

Package ‘wyz.code.metaTesting’

September 25, 2023

Type Package

Title Wizardry Code Meta Testing

Version 1.1.22

Author Fabien Gelineau <neonira@gmail.com>

Maintainer Fabien Gelineau <neonira@gmail.com>

Description Meta testing is the ability to test a function without having to provide its parameter values.

Those values will be generated, based on semantic naming of parameters, as introduced by package 'wyz.code.offensiveProgramming'.

Value generation logic can be completed with your own data types and generation schemes. This to meet your most specific requirements and to answer to a wide variety of usages, from general use case to very specific ones.

While using meta testing, it becomes easier to generate stress test campaigns, non-regression test campaigns and robustness test campaigns, as generated tests can be saved and reused from session to session.

Main benefits of using 'wyz.code.metaTesting' is ability to discover valid and invalid function parameter combinations, ability to infer valid parameter values, and to provide smart summaries that allows you to focus on dysfunctional cases.

Encoding UTF-8

License GPL-3

Depends R (>= 4.0)

Imports methods, data.table (>= 1.11.8), tidyr,
wyz.code.offensiveProgramming (>= 1.1.22), crayon, utils, stats

Suggests testthat, knitr, rmarkdown

RoxygenNote 6.1.1

VignetteBuilder knitr

URL https://neonira.github.io/offensiveProgrammingBook_v1.2.2/

NeedsCompilation no

Repository CRAN

Date/Publication 2023-09-25 07:10:02 UTC

R topics documented:

buildSemanticArgumentName	2
computeArgumentsCombination	3
exploreSignatures	4
generateData	6
opMetaTestingInformation	7
opwf	8
qualifyFunctionArguments	9
retrieveDataFactory	10
setDefaultArgumentsGenerationContext	11
setGenerationContext	12
testFunction	13
usesSemanticArgumentNames	14
Index	16

buildSemanticArgumentName

Build semantic argument name

Description

Build a semantic argument name from the suffix you provide.

Usage

```
buildSemanticArgumentName(suffix_s_1, variableName_s_1 = "x_")
```

Arguments

suffix_s_1 one string to be used as a suffix. Use retrieveDataFactory()\$getKnownSuffixes() to get a vector of known suffixes.

variableName_s_1 a string that is the variable name you want to use.

Details

Know that no checks are done on suffix_s_1. Value you provide will be trusted, regular or irregular one.

Value

A single string that is the argument name build from your variableName_s_1 and suffix_s_1 values.

Author(s)

Fabien Gelineau <neonira@gmail.com>

Maintainer: Fabien Gelineau <neonira@gmail.com>

See Also

Refer to [testFunction](#)

Examples

```
# typical example
buildSemanticArgumentName('i') # x_i

buildSemanticArgumentName('ui_1', 'numberOfItems') # numberOfItems_ui_1
```

computeArgumentsCombination

Compute Function Arguments Combination

Description

Computes a priori legal combinations of function arguments, according to the function definition (see [formals](#)).

Usage

```
computeArgumentsCombination(fun_f_1)
```

Arguments

fun_f_1 an R function

Details

Computes an a priori legal list of argument signatures for the provided function.

Allows to foresee test complexity for a function, as this is in narrow relationship, with the number of various call signatures that should be tested. The number of signatures is in itself a good indicator of complexity.

Value

A list containing following named list

names	names of mandatory arguments, ellipsis (...) arguments and of default arguments.
number	The number provides the number of replacements per argument.
signatures	The signatures are the resulting textual argument combinations.

Author(s)

Fabien Gelineau <neonira@gmail.com>

Maintainer: Fabien Gelineau <neonira@gmail.com>

See Also

Refer to [testFunction](#)

Examples

```
# typical example
computeArgumentsCombination(append)

computeArgumentsCombination(kronecker)
```

exploreSignatures *Explore Signatures*

Description

Test an offensive programming wrapper function, applying various argument signatures.

Usage

```
exploreSignatures(fun_f_1,
                  argumentsTypeRestrictions_l = list(),
                  signaturesRestrictions_l = list())
```

Arguments

`fun_f_1` a single R function. Must be an offensive programming wrapper function.
See [opwf](#).

`argumentsTypeRestrictions_l`
a named list. Each name must match a function argument name. Each content must be a vector of strings, each of them matching a `retrieveDataFactory()$getKnownSuffixes()` known suffix.

`signaturesRestrictions_l`
an unnamed list of single strings, each of them matching one of `computeArgumentsCombination(fun_`

Details

This function offers a really convenient way to test your own functions, without the burden of building the execution context, that is much trickier than one can imagine at first glance.

Moreover it provides argument signature analysis, which is not provided by [testFunction](#).

Arguments restriction parameter `argumentsTypeRestrictions_l` allows to restrict on demand, value types exploration. It is very useful and convenient to reduce the exploration tree, and to shorten execution time.

By default, a total of 768 tests will run for a single function, when no `signaturesRestrictions_1` is set. This may require some time to achieve.

When working interactively, a good practice is to use `computeArgumentsCombination` prior to use function `computeArgumentsCombination`, as it will provide complexity information about the function you wish to test. The number of signature is a good metric of function call complexity. Know that each of them will be tested, and data generation has to be achieved for each parameter according to global or restricted scheme, depending on your `argumentsTypeRestrictions_1` inputs.

Value

A list with names `info`, `success`, `failure`, each of them being a **list**.

The `info` sub list holds execution results. It holds following entries

- `raw` is a list, providing capture of execution context, data and results.
- `good` is a list, providing same information as `raw`, filtered to retain only tests that do not generate any error.
- `bad` is a list, providing same information as `raw`, filtered to retain only tests that do generate error.

The `success` sub list holds analysis results for tests which do not generate errors. It holds following entries

- `code` is a `data.table`, providing used call code and results.
- `table` is a `data.table`, providing used argument signatures and execution context information.
- `synthesis` is a list, providing synthesis information. Much easier to read, than `table` entry.

The `failure` sub list holds analysis results for tests which do generate errors. It holds following entries

- `table` is a `data.table`, providing encountered error messages and execution context information
- `synthesis` is a list, providing synthesis information. Much easier to read, than `table` entry.

Author(s)

Fabien Gelineau <neonira@gmail.com>

Maintainer: Fabien Gelineau <neonira@gmail.com>

See Also

Refer to `testFunction` and to `generateData`.

Examples

```

# typical use case
op_sum <- opwf(sum, c('...', 'removeNA_b_1'))

rv_sum <- exploreSignatures(op_sum, list(... = c('im', 'r', 'cm')))

# which are the errors of exploration and in what context do they occur?
print(rv_sum$failure$synthesis)

# which are the good behaviors of exploration and in what context do they occur?
print(rv_sum$success$synthesis)

# Restrict signatures to use for exploration testing on op_sum
# Consider only two cases: no argument and ellipsis1_, ellipsis2_
cac_sum <- computeArgumentsCombination(op_sum)
rv_sum_f <- exploreSignatures(op_sum, list(... = c('im', 'r', 'cm')),
                                cac_sum$signatures[c(1, 5)])

```

generateData

Generate Data

Description

Function to generate data.

Usage

```

generateData(function_f_1,
             argumentsTypeRestrictions_l = list(),
             replacementContext_l = setGenerationContext(),
             ellipsisReplacementContext_l = setGenerationContext(),
             defaultArgumentsContext_l = setDefaultArgumentsGenerationContext(),
             functionName_s_1 = deparse(substitute(function_f_1))
)

```

Arguments

function_f_1 a single R function, offensive programming ready, therefore using semantic argument names

argumentsTypeRestrictions_l a named list. Each name must match a function argument name. Each content must be a vector of strings, each of them matching a `retrieveDataFactory()$getKnownSuffixes()` known suffix.

replacementContext_l a generation context object, as defined by `setGenerationContext` function, applicable to standard arguments of the function, if any.

ellipsisReplacementContext_1
 an ellipsis replacement context object, as defined by [setGenerationContext](#) function, applicable to ... arguments of the function.

defaultArgumentsContext_1
 a default argument context object, as defined by [setDefaultArgumentsGenerationContext](#) function, applicable to default arguments of the function.

functionName_s_1
 A character vector of length 1, holding the function name. Particularly useful in R scripts.

Details

Generate a driven aleatory set of data to be used as argument in a call to function `fun_f_1`. Generation is driven by the `argumentsTypeRestrictions_1` argument.

Value

A object with following names

<code>generation</code>	argument name generation
<code>codedata</code>	the generated data
<code>context</code>	data type generation context
<code>n</code>	number of first level data generations

See Also

Refer to [coderetrieveDataFactory](#) and to [testFunction](#).

Examples

```
# typical example
op_sum <- opwf(sum, c('...', 'removeNA_b_1'))
op_sum_atr <- list('...' = c('i', 'd', 'c'))
ec <- setGenerationContext(0, TRUE, FALSE)
gd <- generateData(op_sum, op_sum_atr, ec, erc$hetero_vector[[1]], dac$none)
```

opMetaTestingInformation

Package functions information

Description

A reminder of available functions from this package, and, most common usage intent. A poor man CLI cheat sheet.

Usage

```
opMetaTestingInformation()
```

Value

See [opInformation](#) value description.

Author(s)

Fabien Gelineau <neonira@gmail.com>

Maintainer: Fabien Gelineau <neonira@gmail.com>

See Also

Refer also to package vignettes.

Examples

```
##---- typical case ----
opMetaTestingInformation()
```

 opwf

Offensive Programming Wrap Function

Description

Create an offensive programming function, wrapping a standard R function.

Usage

```
opwf(fun_f_1, parameterNames_s, functionName_s_1 = NA_character_)
```

Arguments

`fun_f_1` a single R function

`parameterNames_s`

the new names of the parameter function, must be semantic argument names.
Must be a bijection to actual `fun_f_1` argument names.

`functionName_s_1`

A string holding the function name. Default value, implies evaluation using `deparse(substitute(fun_f_1))`

Details

If any arguments default values are present, they are managed transparently and should be correctly and automatically substituted.

Value

A R function which takes given `parameterNames_s` as arguments.

Author(s)

Fabien Gelineau <neonira@gmail.com>

Maintainer: Fabien Gelineau <neonira@gmail.com>

See Also

Refer to [testFunction](#)

Examples

```
# typical example
op_sum <- opwf(sum, c('...', 'removeNA_b_1'))

# example with substituted argument in existing default valued arguments
op_append <- opwf(append, c('originalValues_', 'valuesToInsert_', 'afterIndex_ui_1'))
```

qualifyFunctionArguments

Qualify function arguments.

Description

Retrieve information about function arguments.

Usage

```
qualifyFunctionArguments(fun_f_1)
```

Arguments

`fun_f_1` A single function, not a string.

Value

A emphlist with following names

<code>argument_names</code>	a character vector of all the function argument names
<code>owns_ellipsis</code>	a boolean. Is TRUE when ... belongs to argument names
<code>symbol_names</code>	a character vector of argument names that are symbols
<code>symbol_indexes</code>	the integer indexes of symbol names in the argument names
<code>stripped_symbol_names</code>	a character vector of argument names that are symbols, not considering ...
<code>stripped_symbol_indexes</code>	the integer indexes of stripped symbol names in the argument names
<code>default_names</code>	a character vector of argument names that owns default values

default_indexes the integer indexes of default valued arguments names in the argument names
codearguments a pairList of argument names and values. Refer to [formals](#) for more information

Author(s)

Fabien Gelineau <neonira@gmail.com>

Maintainer: Fabien Gelineau <neonira@gmail.com>

Examples

```
# typical examples  
qualifyFunctionArguments(Sys.Date)  
qualifyFunctionArguments(cos)  
qualifyFunctionArguments(sum)
```

retrieveDataFactory *Retrieve Data Factory*

Description

As the data factory may be modified, this function allows you to make changes and to record them in your own specialized data generation factory, to match various needs and ease reuse.

Usage

```
retrieveDataFactory()
```

Details

Provides a data factory.

Retrieves a [retrieveDataFactory](#) from options variable `op_mt_data_factory`.

Allow to customize data factory entries.

Value

An R object that is a [retrieveDataFactory](#) .

Author(s)

Fabien Gelineau <neonira@gmail.com>

Maintainer: Fabien Gelineau <neonira@gmail.com>

Examples

```
##---- typical case ----

draw_integer_array_dim2 <- function(n, replace_b_1 = TRUE) {
  m <- n + sample(0:3, 1)
  matrix(seq(1, n * m), byrow = TRUE, nrow = n,
         dimnames = list(paste('row_', 1:n), paste('col_', 1:m)))
}

df <- retrieveDataFactory()
df$addSuffix('a', "array", draw_integer_array_dim2)

options(op_mt_data_factory = df)
fg <- retrieveDataFactory() # retrieves the user defined data factory
fg$getRecordedTypes()[suffix == 'a'] # right behavior !

# wrong behavior as retrieveDataFactory will provide the default factory and not yours!
options(op_mt_data_factory = NULL)
fh <- retrieveDataFactory() # retrieves the default factory
fh$getRecordedTypes()[suffix == 'a']
```

```
setDefaultArgumentsGenerationContext
```

Set default arguments generation context.

Description

Set default arguments generation context

Usage

```
setDefaultArgumentsGenerationContext(useDefaultArguments_b_1 = TRUE,
                                     useAllDefaultArguments_b_1 = FALSE)
```

Arguments

```
useDefaultArguments_b_1
  a single boolean value to specify the usage of default arguments in generated
  function call
useAllDefaultArguments_b_1
  A single boolean value to specify usage of all default valued arguments in gen-
  erated function call. Second argument is considered only when first argument is
  TRUE.
```

Value

A list holding the provided values, allowing easy reuse either interactively or programmatically, accessible through names `use`, and `use_all`.

Predefined variables named `default_arguments_context` and `dac` hold most common definition cases. Very helpful as it simplifies reuses and reduces code length.

Author(s)

Fabien Gelineau <neonira@gmail.com>

Maintainer: Fabien Gelineau <neonira@gmail.com>

Examples

```
# a typical instantiation
mydgc <- list(
  setDefaultArgumentsGenerationContext(FALSE, FALSE),
  setDefaultArgumentsGenerationContext(TRUE, FALSE),
  setDefaultArgumentsGenerationContext(TRUE, TRUE)
)

# uses predefined variable
print(dac$partial)
```

setGenerationContext *Set generation context.*

Description

Use this function to set a generation context

Usage

```
setGenerationContext(replacementNumber_ui_1 = sample(0:3L, 1),
  homogeneousTypeReplacement_b_1 = FALSE,
  allowList_b_1 = TRUE,
  forceList_b_1 = FALSE)
```

Arguments

replacementNumber_ui_1 a single positive integer expressing the number of arguments to generate.

homogeneousTypeReplacement_b_1 A single boolean expressing willingness to replace chosen argument with same type arguments, or not. Useful when dealing with ...

allowList_b_1 a single boolean , expressing the desired result. When TRUE result is a list, a vector otherwise.

forceList_b_1 a single boolean , expressing the desire to get the result as a list.

Value

A list containing all the provided arguments, accessible through names homogeneous_type, number_replacements, and allow_list.

Predefined variables named established_replacement_context and erc hold most common definition cases. Very helpfull as it simplifies reuses and reduces code length.

Author(s)

Fabien Gelineau <neonira@gmail.com>

Maintainer: Fabien Gelineau <neonira@gmail.com>

Examples

```
# a typical instantiation
egc <- list(
  setGenerationContext(homogeneous = TRUE),
  setGenerationContext(allowList = FALSE)
)

# uses predefined variable
print(erc$homo_vector[[2]])
```

testFunction	<i>Test function</i>
--------------	----------------------

Description

Apply data to function signature and record results.

Usage

```
testFunction(function_f_1,
             generatedData_1,
             functionName_s_1 = deparse(substitute(function_f_1)))
```

Arguments

`function_f_1` a single R function, offensive programming ready, with using semantic argument names

`generatedData_1` data to apply to the function. Could be generated by [generateData](#) function is desired.

`functionName_s_1` A string that is the function name. Particularly useful, in scripts.

Details

Executes code and captures execution context and result, for posterior analysis.

Value

A list with following names

generation	argument name generation
data	generated data
context	data type generation context
n	number of first level data generated

Generated data are ready for use and accessible using the data name of the list.

See Also

Refer to [opwf](#).

Examples

```
# typical example
op_sum <- opwf(sum, c('...', 'removeNA_b_1'))
op_sum_atr <- list('...' = c('i', 'd', 'c'))
ec <- setGenerationContext(0, TRUE, FALSE)
gd <- generateData(op_sum, op_sum_atr, ec, erc$hetero_vector[[1]], dac$none)
tf <- testFunction(op_sum, gd$data)
```

usesSemanticArgumentNames

Uses semantic argument names.

Description

Determine if the given function uses semantic argument names.

Usage

```
usesSemanticArgumentNames(fun_f_1)
```

Arguments

fun_f_1 A single function

Value

A TRUE when arguments used by function are all semantic names.

Author(s)

Fabien Gelineau <neonira@gmail.com>

Maintainer: Fabien Gelineau <neonira@gmail.com>

Examples

```
f <- function(x_) x_  
  
usesSemanticArgumentNames(f)  
# TRUE  
  
usesSemanticArgumentNames(sum)  
# FALSE
```

Index

- * **code evaluation mode**
 - retrieveDataFactory, 10
- * **data generation**
 - generateData, 6
 - setDefaultArgumentsGenerationContext, 11
 - setGenerationContext, 12
- * **meta testing**
 - buildSemanticArgumentName, 2
 - computeArgumentsCombination, 3
 - exploreSignatures, 4
 - generateData, 6
 - opMetaTestingInformation, 7
 - opwf, 8
 - qualifyFunctionArguments, 9
 - setDefaultArgumentsGenerationContext, 11
 - setGenerationContext, 12
 - testFunction, 13
 - usesSemanticArgumentNames, 14
- * **programation**
 - exploreSignatures, 4
 - generateData, 6
 - qualifyFunctionArguments, 9
 - setDefaultArgumentsGenerationContext, 11
 - setGenerationContext, 12
 - usesSemanticArgumentNames, 14
- * **programming**
 - buildSemanticArgumentName, 2
 - computeArgumentsCombination, 3
 - opMetaTestingInformation, 7
 - opwf, 8
 - retrieveDataFactory, 10
 - testFunction, 13
- * **utilities**
 - buildSemanticArgumentName, 2
 - computeArgumentsCombination, 3
 - exploreSignatures, 4
 - generateData, 6
 - opMetaTestingInformation, 7
 - opwf, 8
 - qualifyFunctionArguments, 9
 - retrieveDataFactory, 10
 - setDefaultArgumentsGenerationContext, 11
 - setGenerationContext, 12
 - testFunction, 13
 - usesSemanticArgumentNames, 14
- buildSemanticArgumentName, 2
- computeArgumentsCombination, 3, 5
- dac
 - (setDefaultArgumentsGenerationContext), 11
- default_arguments_context
 - (setDefaultArgumentsGenerationContext), 11
- erc (setGenerationContext), 12
- established_replacement_context
 - (setGenerationContext), 12
- exploreSignatures, 4
- formals, 3, 10
- generateData, 5, 6, 13
- offensiveProgrammingWrapFunction
 - (opwf), 8
- opInformation, 8
- opMetaTestingInformation, 7
- opwf, 4, 8, 14
- qualifyFunctionArguments, 9
- retrieveDataFactory, 7, 10, 10

setDefaultArgumentsGenerationContext,
7, 11
setGenerationContext, 6, 7, 12
testFunction, 3–5, 7, 9, 13
usesSemanticArgumentNames, 14