(),
  copy_strategy = default_copy_strategy()
)
```

Arguments

`plan`	A data frame with the test plan. See `plan()`.
`path`	Path to the test directory.
`reporter`	Reporter to use for mutation testing results. See `?MutationReporter`.
`test_strategy`	Strategy for running tests. See `?TestStrategy`. The purpose of test strategy is to control how tests are executed. We can run all tests for each mutant, or only tests that are relevant to the mutant.
`copy_strategy`	Strategy for copying the project. See `?CopyStrategy`. This strategy controls which files are copied to the temporary directory, where the tests are run.

Value

A numeric value representing the mutation score.

operator — *Mutate an operator*

Description

It changes a binary operator to another one.

Usage

```
operator(from, to)
```

Arguments

`from`	The operator to be replaced.
`to`	The operator to replace with.

Examples

```
operator("==", "!=")
operator(">", "<")
operator("<", ">")
operator("+", "-")
```

`PackageCopyStrategy` *Package copy strategy*

Description

It copies all files and directories from the original directory to a temporary directory.

Super class

`muttest::CopyStrategy` -> `PackageCopyStrategy`

Methods

Public methods:

- `PackageCopyStrategy$execute()`
- `PackageCopyStrategy$clone()`

Method `execute()`**:** Copy project files, excluding hidden and temp directories

Usage:

```
PackageCopyStrategy$execute(original_dir, plan)
```

Arguments:

`original_dir` The original directory to copy from

`plan` The current test plan

Returns: The path to the temporary directory

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PackageCopyStrategy$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other CopyStrategy: `CopyStrategy`, `default_copy_strategy()`

`plan` *Create a plan for mutation testing*

Description

Each mutant requires rerunning the tests. For large project it might be not feasible to test all mutants in one go. This function allows you to create a plan for selected source files and mutators.

Usage

```
plan(mutators, source_files = fs::dir_ls("R", regexp = ".[rR]$"))
```

Arguments

`mutators`	A list of mutators to use. See `operator()`.
`source_files`	A vector of file paths to the source files.

Details

The plan is in a data frame format, where each row represents a mutant.

You can subset the plan before passing it to the `muttest()` function.

Value

A data frame with the test plan. The data frame has the following columns:

- `filename`: The name of the source file.
- `original_code`: The original code of the source file.
- `mutated_code`: The mutated code of the source file.
- `mutator`: The mutator that was applied.

`ProgressMutationReporter` *Progress Reporter for Mutation Testing*

Description

A reporter that displays a progress indicator for mutation tests. It provides real-time feedback on which mutants are being tested and whether they were killed by tests.

Super class

`muttest::MutationReporter` -> `ProgressMutationReporter`

Public fields

`start_time` Time when testing started (for duration calculation)

`min_time` Minimum test duration to display timing information

`col_config` List of column configuration for report formatting

Methods

Public methods:

- `ProgressMutationReporter$format_column()`
- `ProgressMutationReporter$fmt_h()`
- `ProgressMutationReporter$fmt_r()`
- `ProgressMutationReporter$new()`
- `ProgressMutationReporter$start_reporter()`
- `ProgressMutationReporter$add_result()`
- `ProgressMutationReporter$update()`
- `ProgressMutationReporter$end_file()`
- `ProgressMutationReporter$cr()`
- `ProgressMutationReporter$end_reporter()`
- `ProgressMutationReporter$clone()`

Method `format_column()`**:** Format a column with specified padding and width

Usage:

```
ProgressMutationReporter$format_column(text, col_name, colorize = NULL)
```

Arguments:

`text` Text to format

`col_name` Column name to use configuration from

`colorize` Optional function to color the text

Method `fmt_h()`**:** Format the header of the report

Usage:

```
ProgressMutationReporter$fmt_h()
```

Method `fmt_r()`**:** Format a row of the report

Usage:

```
ProgressMutationReporter$fmt_r(status, k, s, e, t, score, mutator, file)
```

Arguments:

`status` Status symbol (e.g., tick or cross)

`k` Number of killed mutations

`s` Number of survived mutations

`e` Number of errors

`t` Total number of mutations

`score` Score percentage

`mutator` The mutator used

`file` The file being tested

Returns: Formatted row string

Method `new()`**:** Initialize a new progress reporter

Usage:

```
ProgressMutationReporter$new(
  test_reporter = "silent",
  min_time = 1,
  file = stdout()
)
```

Arguments:

`test_reporter` Reporter to use for testthat::test_dir

`min_time` Minimum time to show elapsed time (default: 1s)

`file` Output destination (default: stdout)

Method `start_reporter()`**:** Start reporter

Usage:

```
ProgressMutationReporter$start_reporter(plan = NULL)
```

Arguments:

`plan` The complete mutation plan

Method `add_result()`**:** Add a mutation test result

Usage:

```
ProgressMutationReporter$add_result(plan, killed, survived, errors)
```

Arguments:

`plan` Current testing plan. See `plan()`.

`killed` Whether the mutation was killed by tests

`survived` Number of survived mutations

`errors` Number of errors encountered

Method `update()`**:** Update status spinner (for long-running operations)

Usage:

```
ProgressMutationReporter$update(force = FALSE)
```

Arguments:

`force` Force update even if interval hasn't elapsed

Method `end_file()`**:** End testing current file

Usage:

```
ProgressMutationReporter$end_file()
```

Method `cr()`**:** Carriage return if dynamic, newline otherwise

Usage:

`ProgressMutationReporter$cr()`

Method `end_reporter()`**:** End reporter with detailed summary

Usage:

`ProgressMutationReporter$end_reporter()`

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

`ProgressMutationReporter$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

See Also

Other MutationReporter: `MutationReporter`, `default_reporter()`

`TestStrategy` *TestStrategy interface*

Description

Extend this class to implement a custom test strategy.

Methods

Public methods:

- `TestStrategy$execute()`
- `TestStrategy$clone()`

Method `execute()`**:** Execute the test strategy

Usage:

`TestStrategy$execute(path, plan, reporter)`

Arguments:

`path` The path to the test directory

`plan` The current mutation plan. See `plan()`.

`reporter` The reporter to use for test results

Returns: The test result

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

`TestStrategy$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

See Also

Other TestStrategy: `FileTestStrategy`, `FullTestStrategy`, `default_test_strategy()`

Index