Embedding Constructural Documentation in Unit Tests

Mathieu Nassif

School of Computer Science

McGill University, Montreal

August 2018

A thesis submitted to McGill University

in partial fulfillment of the requirements of the degree of

Master of Science

Abstract

Software projects capture information in artifacts that include production code, test suites, and documentation. Because different artifacts serve different purposes, the information they encode can be redundant. We present an approach to mitigate this redundancy by allowing developers to encode in unit tests information that is used to automatically generate documentation fragments. The result is a set of documentation fragments injected in the reference documentation of the methods under test. We implemented this approach in the form of an Eclipse plug-in. The ability to regenerate the documentation fragments allows documentation to be automatically updated when unit tests are modified. This approach is part of the larger idea of embedding information in source code through the use of meaningful constructs, a concept we termed constructural documentation. We report on a multi-case study that provides insights on the pertinence of constructural information in real world systems. This study highlighted several practices that favor or prevent embedding constructural information in tests, such as the use of standardized local variable names to encode recurring concerns of a test, thus acting as anchors for constructural documentation. Combined with the detailed description of a working tool, the study shows constructural documentation to be a promising way to increase automation in software documentation practices.

Résumé

L'exercice de programmation nécessite de la part des développeurs d'encoder beaucoup d'information. Cependant, il est commun dans des projets de développement de logiciel de recoder cette information dans un format différent pour documenter le code, dans l'intérêt des lecteurs humains. De plus, d'autres artéfacts de code, comme les tests unitaires, requièrent que les développeur encode à nouveau la même information dans un format différent.

Dans cette thèse, nous présentons une approche exploitant ce parallèle entre l'information encodée dans les tests unitaires et la documentation, dans le but de générer automatiquement la documentation. Cette approche génère des fragments de documentation, insérés directement dans le code de production, dans les commentaires d'en-tête de chaque méthode testée. La possibilité de régénérer facilement ces fragments de documentation lorsque les tests unitaires sont modifiés permet à la documentation d'être constamment mise à jour. Cette approche fait partie de l'idée plus générale d'intégrer de l'information dans du code source grâce à des structures sémantiques. Nous appelons ce concept «information constructurale».

Nous présentons une étude de cas ayant pour but d'évaluer la pertinence de l'information constructurale dans des systèmes logiciels réels. Cette étude a permis de mettre en valeur certaines pratiques favorisant or entravant l'incorporation d'information constructurale dans les tests unitaires, comme l'utilisation de noms de variable génériques, plutôt que spécifiques au contexte du test, permettant ainsi d'encoder de façon constante les éléments récurrents des tests unitaires, favorisant donc l'information constructurale.

Acknowledgments

I express my dearest gratitude towards Prof. Martin Robillard, who guided me through the intricate paths of the universe of scientific research. His advice helped me through all the development of this project, from the original conception of the research objective to the last sentence written in this thesis. His commitment to my research was an invaluable asset in the design and application of a sound scientific methodology.

I thank the members of the SWEVO group, for creating a friendly atmosphere, both inside and outside the lab. I also thank my family for their unconditional love, even through the worst periods.

Finally, I am most grateful to Ariane, for always being at my side through these years, and supporting me when I need it the most.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Developing a large scale software system requires more than just writing production code: it is also common practice to provide testing code and documentation. Writing these artifacts can represent a significant effort for developers.

Testing code can be composed of different kinds of tests, from performance tests to manual interface interaction tests. In particular, unit tests test the execution of a single unit of code, typically a method, in isolation for a specific case. To ease the creation of unit tests and their integration in the development cycle, several frameworks have been created. Among them, some of the most widely adopted are part of the xUnit family of frameworks, with JUnit [1] being the implementation for Java systems. Such frameworks offer support to write and run the tests and report the results in a convenient manner, allowing developers to quickly discover failing tests, which represent potential errors. JUnit is currently at its fifth version, however earlier versions, which used a different syntax, are still being used today.

Similarly, the documentation in a software system can take various forms, from architectural design models in separate files to comments inserted in the source code. In this thesis, we focus on documentation comments inserted above the declaration of a method, for systems programmed in Java. They are delimited by `/**` and `*/`, and contain a mix of text

and tag-delimited blocks for specific information. These comments are not only visible to developers reading the source file, but they are also automatically transformed into HTML documentation by the Javadoc tool [2].

In addition to requiring additional effort and time, writing documentation and testing code often consists of encoding information that is already encoded in the production code, but in a different format. For example, the method `ArrayList.indexOf(Object)`, from OpenJDK version 10, returns -1 if its argument is not found. The test suite checks scenarios where `indexOf` is called with arguments not in the list, and expects it to return -1. This special value is also mentioned in the documentation. Hence, for a single specification, developers encoded the information three times, in three different formats.

Furthermore, manually written documentation suffers from several problems, such as high proportions of outdated [3] or missing [4] documentation in some systems. To address the problem of missing documentation, some organizations impose strict guidelines, such as the requirement to document every public method [5,6]. However, this approach can lead to excessive and non informative documentation [7]. Furthermore, because documentation is not tied to the production code as are unit tests, and because recovering such traceability links is a non trivial task [8–11], it is hard to automatically determine where documentation is lacking or should be updated [12]. This leads to some developers expressing their concerns over the usefulness of spending too much effort on documentation, such as the Agile Programming community, which values "working software over comprehensive documentation" [13].

To alleviate the burden of writing repetitive documentation, and to encourage developers to write unit tests of high quality, we propose an approach to generate documentation based on well-defined constructs in the coding of unit tests. This approach both provides a framework for developers to actively embed information by using these specific constructs, and supports the generation of human-readable documentation. Thus, it integrates writing tests and documentation into a single, streamlined approach: a combination we call *constructural*

documentation. Because this type of documentation can be generated seamlessly, and therefore updated at minimal cost, and because it is backed by passing unit tests, constructural documentation can reduce the amount of outdated or incorrect documentation.

This approach is part of the larger idea that a secondary layer of information can exist in programming constructs, due to the explicit and shared agreement on the meaning of particular arrangements in the source code. For example, organizations' conventions [5,14–16] and design patterns [17] crystallize specific constructs to express common ideas. We introduce the term *constructural information* to refer to this kind of information. For example, early versions of JUnit used the prefix `test` to detect unit tests. Thus, this prefix captured more information than the pure semantic enforced by a compiler.

The approach is supported by a tool, called DScribe, whose implementation explores some of the trade-offs between the applicability and usability of the approach.

1.1 Motivating Example

Consider the `get` method of Google Guava [18]'s `Optional` class. This class represents an object that may or may not encapsulate a reference: the reference is either *present* or *absent*. It comes with the guarantee that if the `Optional` object contains a reference, this reference is not `null`.

The associated testing class `OptionalTest` contains the unit test `testGet_absent`, shown in Figure 1.1, which tests the method `get`.

This unit test contains elements of information that are typical to many tests: the last segment of the test name, `absent`, identifies the state of the instance under test, and the `try-catch` statement is a common testing pattern to ensure that a method throws an exception of a type that is assignable to the type declared in the `catch` clause.

Reading this test, it is clear that when `get` is called on an `Optional` instance that is

```
1 public void testGet_absent() {
2   Optional<String> optional = Optional.absent();
3   try {
4     optional.get();
5     fail();
6   } catch (IllegalStateException expected) {
7   }
8 }
```

Figure 1.1: Example of a unit test from Google Guava

```
1  /**
2   * Returns the contained instance, which must be present. If the
3   * instance might be absent, use {@link #or(Object)} or {@link
4   * #orNull} instead.
5   *
6   * <p><b>Comparison to {@code java.util.Optional}:</b> when the
7   * value is absent, this method throws {@link IllegalStateException},
8   * whereas the Java 8 counterpart throws
9   * {@link java.util.NoSuchElementException NoSuchElementException}.
10  *
11  * @throws IllegalStateException if the instance is absent ({@link
12  *     #isPresent} returns {@code false}); depending on this
13  *     <i>specific</i> exception type (over the more general {@link
14  *     RuntimeException}) is discouraged
15  */
```

Figure 1.2: Documentation comment of Guava's `Optional.get` method

`absent`, an `IllegalStateException` is thrown.

Figure 1.2 shows the documentation comment of the `get` method. Line 11 of this comment describe the constraint tested by `testGet_absent`.

In this example, manually writing the same information in `get`'s documentation comment is a redundant effort. Nevertheless, this information must appear in the documentation to inform developers who will use the API without access to the production or testing code. Moreover, the manually-written constraint is brittle: if the behavior of the method is changed, the modification will be detected by the failing test, but the documentation can silently become inconsistent.

By removing the need to write and maintain documentation that can be directly extracted from unit tests, constructural documentation will allow developers to focus their efforts on documenting non trivial information, such as abstractions and unusual designs, instead of common low-level usage specifications, and help ensure the consistency of the information for readers of the documentation.

1.2 Contributions

With this thesis, we present four contributions in the area of software evolution. First, we introduce the concept of constructural documentation, as well as a related terminology. Second, we present a general approach to extract constructural information embedded in unit tests as a potential solution to mitigate the redundancy between unit tests and documentation. Third, we contribute the design of a fully-implemented realization of this approach for Java systems that use JUnit as their unit testing framework. Finally, we report on a multi-case study about the enablers and obstacles to embedding constructural information in unit tests. The findings suggest contexts in which constructural documentation may have limited applicability, and practices to mitigate obstacles to constructural documentation.

This thesis introduces a new line of research, constructural software documentation, through one application, the generation of documentation from constructural information embedded in unit tests. Nevertheless, the potential of constructural documentation expands far beyond unit testing, or the generation of human-readable documentation.

Chapter 2

Terminology

This chapter defines the concept of constructural documentation, and clarifies the terminology related to this concept and its application to unit tests.

2.1 Constructural Documentation

We refer to a defined, parameterized structure in source code as a **shape**. Shapes can be composed of any element of code, from the structure of an identifier to a sequence of statements. A **shape instance** is the instantiation of a shape within a specific fragment of code, binding values to the shape's parameters. As an example, the shape that captures the convention to name unit tests with `test`, followed by the name of the unit under test, an underscore, and finally the state of the instance being tested, henceforth referred to as StateInName, is instantiated in the unit test of Figure 1.1 by `testGet_absent`, which binds the string "absent" to a parameter "state" (among other parameters). Another shape, henceforth referred to as TryFail, is composed of a `try-catch` statement, where the `try` block contains a single method call immediately followed by JUnit's `fail` method, and where the `catch` block is empty. Figure 1.1 also contains an instance of this shape. This shape has

one parameter: the type of exception declared in the `catch` clause. In the shape instance, this parameter is bound to the string "IllegalStateException".

While shapes only represent conventional or idiomatic structures in the code, a meaning can be associated with certain arrangements of shapes. We define a **construct** as a set of one or more shapes that encapsulates a self-contained meaning. Constructs are self-contained in the sense that the same construct, when used in different fragments of code, should encapsulate the same idea. This concept is similar to Allamanis and Sutton's definition of a programming idiom [19], but allows constructs to involve identifier names. As an example, consider both shape instances in the code of Figure 1.1. Assembled into a construct, their combination encodes some information about the tested method, namely that if an `Optional` instance is in the `absent` state, its method `get` throws an `IllegalStateException`. This information, embedded in source code by the use of a specific construct, is called **constructural information**.

Constructural information originates from a shared agreement among a community of developers. Therefore, the same construct can be associated with different information in different projects. For example, one organization may use the prefix `m` for all member variables of a class, whereas another may reserve the same prefix for mutable attributes only.

Constructural documentation designates the documentation potential of constructural information (note the difference between constructural information and documentation). This term can be used both for the action of embedding constructural information in source code and its final result, i.e., the documentation generated from this information.

2.2 Testing Terminology

A **unit test**, henceforth simply **test**, refers to the single code element (generally a method) that tests a feature of the production code. A class containing one or more unit tests is a **test class**. Test classes can also contain code other than unit tests, such as helper methods. More generally, a **test suite** refers to a set of unit tests, composed of one or more test classes. We reserve the term **method** for production code methods. To indicate that a particular method is tested by a unit test, we refer to this method as a **focal method** [20] of the unit test. A unit test can have zero or more focal methods, but typically has a single one. In the example of Figure 1.1, the focal method is `get`.

The **assertions** of a unit test are the statements that indicate expectations, e.g., that an expression evaluates to `true` or that a computed value is equal to the expected one, and fail the test if the expectations are not met. In JUnit, assertions use the static methods of the `Assert` class, such as `assertTrue` or `assertEquals`. The static method `fail`, which always fails a test, is also considered an assertion. Beside assertions, a unit test usually contains **setup** code that prepares the objects and environment used in the test.

Chapter 3

Constructural Documentation Approach

Applying the constructural documentation approach involves four steps. **Focal method detectors** detect the focal methods of the unit tests, then **shape functions** identify the shape instances present in the tests, and **documentation templates** generate the **information fragments** from the shape instances. The information fragments are then transformed into **documentation fragments** that are injected into the existing documentation. Note the difference between information fragments and documentation fragments: the former is an intermediate representation of the information, that will be synthesized to produce the latter, which is in a human-readable format.

3.1 Configuration

The first three steps rely on configuration provided in advance, which defines the constructs and their meaning. This configuration constitutes an initial investment of effort, whose dividends can thereafter be reaped. The configuration allows a fine-grained control over the conventions used: developers can define local constructs that apply to only one part of a system, on top of system-wise constructs.

The configuration essentially serves as a domain-specific language (DSL) for defining how to encode constructural documentation in unit tests. This DSL prevents the approach from prescribing a fixed set of constructs, which may not be intuitive to every community of developers. The use of a DSL also provides the flexibility to go beyond information that can be mined or inferred, an idea that has been addressed in other work [19, 21].

3.2 Focal Method Detection

The identification of focal methods relies on **focal method detectors**. These detectors are functions that take as input a unit test and return a possibly empty set of focal methods, identified using constructs that indicate the focal methods of a test.

For example, a common construct used to identify focal methods is for the fully qualified name of tests to systematically refer to the name of the method under test. This construct is used in the example of Figure 1.1.

3.3 Shape Functions

The main building blocks for the extraction of constructural information are **shape functions**. They take as input a unit test and its focal methods, and outputs either `false`, if the shape is not instantiated in the input, or a map of the values bound to each parameter of the shape instance.

For example, a shape function that detects the TryFail shape, when applied to Figure 1.1's test, would return the key "exception" associated with the value "IllegalStateException".

3.4 Documentation Templates

Documentation templates (henceforth, templates) encode constructs. A template is a pair that consists of a non empty set of shape functions and a **generator**. A generator is a function that takes as input the maps returned by the shape functions, and outputs an **information fragment**. Thus, the template can generate a piece of information when all of its shape functions detect an instance of their respective shapes.

For example, consider a template T composed of the two shapes StateInName and TryFail, and a generator that, given a value for a state and an exception type, generates the sentence `In the state "[state]", this method throws an [exception].` Using T with our running example generates the information fragment *In the state "absent", this method throws an IllegalStateException.*

The format of the information fragments generated by the generator is not prescribed by the approach. The format will differ according to the intended use of the generated documentation (e.g., to be inserted in production code, in a special XML file, or in a HTML-formatted documentation). The implementation described in Chapter 4 provides an example of a format designed to reduce unnecessary repetitions.

3.5 Documentation Generation

Before being injected into documentation, the information fragments go through a transformation step, which changes the set of information fragments (in an implementation-dependent intermediate representation) into a set of **documentation fragments**, ready to be injected into the documentation. The transformation step also allows the approach to leverage the interactions between the information fragments, which were generated independently, to improve the documentation.

3.6 Complete Algorithm

Figure 3.1 shows the complete algorithm for the production of constructural documentation.

The first phase of the algorithm is to extract information fragments from unit tests. For each test t, the algorithm applies all focal method detectors f and collects all focal methods in M_t (line 6). Using the example shown in Figure 1.1, the focal method `Optional.get` would be identified. It then applies each shape function s to the pair formed by the test t and its focal methods M_t, and filters out the shape functions that returned `false` (S_t, line 7). In our running example, S_t would contain the shape functions associated with StateInName and TryFail at this point.

Next, for each template p whose shape functions is a subset of S_t (line 11), p's generator is applied to the outputs of the shape functions (line 12). The generated information fragment i, together with the unit test it originates from and its focal methods, is added to a collection I. In our running example, using the template T described in Section 3.4, the triple (`OptionalTest.testGet_absent`, `{Optional.get}`, *In the state "absent", this method throws an IllegalStateException.*) would be added to I.

The second phase of the algorithm synthesizes the generated information and publishes the resulting documentation fragments.

The first operation is to group the information fragments by the focal method they describe (line 20). The set I of information fragments is then transformed into the set D of documentation fragments (line 21). The transformation can include, for example, grouping similar fragments or changing their format.

Once the information fragments have been transformed into documentation fragments, these fragments are published into the existing documentation of its focal methods. The publication process can be realized as anything from writing the documentation in a distinct file to modifying the documentation comment of a method. The publication process also links

Input: Test suite T

// Configuration
Variable F: focal methods detectors
Variable S: shape functions
Variable P: documentation templates

// Information Extraction
def I: *extracted documentation triples*
def M_t: *focal methods of test* t
def S_t: *shapes present in test* t
$I \leftarrow \emptyset$
for each test t in T **do**
    $M_t \leftarrow \bigcup_{f \in F} f(t)$
    $S_t \leftarrow \{s \in S : s(t, M_t) \neq \text{false}\}$
    for each template p in P **do**
        $S_p \leftarrow$ set of shape functions of p
        $g \leftarrow$ generator of p
        if $S_p \subset S_t$ **then**
            $i \leftarrow g\left(\bigcup_{s \in S_p} s(t, M_t)\right)$
            $I \leftarrow I \cup \{(t, M_t, i)\}$
        end if
    end for
end for

// Documentation Production
def D: *documentation fragments*
$M \leftarrow \bigcup_{(t, M_t, i) \in I} M_t$
for each m in M **do**
    $I_m \leftarrow \{(t, M_t, i) \in I : m \in M_t\}$
    $D \leftarrow \text{TRANSFORM}(I_m)$
    $\text{PUBLISH}(D, m)$
end for

Figure 3.1: Algorithm to generate documentation from constructural information

the documentation fragments to the focal method, so that if there is a need to regenerate the documentation, any outdated fragments can be trivially removed.

The specific details of the second phase depend on the format of the extracted documentation fragments and the intended use of the generated documentation. They are thus dependent on the implementation that supports the approach for a specific purpose. The implementation presented in the next chapter provides an example instantiation of the TRANSFORM and PUBLISH methods.

Chapter 4

Tool Support with DScribe

We implemented the proposed constructural documentation approach for Java systems that use JUnit as their testing framework. Our implementation takes the form of an Eclipse plug-in, called DScribe. DScribe's output consists of natural language sentences that are injected directly in the documentation comment of the focal methods, in production code. Therefore, readability, and in particular avoiding repetitive documentation, was a main design goal for DScribe. Our implementation relies on Eclipse's Java Development Tools (JDT) component to provide the abstract syntax trees (AST) of the unit tests to analyze and the source files to modify. Both JUnit 3's name-based and JUnit 4's annotation-based syntaxes are supported, but JUnit 5's new annotations are not, yet.

This implementation is a proof of concept of the applicability of constructural documentation to real world systems, and provides a concrete example of the implementation-dependent aspects of the approach.

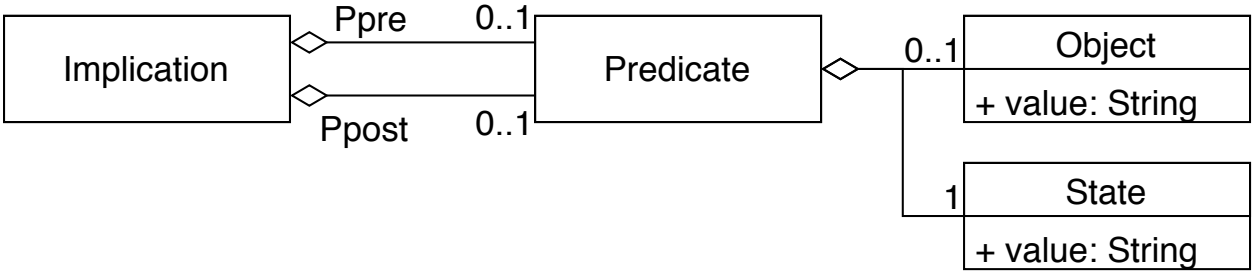

Figure 4.1: Data model of the format of the information fragments

4.1 Format of the Information Fragments

Figure 4.1 shows the data model of the information fragments that DScribe uses. An information fragment represents an *implication* by aggregating two predicates: a condition, denoted P_{pre}, and its consequent, P_{post}. Each predicate is itself composed of an object and a state, which are represented by strings. Hence, an implication comprises a 4-tuple of strings, (P_{pre}'s object, P_{pre}'s state, P_{post}'s object, P_{post}'s state), which encodes the idea *If [P_{pre}'s object] is [P_{pre}'s state], then [P_{post}'s object] is [P_{post}'s state].*

For example, the test in Figure 1.1 could embed the information fragment ("instance", "absent", "IllegalStateException", "raised"). The condition (P_{pre}) of this implication is composed of the object "instance" and the state "absent", i.e., *the instance is absent.* Similarly, the consequent (P_{post}) indicates that an *IllegalStateException is raised.* Altogether, this information fragment encodes the idea: *If the instance is absent, then an IllegalStateException is raised.* As illustrated by this example, our definition of "object" and "state" is quite flexible, to allow a maximum of flexibility in expressing information to document.

To increase the expressiveness of this representation, DScribe makes two provisions. First, although a predicate is normally composed of two elements, an object O and a state S, translating to O *is* S, predicates can also have no object. In this case, the predicate translates directly to the state, S. Hence, virtually any expression can be encoded as a predicate,

by omitting the object. However, using this strategy to encode predicates hinders their aggregation.

The second provision allows implications to lack either of its predicates. A missing predicate indicates that the other predicate must be or is always true. Thus, if an implication lacks P_{pre}, the implication is understood as a postcondition in the sense defined by the Design by Contract paradigm [22]. For example, the "implication" `( , , returned value, non null)` would translate to *The returned value is always non null.* The same logic applies to an implication that lacks P_{post}: it becomes a precondition.

This format is designed to facilitate the aggregation of similar information (described in Section 4.3). Furthermore, implications naturally map to the Arrange–Act–Assert [23, 24] (or Four-Phase Test [25]) pattern: P_{pre} encodes the *Arrange* (resp. *Setup*) step, and P_{post} encodes the *Assert* (resp. *Verify*) step.

4.2 Supported Functions

As a proof of concept, DScribe currently supports one type of focal method detectors, three types of shape functions, and one type of generators.

4.2.1 Focal Method Detectors

DScribe's focal method detectors rely on naming conventions [26], such as the one showcased in Figure 1.1. To detect the focal methods of a test, a focal method detector identifies the component of the test's name referring to the focal method and its class. As an example, given the unit test `OptionalTest.testGet_absent` as input, the detector uses the first part of the test class (`Optional`), and the part of the test name between `test` and the underscore (`Get`), to detect the focal method: `Optional.get` (the first letter is case-insensitive, to account for transformations due to camel case). If a unit test's name (or the test class' name) does

not follow the expected naming convention, or if the strings extracted from the names do not match an existing method, the focal method detector returns an empty set.

DScribe's focal method detector only considers methods *declared* in the target class (i.e., it does not search inherited methods). This is a design decision to avoid the ambiguity of overriden methods. Nevertheless, overloaded methods remain ambiguous. To discriminate between them, the names or types of the parameters of the method can be indicated in the unit test's name. If an ambiguity cannot be resolved, all candidates are returned.

4.2.2 Shape Functions

DScribe's shape functions rely on either the name, the signature, or the code of the unit test. For all three kinds of shape functions, the values bound to the parameters of the shape instances are encoded as strings. For example, if a parameter represents a type of exception, the name of the type is returned, rather than an instance of some more specialized API type that represents the exception type.

Similarly to focal method detectors, name-based shape functions identify the different components of a naming convention. They match the identified components against templates to validate the shape instance. For example, a name-based function can target the part of the test's name that comes after the keyword `Should` and match it against the template `Return[rval]` (the brackets indicate a placeholder, which act as the parameters of the shape). Thus, given the unit test `testCompareTo_WhenSmallerShouldReturnNegative`, the shape function returns the map `{rval: Negative}`.

Signature-based shape functions are used to detect the existence of specific elements in the signature of a unit test or focal method. Signature-based functions can detect if a modifier, an annotation, or a specific attribute of an annotation is declared in the signature of the method. They can also detect that a specific type or name is used for the arguments of the method, or validate the return type. For example, JUnit 4 introduced the `@Test`

annotations, which have an `expected` attribute that holds a reference to the class of an exception, to indicate that the given exception is expected as the result of the test. Thus, a signature-based shape function can detect the use of this annotation, and return the name of the thrown exception.

Finally, code-based shape functions compare the body of the test to a code fragment that serves as a model (described in the next paragraph). The model can contain placeholders, which act as the parameters of this shape function. For example, a code-based function can describe the TryFail shape previously mentioned. The model used to describe this shape would include a placeholder for the type of exception caught in the `catch` block, and the name of this type will be returned by the shape function.

4.2.3 Models of Code-Based Shape Functions

A model used for a code-based shape function is defined as a fragment of code that, injected directly inside a method and a class declarations, should be compilable, except for the resolution of the symbols. For example, the fragment

```
Type x = getInstance(input);
assertNull(x);
```

constitute a valid model, even if `Type`, `input`, `getInstance` and `assertNull` are unresolved, but

```
Type x = // ...
```

is invalid, because the equal operator does not have a right-hand side.

Code models can include four kinds of placeholders. First, a placeholder can be inserted between two Java statements, to indicate that any number of additional statements can be inserted at this point of the model. A second kind of placeholder applies the same semantic to the arguments of a method call.

The third and fourth kinds of placeholders indicate that an identifier or a literal value in the model can be substituted by code in the test. The third kind of placeholders can be substituted by any test code that respects the constraints detailed in the next paragraph, whereas the fourth kind of placeholder can be substituted by the name of the focal method or its class.

Identifiers are only allowed to be replaced by other identifiers, but a simple identifier can be replaced by a qualified one. In the previous example, assuming the variable `input` is a placeholder, it can be replaced by another variable, for example a class attribute named `data`, or a qualified name such as `this.data` or `Double.NaN`. It cannot, however, be replaced by a literal value (e.g., 0, `null`, a string) or a method call. On the other hand, literal values can be replaced by any Java expression, such as another identifier, a method call, a variable, or a combination of them using operators.

The comparison between the body of the test and the model is performed at the AST level. The ASTs of the two fragments are generated by Eclipse, without the binding information (types, methods and variables are not resolved). For the fragments to match, both ASTs must be exact copies, including the identifiers, except for eligible placeholder substitutions. The whitespace and comments are ignored.

DScribe does not use specialized clone detection techniques [27, 28], because the result of the comparison should be immediately obvious to the developers. It also aligns with the definition of a construct as a structure of code preferred over functionally equivalent alternatives to encode information. Hence, the shapes of the construct should not allow anything other than Type-I clones, except for explicitly defined placeholder substitutions.

4.2.4 Generators

DScribe's generators are composed of four text templates, one for each element of the implications. The text templates are strings with placeholders that indicate where to inject

the values of the maps returned by the shape functions. For example, a text template can be `an instance of [tryFail.exception]`. This template injects the value captured by the `exception` placeholder of the shape function `tryFail`.

4.2.5 Schema of the Configuration

DScribe is configured by writing a configuration file that encodes the sets of focal method detectors, shape functions, and documentation templates to use. Different configurations can be used inside the same project, e.g., for different modules, and each can rely on a base configuration provided for the whole project.

A configuration file is divided into four sections, identified by the four headers `PLACEHOLDERS`, `FOCAL_METHOD`, `SHAPES`, and `CONSTRUCTS`. The four sections can be defined in any order.

Comments are indicated by a leading hash sign (#). Comments can appear anywhere in the configuration file, and will be discarded before the configuration file is parsed.

Figure 4.2 shows the format of the configuration file. The top part shows the format of the placeholders, focal method detectors, shape functions and constructs. Angle brackets (<>) indicate the elements of the different formats. The production rules for these elements are described in the comments, in the bottom part.

The `PLACEHOLDERS` section defines the placeholders to use in the focal method detectors, name-based shape functions and signature-based shape functions. The declaration of a placeholder consists of a name and a regular expression, separated by a colon. For example, a general placeholder can be `any: .*`, which matches any string. A more restricted placeholder, enforcing the first character to be a letter, and forbidding underscores, is `PascalCase: [A-Z][A-Za-z0-9]`.

The special placeholders `package`, `class`, `method`, `argtype`, and `argname` must not be defined in this section (or anywhere else). Instead of using a regular expression, they are compared with the identifiers of the corresponding elements of the focal method when used

```
PLACEHOLDERS

<name>: <regular expression>

FOCAL_METHOD

<template>.<template>#<template>

SHAPES

<name>(Name): <placeholder>=<template>
<name>(Signature): <method or test>.<signature predicate>
<name>(Code): <code model>

CONSTRUCTS

<shapes> -> (<gen. template>, <gen. template>,
             <gen. template>, <gen. template>)

# <name> := sequence of letters
# <regular expression> := regular expression (Java syntax)
# <placeholder> := name of a placeholder
# <shape> := name of a shape
# <shapes> := one or more <shape>
# <bracket group> := [ <placeholder> ]
# <template> := free-form string, with <bracket group>
# <gen. group> := [ <shape>.<placeholder> ]
# <gen. template> := free-form string, with <gen. group>
# <method or test> := "method" or "test"
# <signature predicate> := one of <argnumber>, <argtypes>,
#                          <argnames>, <returntype>, <modifier>
#                          <annotationtype>, <annotationvalue>
# <argnumber> := argnumber=<number>
# <number> := a number
# <argtypes> := argtypes=<template>
# <argnames> := argnames=<template>
# <returntype> := returntype=<template>
# <modifier> := modifier=<name>
# <annotationtype> := annotationtype=<name>
# <annotationvalue> := annotationvalue=<name>.<name>=<template>
# <code model> := fragment of code
```

Figure 4.2: Format of DScribe's configuration file

in a shape function, and of the classes in the classpath when used in a focal method detector.

The `FOCAL_METHOD` section contains the definitions of the focal method detectors. The definition of a focal method detector consists of a template for the fully qualified name of the unit test (with the hash sign (#) to separate the test class from the unit test), using placeholders. At the minimum, the placeholders `package`, `class`, and `method` must each appear once. These placeholders must appear in this order, because the class will be looked for among the previously identified package, and the method among the previously identified class.

For example, the string `[package].[class]Test#test[method]` encodes the focal method detector associated to a minimalist naming convention. Although unusual, the focal method can also appear in the test class, e.g., `[package].[class]Test_[method]#[any]`, or the focal class in the unit test, e.g., `[package].[any]#test[class]_[method]`.

The `package` placeholder can appear more than once. In this case, each instance matches one or more of the fragments of the package. For example, the focal method detector `[package].test.[package].[class]Test#test[method]` associates the unit test `com.abc.test.def.SomeTest.testSomeMethod` with the focal method `com.abc.def.Some.someMethod`. The `method` placeholder can also appear more than once, to indicate multiple focal methods.

The `SHAPES` section contains the definitions of the three types of shape functions. Each definition starts with a name, the type of the shape function in parentheses (one of *Name*, *Signature*, or *Code*), and a colon. The rest of the definition depends on the type of the shape function. The format of a name-based function consists of the name of a placeholder (used in at least one focal method detector) and a template for its value, separated by an equal sign (=).

The format of signature-based functions starts with either `method.` or `test.`, to indicate whether to give the focal method's or the unit test's signature to the function. Next, the element of the signature targeted by the function is identified, by one of `argnumber`, `argtypes`,

`argnames`, `returntype`, `modifier`, `annotationtype`, or `annotationvalue`, followed by an equal sign (=). If the target is `argnumber`, the format continues with the expected number of arguments. If the target is `argtypes`, `argnames`, or `returntype`, the format continues with a template for the types or names of the arguments or return value. If the target is `modifier` or `annotationtype`, the format continues with the exact modifier or name of the annotation that must be present. Finally, if the target is `annotationvalue`, the format continues with the name of the annotation type, a period (.), the name of the attribute to verify, an equal sign, and finally a template for the value of this attribute. The following strings encode different signature-based shape functions (without the name and type):

```
method.argnumber=1

method.argtypes=String

method.argnames=x

method.returntype=void

method.modifier=static

method.annotationtype=@Override

test.annotationvalue=@Test.expected = [exception].class
```

The format of code-based functions consists of list of compilable Java statements, forming the model to match to the body of the unit tests. The model can extends on multiple lines, by indenting the code fragment. Thus, the code fragment extends until the first line that does not start with a white character.

Placeholders indicating an unknown number of statements or method arguments are encoded by a comment containing only an ellipsis (e.g., `/* ... */`). Identifiers and literal values used as placeholders for other identifiers or expressions are marked by placing a comment immediately after the identifier or the literal value. These comments must contain only a name to give to the placeholder. This name will be the key used to access the placeholder's value in the map returned by the shape function. The special names `method`

```
1 try {
2   call/*method*/ ( /*...*/ );
3   fail();
4 }
5 catch (Exception/*type*/ expected/*x*/) {}
```

Figure 4.3: Example of a code structure for a shape function

and `class` refer to the focal method and its class. The model should not contain any other comment (which the comparison algorithm ignores anyway).

Figure 4.3 shows an example of a code-based function. On the second line, the comment `/*method*/` indicates that the method `call` is a placeholder for the focal method, and the comment `/*...*/` allows any number of arguments to be passed to the focal method. On the last line, the two comments indicate placeholders, respectively for a type and a variable name.

The `CONSTRUCTS` section defines the documentation templates to use. The declaration of a documentation template consists of the list of shape function names to use, joined by a plus sign (+), and an anonymous generator declaration, separated by the two characters `->`.

The declaration of a generator consists of a template for each of the four elements of the implications to generate, joined by a comma (,), and all included inside parentheses.

Hence, the following string encodes a documentation generator:

```
stateInName + tryFail -> (instance, [stateInName.value], , throws a [tryFail.type])
```

Appendix A contains an example of a complete configuration file, used in the sample application described in Section 4.6.

4.3 TRANSFORM Function

DScribe's implementation of the TRANSFORM function (Figure 3.1, line 21) starts with the aggregation of the implications (i.e., information fragments), before translating each aggre-

Table 4.1: Values of the FLATTEN function

Origin (Same test?)	**2 elements** a, b	**3+ elements** $a, b, c, ...$
true	a and b	all of a, b, c, ...
false	a or b	any of a, b, c, ...

gated implication into a documentation fragment.

The aggregation algorithm relies on a notion of similarity, defined as follows: two or more predicates are similar if they all share the same object or state (the equivalence between objects or states is based on an exact string comparison). A set of similar predicates is referred to as a *similarity set*. The *common factor* of a similarity set is the shared object (or state) of all predicates of the set. The set of *variants* is the set of all the other states (or objects).

This notion of similarity extends to implications as well: two or more implications are similar if they all share the same P_{pre} or P_{post} (two predicates are the same if they have both the same object and state). Thus, a similarity set can be either a set of implications or a set of predicates.

Figure 4.4 shows the aggregation algorithm. It takes as input a set of implications and all unit tests from which the implications were produced, and outputs a new, aggregated set of implications. The FLATTEN function (line 25) is completely defined by Table 4.1. Its output depends on the *origin* argument and the number of elements in the set to flatten. When the arguments of this function are predicates, the words *and*, *or*, *any of*, and *all of* are inserted in upper case, to distinguish the aggregation of objects or states from the aggregation of predicates.

The NEW function (line 26) creates a new implications or predicate from its two arguments, depending on their type. The two arguments are always a P_{pre} and a P_{post}, or an object and a state (but not necessarily in this order).

Input: Set of implications I_0
Input: Set of unit tests T
Output: Aggregated set of implications I_2

$I_1 \leftarrow \emptyset$
for each test t in T **do**
    $I_{0,t} \leftarrow \{i \in I_0 : i \text{ was generated by } t\}$
    $I_{1,t} \leftarrow$ AGGREGATE(I_t, true)
    $I_1 \leftarrow I_1 \cup I_{1,t}$
end for
$I_2 \leftarrow$ AGGREGATE(I_1, false)
return I_2

// S can be a set of implications or predicates
function AGGREGATE(set S, origin)
    while $\exists$ similarity set $S' \subset S$ **do**
        $m \leftarrow$ MERGE(S', origin)
        $S \leftarrow (S \setminus S') \cup \{m\}$
    end while
    return S
end function

// S can be a set of implications or predicates
function MERGE(similarity set S, origin)
    $c \leftarrow$ common factor of S
    $D \leftarrow$ variants of S
    if S contains implications **then**
        $D \leftarrow$ AGGREGATE(D, origin)
    end if
    $d \leftarrow$ FLATTEN(D, origin)
    return NEW(c, d)
end function

Figure 4.4: Algorithm for the aggregation of implications

The resulting aggregated implications are translated into natural language sentences in two steps. First, each predicate is transformed into a string by returning only the state if the object is missing, or by concatenating the object, the word *is* (or *are* is the object is a combination of multiple values joined by *and* or *all of*), and the state.

Then, the strings generated from both predicates are transformed into a single sentence by the following rules.

1. If the implication is a precondition, return *[P_{pre}] MUST BE TRUE.*
2. If the implication is a postcondition, return *AFTER calling this method, [P_{post}].*
3. Otherwise, return *IF [P_{pre}], THEN [P_{post}].*

4.4 PUBLISH Function

The PUBLISH function (Figure 3.1, line 22) adds the output of the TRANSFORM function to the documentation of the focal method, directly in its documentation comment (creating it if necessary). To avoid interfering with the manually written documentation, and to indicate which information is generated automatically, each injected sentences are prefixed by a `@autogen` tag. To ensure that outdated documentation is removed, preexisting `@autogen` tags and their values are removed before inserting new ones.

4.5 Limitations

The constructural documentation approach, and its current implementation, are not without limitations.

One of the main limitations of the approach is that it does not consider the existing, manually-written documentation when injecting the generated documentation fragments.

Hence, manual and generated documentation can repeat some information, which would negatively impact the quality of the documentation. To mitigate this issue, an improved approach would need to parse the manual documentation to compute the textual similarity with the generated fragments, which is a hard problem in itself. While some techniques have already been proposed [29], using them would intrinsically add imprecision to the approach, and make the results less predictable, thus counteracting the fundamental principles of constructural documentation. Another strategy would be to write manual documentation in a highly structured way to make it machine-readable. However, this option would also counteract the original intent of constructural documentation by adding a new burden to documentation writers. Furthermore, the improved approach would need to account for the fact that manual documentation can be incorrect or outdated. In this case, the presence of incorrect documentation should not prevent the approach to inject correct information. Hence, while the possible redundancy of the generated documentation is an important limitation to the usability of the current approach, designing a viable solution constitutes a research project in itself.

In some situations, constructural documentation negatively affects readability. For example, the name `testSaveAndLoadOptions` is appropriate for a unit test whose focal methods are `saveOptions` and `loadOptions`. However, in the context of constructural documentation, the suitable name would be `testSaveOptionsAndLoadOptions`, which is less readable.

While our approach is designed to have completely predictable outcomes to reduce the possibility of false positives, it cannot completely avoid them. False positives can occur if two constructs are very similar, and are encoded in a way that some unit tests would instantiate both templates. Similarly, if a construct is encoded in an overly general way, there is a possibility that a unit test could instantiate it, but without conveying the intended meaning. Thus, when designing the constructs, developers must be careful not to make them too general or similar. While this limitation can be circumvented by injecting tracing

information in the constructs, e.g., specific codes in the test's name, this strategy adds a small burden for developers and can further decrease the readability of unit tests.

Reliance on low-level constructs is also a limitation of the approach. Any non trivial software will require additional (non-constructural) documentation to describe original ideas, abstractions and conceptual models of the system.

Another limitation is that some tests which cannot necessarily be organized so as to have a clear set of focal methods, can nevertheless contain valuable information. For example, integration tests can document the interactions between various components of the system, but they are not directly supported by our approach.

In addition to these intrinsic limitations, some limitations are implementation-dependent, and we plan to address them in future work:

DScribe assumes that unit tests focus on a single scenario. In particular, it assumes that the environment of a test is the same throughout the test. Unit tests with multiple scenarios, e.g., different arguments for a method, are therefore not directly supported by DScribe.

The information and documentation fragments of DScribe use strings to encode information. This format is intended to be flexible and support capturing diverse types of information. However, it does not allow to formally discriminate specific types of information, such as specifications about the returned value or thrown exceptions, which could be used to generate `@return` and `@throws` tags.

Finally, each shape function is applied independently by DScribe. This design prevents DScribe from leveraging more complex relations between shapes, such as precise sequences of shapes, or nested shapes. For example, nesting the TryFail shape inside a loop over an array of values indicates that the method should throw an exception for all values of the array.

4.6 Sample Application of DScribe

To demonstrate the practical feasibility of constructural documentation as embodied by DScribe, we discuss the generation of documentation for two sample test classes of the Weka machine learning library [30], `AlgVectorTest` and `UtilsTest`, which respectively test the `AlgVector` and `Utils` classes. This sample application of DScribe does not constitute an evaluation of its performance, but rather a validation that the implementation of the constructural documentation approach works as expected, and a concrete example of the results of the approach.

To enable this application of the approach, we created a copy of both test classes and refactored their unit tests to embed constructural information while keeping the original structure as much as possible. We made sure that both the original and the modified versions passed all tests.

To adapt `AlgVectorTest` to constructural documentation, we changed some identifiers for more common ones. For example, we standardized the names of the unit tests, and replaced variables such as `v` by `instance`, to indicate the variable on which the test is performed. We also declared a private helper method, `assertSameContent`, to encode a more complex assertion relating the content of two data classes. Additionally, three of the unit tests originally called a complex helper method to verify different assertions depending on the input. We replaced it with smaller helper methods. While this modification created some redundancy, it also allowed us to split the different assertions tested by the original method, and allowed DScribe to leverage this information. In the end, all five original unit tests have been modified or split, resulting in a new suite of nine tests. However, despite the 4 new tests, the file size only increased from 193 lines to 223 lines.

`UtilsTest`'s tests were designed to test both the typical usage of a method and its corner cases. This led to multiple scenarios being tested in the same unit test (a situation not

supported by DScribe). Therefore, we extracted the corner cases of each test into new tests. We wrote these tests to embed constructural documentation, using similar techniques as those used for `AlgVectorTest`. The tests left to use typical scenarios were not further modified, because they no longer contain information that would be relevant to add to the documentation.

In total, we could use DScribe to generate 13 documentation fragments for eleven methods in two classes. For the two modified test classes, the final configuration file contains a total of 72 lines, with five focal method detectors, nine shapes, and six documentation templates. Appendix A shows the original and modified versions of the test files, the generated documentation in the production files, and the configuration file encoding the the focal method detectors, shape functions and constructs.

This application provided insights about some challenges of using DScribe in a real world project, notably regarding the generated documentation. First, the amount of initial effort required to adapt existing tests for constructural documentation is not trivial. This sample application required approximately one day of work. However, this includes the time spent to become familiar with the test suites and the tested classes. Designing the constructs that will be used constitute an important effort, because the quality of the constructs will affect how they will be reused in other unit tests. Hence, the task of refactoring a whole existing test suite to embed constructural information is a large investment of effort. However, in the context of a progressive migration, where each new test is written using the defined constructs, the benefits of automatically generated documentation can repay more quickly the initial time and effort investment necessary to design these constructs.

Furthermore, while the use of implications as the format for the generated information did not strictly prevent information to be encoded, it led to some convoluted ways to express simple ideas. For example, to denote that the methods `quote` and `unquote`, from the `Utils` class, represent opposite actions, the generated implication is *IF the input to unquote is the*

output of quote, THEN its output is the original input of quote. Hence, while implications allow to encode a wide range of information, including a particular relation between two methods, enforcing this format is not suitable for documentation read by humans.

Chapter 5

Case Study on Constructural Documentation

We conducted a multi-case study to answer the question:

What technical factors act as enablers or obstacles for embedding constructural information in unit tests?

Testing code can be as rich and idiosyncratic as production code. Thus, although some practices can be conjectured in advance as having a positive or negative impact on constructural documentation, e.g., breaking complex tests into smaller ones, only the study of real systems can provide reliable insights on the potential of constructural documentation.

Furthermore, while similar, the concepts of constructural documentation and readability are not identical. Constructural documentation relies on hard-coded constructs that can be parsed by a machine. Readability, on the other hand, involves the ability of the human reader to fill gaps in incomplete information, and can express information at a higher level of abstraction. For example, the name `testSaveAndLoadOptions`, for a unit test whose focal methods are `saveOptions` and `loadOptions`, is easier to read than `testSaveOptionsAndLoadOptions`, but the latter allows to embed constructural information.

Table 5.1: Systems Used in the Case Study

System	Hash/Revision	URL
Freemind	643c55c9	`http://freemind.sourceforge.net/wiki/index.php/Main_Page`
Eclipse	d6d8a6aa51	`https://projects.eclipse.org/projects/eclipse.platform.ui`
Weka	r14866	`https://www.cs.waikato.ac.nz/ml/weka/`
Tomcat	r1835131	`https://tomcat.apache.org/`
Hibernate	35806c9dcb	`https://hibernate.org/orm/`

Table 5.2: Properties of the Subject Systems

System	Description	Prod. Files	Test Files	Inspected
Freemind	Desktop application	379	26	18
Eclipse	IDE component	3933	1669	49
Weka	ML library	1614	253	20
Tomcat	Servlet container	1402	475	16
Hibernate	ORM framework	3845	5647	12

While different groups of developers may already integrate recurrent constructs in unit tests (e.g., through naming conventions [31–33]), we are not aware of any project where constructs are systematically embedded in unit tests. Hence, we conducted a case study to gain insights about the potential of constructural documentation in real world systems, rather than in a controlled environment.

5.1 Cases and Units of Analysis

The five selected systems are Freemind, the Eclipse Platform UI (Eclipse), Weka, Apache Tomcat (Tomcat) and Hibernate ORM (Hibernate). Tables 5.1 shows the exact version of the systems used, by indicating the Git hash or Subversion revision number of the last commit to the system in their official repository, at the time they were downloaded, in July 2018. Table 5.2 presents the domain of each system, together with the number of Java files

in the production and testing code for each system. The *Inspected* column shows the number of test classes that were used for the case study. All systems are over 15 years old.

The research question focuses on *technical* factors, i.e., properties of the source code of a snapshot of the systems. Other factors, such as the development process, while potentially related, are out of the scope of this study. Consequently, a unit of analysis corresponds to a single unit test.

For a specified system, the following iterative sampling procedure generated samples of unit tests. First, the procedure is to select a package at random, uniformly from all packages in the testing code. Next, the procedure is to randomly select three of the test classes from the selected package. If a package contains less than three test classes, the procedure is to select a new (unselected) one. Finally, all unit tests from the three selected test classes are integrated into the sample.

We used as many iterations of the selection procedure as needed to reach saturation, which we defined as when three consecutive iterations generate no new noted observations.

5.2 Data Collection Phase

For each project, the sampling produced a set of unit tests that we manually analyzed. To guide the collection of data, we reported the unit under test, and coded the purpose, format, and recurrent constructs of each unit test using open codes [34]. Finally, we noted any other factor contributing to prevent or enable the embedding of constructural information in the unit test, and any other relevant observation, either from a single unit test, a test class, or the selected package.

After the initial investigation, and a preliminary analysis of the open codes, we systematically re-coded each tests, using a closed set of codes, derived from our previous observations. These codes are presented in Appendix B.

5.3 Results

We present our findings as eight technical factors that have a direct impact, either positive (marked by [+]), negative ([-]), or mixed ([!]), on constructural documentation.

When presenting a factor, we present the proportion of tests in each system that presented the factor, discretized into five categories: none (0%), a few (0-5%), some (5-10%), common (10-50%), and pervasive (50-100%). This split between system is reported using in-line histogram-like diagrams, such as  . The x-axis represent the five different systems, Freemind, Eclipse, Weka, Tomcat, and Hibernate, always in this order. The y-axis represents the five categories, from pervasive (bar reaches the top of the diagram) to none (no bar). Thus, in this example, the hypothetical factor is pervasive in Freemind, common in Eclipse, etc. The discretization was necessary for meaningful differences in the data to be perceptible in the small in-line format. It is important to note that we provide this data to be transparent about the amount of evidence that supports each observation. It should not be assumed that they generalize to larger populations of tests. It is also important to interpret the y-value of the diagrams as *levels*, not *absolute values*, otherwise the amount of evidence will appear to be greater than reported. As an additional precision to mitigate the risk of misinterpretation of the summary data, we include, with each diagram, the absolute number of tests to which the code was assigned.

5.3.1 [!] Unit Test Coverage

The definition of a "unit", in the context of unit testing, varies between development teams. Different definitions change what constructural documentation should describe (e.g., a class or a method), but also how it can be embedded in the test. For example, tests where the unit under test is a class (266, ), such as Hibernate's `AbstractBasicJtaTestScenarios.basicBmtUsageTest`, can test the interaction between different methods, or the state of the ob-

ject after a specific sequence of method calls. Furthermore, such tests can execute several scenarios to test different specifications of the class. In contrast, tests where the unit under test is a single scenario for a single method (110, ), such as Eclipse's `ComputedObservableMapTest.testGet_ElementInKeySet`, are typically shorter, and test a single specification. Hence, constructs in this type of unit test are likely to relate to the same specification, whereas constructs found in unit tests focusing on a whole class may represent different specifications. Other definitions of the unit under test found in the sample include a whole method (173, ) and even public static fields of a class (6, ), like Freemind's `HtmlConversionsTest.testEndContentMatcher`, which checks that a string encodes a correct regular expression.

5.3.2 [!] Purpose of the Test

While an exhaustive classification of the purpose of unit tests does not exist, different canonical cases can be identified. For example, tests designed to evaluate the performance of a method (1, ) or the capacity of a system to support high workloads (2,) involve different techniques than tests designed to check the usage of a method under typical circumstances (370, ), e.g., testing `Math.sqrt` for positive numbers, tests for corner cases of a method (23, ), e.g., `Math.sqrt` with `Double.NaN` as input, or tests for special cases like invalid inputs (98, ), e.g., `Math.sqrt` with a negative argument.

This purpose influences the information that can be extracted from a unit test, and whether it should be extracted at all. For example, Weka's `RemoveByNameTest.testTypical` tests that the `RemoveByName` filter correctly removes the expected values from a data set. While this scenario is important to test, it does not capture a useful piece of information to include in the documentation.

In the same vein, tests related to a bug (34, ), such as Tomcat's `TestELInterpreterFactory.testBug54239`, are not good targets for constructural documentation, because they

test that an unexpected behavior should not happen, which is a trivial information.

5.3.3 [+] Structured Unit Test Names

The value of the information carried by identifiers has already been recognized in previous work [35]. In the context of constructural documentation, this information must additionally be encoded in explicit structures. At the minimum, names that indicate the focal method of the test (108, ) in a structured way avoid the need to infer this information, which is critical to understand the test (and it is a requirement in our implementation of the focal method detectors). More informative names can contain the input being tested, or the expected outcome (51, ). The tests in Eclipse's `ComputedObservableMapTest` include such information. However, this practice leads to very long names (with up to 64 characters just in our sample).

A special case of this practice is the use of structures local to a small context, such as a single class or package. For example, the twelve tests in Tomcat's `CheckOutThreadTest` all match the following pattern: `test(DBCP|Pool)Threads(10|20)Connections(10|20)(Validate)?(Fair)?` Nevertheless, this structure must be explicitly defined for the purpose of constructural documentation.

5.3.4 [+] Recurrent variable names

Recurrent variable names can have a positive impact on constructural documentation, by indicating the overall structure of the test. This practice differs from the more general advice to use descriptive variable names: it expects developers to reuse *exactly* the same names in specific contexts, even if the name is less informative in the context of the test. Informative names that are not recurrent can be useful to the human reader, but are typically not helpful for constructural documentation. This practice allow developers to group similar

specifications, applicable to many contexts, into a single construct.

For example, consider the following test fragment, adapted from Freemind's `Base64-Tests.testDifferentBase64ers`:

```
Object input = ...;
Object output = met1(input);
Object back = met2(output);
```

Without knowing the purpose of the methods, one can expect `met2` to undo what `met1` does, and `input` to equal `back`. Thus, the use of variables specifically named `input` or `back` (22, ) already embeds constructural information. In the case of this test, the two tested methods encode and decode a string using a Base64 encoding scheme. However, other tests reused a similar pattern, for example to quote and unquote a string.

5.3.5 [!] Helper Methods to Hide Information

Helper methods are commonly used in our sample (438, ). They can be beneficial (252, ) by mitigating other threats, for example by hiding complex assertion logic, assertions in the setup phase of the test, or handling of external resources. However, they are a double-edged sword: they can hide pertinent information from the test.

A helper method hides pertinent information when it performs both the *Act* and the *Assert* phases of the test (184, ). For example, Eclipse's `IWorkbenchWindowActionDelegate-Test.testInit` relies on the helper method `testRun`, which performs the manipulation under test, then asserts the result. Using this method completely hides what is being tested, and how.

In particular, "one-size-fits-all" methods (158, ), such as `doTestParser`, used by all tests of Tomcat's `TestELParser` test class, can perform different tests depending on their arguments, but encapsulate a complex logic to adapt the operations and assertions to the

arguments. Thus, while such methods can reduce repetitions in a test suite, they are detrimental to constructural documentation.

5.3.6 [-] Complex Assertion Structures

JUnit provides a limited set of methods to perform assertions. Thus, many non trivial tests must use complex assertion structures to verify a correct output or behavior. These structures can hide the expected output or behavior tested by the unit test.

Complex structures include assertions in long sequences (305, ), or in loops and conditions (44, ). For example, Weka's `DiscretizeTest.testInverted` contains an `if-else` statement nested inside a `for` loop, to perform the assertions. Thus, to extract the information about what is being tested, the construct must relate the parameters of the loop (what is being iterated over?) to the nested condition (when does each branch is taken?), and to the other aspects of the assertions (what is being asserted in each branch?). An effective way to mitigate this threat is to hide the complex assertion structures inside well-named helper methods (52, ).

More complex structures, such as performing the assertions in stubs and mock objects (2, ), or lambda expressions (7, ) are even more detrimental, as the flow of the test is harder to analyze statically, which again adds to the complexity of the assertion structure. To mitigate the need for a mock object, Eclipse's `IntroPartTest.testOpenAndClose` registers the pertinent information in an attribute of the mock object instead of performing the assertion. Thus, the unit test can perform the necessary assertions itself, hence exposing the expected behavior.

5.3.7 [-] Assertions in the Setup Phase

Including assertions in the setup phase of a unit test (64, ) can result in the elements of the setup being extracted as documentation, instead of the units truly under test. For example, Hibernate's `AbstractBasicJtaTestScenarios.basicBmtUsageTest` includes assertions to validate the preconditions of the test. These assertions could be identified by a construct, which would then generate documentation about the wrong objects.

5.3.8 [-] Reliance on Constrained Resources

Reliance on constrained resources such as a connection to a server (30, ), I/O operations (123, ), or multiple threads (15, ) can be detrimental to constructural documentation.

In our sample, tests handling such resources generally had a more complex structure, e.g., a large `try-finally` to ensure to leave the resource in an uncorrupted state at the end of the test. For example, Tomcat's `TestPersistentProviderRegistrations.testSaveProvider-WithoutLayerAndAC` uses a `try-finally` block to ensure that a file is deleted after the test is completed. These additional structures hinder constructural documentation, because they represent additional cases to take into account by the shapes and the constructs, but they do not represent relevant information about the unit under test.

Additionally, constrained resources may be in a non-deterministic environment. Therefore, tests must handle the unexpected behavior of the environment. For example, Tomcat's `TestMapperPerformance.testPerformance` executes the manipulation under test twice, so that if it fails the first time, possibly due to an exceptional event in the environment, it can pass the second time.

5.4 Threats to Validity

The case study relied on the identification of constructs, a subjective concept relative to the experience of every developer: different development teams can identify constructs in different structures in the code. Hence, there is a risk of investigator bias: the conclusions may reflect the personal experience or cultural context of the author. This experimental design choice was necessary because the data analysis required a very high initial effort investment to study the systems, and a consistent point of view from one system to the other. Thus, the coding procedure could not be packaged into independent sets of data to be labeled by independent coders. To mitigate this risk, we carefully linked each test to each conclusion presented in this thesis.

Additionally, the investigator was not a contributor of any of the subject systems, and thus did not know the common practices and shared knowledge of the development community. Therefore, there is a possibility that the investigator missed existing constructs. However, these missed opportunities do not invalidate the findings, because the objective of the case study was not the creation of an exhaustive classification of the constructs.

Finally, the limited number of systems, due to the very high cost of inspection, means some practices will have been missed. Again, those missed practices do not invalidate our findings, and the five subjects were selected to represent a wide range of domains and applications, from different development contexts.

Chapter 6

Related Work

Literate programming [36] encourages developers to write code that is easily readable by humans. The code to perform recurrent operations is often crystallized into common idioms [19] or patterns [17], thus providing a familiar way to express a known operation to experienced developers. Approaches have been developed to mine these idioms [19] and patterns [21, 37–39]. These techniques can provide key insights about the design of a system. However, in contrast to constructural documentation, they do not allow developers to control the idioms or patterns that are targeted, or their associated meaning.

Due to the common issues with documentation, such as high proportions of outdated [3], missing [4], or non informative [7] documentation, many automatic documentation generation techniques have been proposed. These techniques rely on static [40] and dynamic [41] analysis of the method, its context [42], or other kinds of information [43, 44]. The generated documentation can apply to a whole class [45], a single method [46], or its parameters [40]. Other work focused on mining specifications [47], program invariants [48], or usage scenarios [38]. However, these techniques are limited by the quality of the inference that can be made, while our approach does not infer any information, but rather extracts the information previously embedded by developers.

In the same vein, previous work has focused on the summarization of tests to produce multi-line comments [49] to help developers understand unit tests. In particular, Zhang et al. [50] developed a test summarization technique to generate informative test names. Other work focused on detecting the units under test of testing code, using techniques from the Last Call Before Assert (LCBA) [26] to advanced slicing and coupling based approaches [51]. Such techniques can complement our approach by suggesting constructural information to embed in existing tests. However, these automated techniques come at the cost of uncertainty and imprecision, whereas with constructural documentation, the cost is in the initial investment in configuration, which then yields the benefit of preventing inaccurate results, and making the information extraction process more transparent to the developers.

Closer to our perspective are the ideas behind behavior-driven development (BDD) [52], a methodology derived from test-driven development [24], that views testing code as more than just a way to test specific aspects of the production code. According to BDD, testing code should be a primary source of information regarding the specifications of the behaviors of the code. BDD frameworks such as JBehave [53] often mix documentation with testing code. This methodology helps address the problem of the evolution of documentation, as changing a unit test will likely result in the documentation being updated too. The downside of these frameworks, however, is that unit tests can become more verbose, as even the simplest tests must be described in human-readable format.

Another line of research related to our work attempts to identify incorrect information in documentation. Approaches to detect fragile comments [11], outdated requirements [54], or documentation defects [55, 56] helps improve the quality of documentation in software systems. Other work have proposed approaches to recover traceability links between documentation and code [8, 9], to facilitate the maintenance of this documentation. However, by automatically removing outdated information and regenerating it, our approach offers a more active solution, but possibly at the cost of lower applicability.

Finally, we note some previous work on the development of techniques to automatically generate unit tests, either from documentation [57] or source code [58, 59]. These techniques perform a task complementary, or even opposite, to our approach.

Chapter 7

Conclusion

Motivated by the observation that documentation and testing code often contain redundant and recurring information, we designed an approach to allow developers to explicitly assign a meaning to code constructs, and leverage such constructs in unit test to generate documentation. This approach can relieve developers of some of the burden of writing repetitive documentation, while providing an incentive to write extensive and well-structured test suites.

We implemented our approach as an Eclipse plug-in, called DScribe. The development of this plug-in provided an example for the design of the implementation-dependent parts of the approach, and allowed us to assess the feasibility, as well as the challenges related to a tool-supported application of constructural documentation. The application of the approach to two test classes highlighted the limitations and weaknesses of the approach and its implementation, and gave us ideas to integrate to the future versions of DScribe.

The development of this approach led us to the larger concept of embedding documentation directly in their code by using commonly agreed upon constructs. We term this concept *constructural documentation*, and define a terminology related to it. In this context, the proposed approach to generate documentation from unit tests constitute an example of the

potential of constructural documentation.

To assess the potential usability of our approach, we report on a case study in which we investigated the testing practices in real world systems that act as enablers or obstacles for constructural documentation. Our findings show that constructural documentation is not a silver bullet: not all tests capture information that is pertinent for documentation, and many features of non trivial systems, such as concurrent programming and I/O management can hinder the embedding of constructural information. However, the case study also suggests ways to handle these limitations, e.g., by the use of recurrent variable names. Thus, where it is applicable, constructural documentation has the potential to streamline the generation of low-level specifications of the units of a system. In future work, we hope to improve our constructural documentation approach and implementation, to address their weaknesses highlighted by the case study.

This thesis demonstrates the potential of constructural documentation to leverage the parallel between the information encoded in the unit tests and the documentation. The natural extension of this work is to apply constructural documentation to production code as well. One possible avenue would be to enforce developers to embed constructural documentation in specific places, such as in the names of their unit tests, rather than simply incentivize them to do it. This approach could lead to a new development methodology, integrating the use of meaningful constructs as a primary concern.

Bibliography

[1] V. Massol and T. Husted, *JUnit in action*. Manning Publications Co., 2003.

[2] Oracle Corporation. javadoc – The Java API Documentation Generator. [Online]. Available: https://docs.oracle.com/javase/1.5.0/docs/tooldocs/windows/javadoc.html

[3] T. C. Lethbridge, J. Singer, and A. Forward, "How Software Engineers Use Documentation: The State of the Practice," *IEEE Software*, vol. 20, no. 6, pp. 35–39, 2003.

[4] B. Fluri, M. Würsch, and H. C. Gall, "Do code and comments co-evolve? On the relation between source code and comment changes," in *Proceedings of the Working Conference on Reverse Engineering*, 2007, pp. 70–79.

[5] Google Inc. Google Java Style Guide. [Online]. Available: https://google.github.io/styleguide/javaguide.html

[6] Apache Portals. (2013) Coding Standards. [Online]. Available: https://portals.apache.org/development/code-standards.html

[7] W. Maalej and M. P. Robillard, "Patterns of Knowledge in API Reference Documentation," *IEEE Transactions on Software Engineering*, vol. 39, no. 9, pp. 1264–1282, 2013.

[8] A. Marcus and J. I. Maletic, "Recovering documentation-to-source-code traceability links using latent semantic indexing," in *Proceedings of the 25th International Conference on Software Engineering*, 2003, pp. 125–135.

[9] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo, "Recovering traceability links between code and documentation," *IEEE Transactions on Software Engineering*, vol. 28, no. 10, pp. 970–983, 2002.

[10] B. Dagenais and M. P. Robillard, "Recovering Traceability Links between an API and Its Learning Resources," in *Proceedings of the IEEE International Conference on Software Engineering*, 2012, pp. 47–57.

[11] I. K. Ratol and M. P. Robillard, "Detecting Fragile Comments," in *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*, 2017, pp. 112–122.

[12] B. Dagenais and M. P. Robillard, "Creating and evolving developer documentation: Understanding the decisions of open source contributors," in *Proceedings of the 18th ACM SIGSOFT international symposium on Foundations of Software Engineering*, 2010, pp. 127–136.

[13] K. Beck, J. Grenning, R. C. Martin, M. Beedle, J. Highsmith, S. Mellor, A. van Bennekum, A. Hunt, K. Schwaber, A. Cockburn, R. Jeffries, J. Sutherland, W. Cunningham, J. Kern, D. Thomas, M. Fowler, and B. Marick. (2001) Manifesto for Agile Software Development. [Online]. Available: http://agilemanifesto.org/

[14] Mozilla. (2018) Coding Style. [Online]. Available: https://developer.mozilla.org/en-US/docs/Mozilla/Developer{_}guide/Coding{_}Style

[15] Python.org. (2013) PEP 8 – Style Guide for Python Code. [Online]. Available: https://www.python.org/dev/peps/pep-0008/

[16] Sun Microsystems, Inc. (1999) Code Conventions for the Java Programming Language. [Online]. Available: https://www.oracle.com/technetwork/java/codeconvtoc-136057.html

[17] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[18] Google Inc. Guava: Google Core Libraries for Java. [Online]. Available: https://github.com/google/guava

[19] M. Allamanis and C. Sutton, "Mining idioms from source code," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2014, pp. 472–483.

[20] M. Ghafari, C. Ghezzi, and K. Rubinov, "Automatically identifying focal methods under test in unit test cases," in *IEEE 15th International Working Conference on Source Code Analysis and Manipulation*, 2015, pp. 61–70.

[21] J. Dong, Y. Zhao, and T. Peng, "A Review of Design Pattern Mining Techniques," *International Journal of Software Engineering and Knowledge Engineering*, vol. 19, no. 06, pp. 823–855, 2009.

[22] B. Meyer, "Applying 'design by contract'," *Computer*, vol. 25, no. 10, pp. 40–51, 1992.

[23] B. Wake, "3A – Arrange, Act, Assert," https://xp123.com/articles/3a-arrange-act-assert/, 2011.

[24] K. Beck, *Test-driven development: by example*. Addison-Wesley, 2003.

[25] G. Meszaros, *xUnit Test Patterns: Refactoring Test Code*. Oxford University Press, 2006.

[26] B. Van Rompaey and S. Demeyer, "Establishing Traceability Links between Unit Test Cases and Units under Test," in *13th IEEE European Conference on Software Maintenance and Reengineering*, 2009, pp. 209–218.

[27] S. Bellon, R. Koschke, G. Antoniol, J. Krinke, and E. Merlo, "Comparison and evaluation of clone detection tools," *IEEE Transactions on software engineering*, vol. 33, no. 9, 2007.

[28] C. K. Roy, J. R. Cordy, and R. Koschke, "Comparison and evaluation of code clone detection techniques and tools: A qualitative approach," *Science of computer programming*, vol. 74, no. 7, pp. 470–495, 2009.

[29] R. Mihalcea, C. Corley, and C. Strapparava, "Corpus-based and Knowledge-based Measures of Text Semantic Similarity," in *Proceedings of the American Association for Artificial Intelligence*, 2006, pp. 775–780.

[30] E. Frank, M. A. Hall, and I. H. Witten, *The WEKA workbench. Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques"*, 4th ed. Morgan Kaufmann, 2016.

[31] R. Osherove, "Naming standards for unit tests," http://osherove.com/blog/2005/4/3/naming-standards-for-unit-tests.html, 2005.

[32] A. Kumar, "7 Popular Unit Test Naming Conventions," http://vitalflux.com/7-popular-unit-test-naming-conventions/, 2014.

[33] J. Reid, "Unit Test Naming Convention: The 3 Most Important Parts," https://qualitycoding.org/unit-test-names/, 2015.

[34] M. B. Miles, A. M. Huberman, and J. Saldana, *Qualitative data analysis*. Sage, 2013.

[35] B. Caprile and P. Tonella, "Nomen Est Omen: Analyzing the Language of Function Identifiers," in *Sixth Working Conference on Reverse Engineering*, 1999, pp. 112–122.

[36] D. E. Knuth, "Literate Programming," *The Computer Journal*, vol. 27, no. 2, pp. 97–111, 1984.

[37] Z. Li and Y. Zhou, "PR-Miner: automatically extracting implicit programming rules and detecting violations in large software code," in *Proceedings of the Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, 2005, pp. 306–315.

[38] M. Acharya, T. Xie, J. Pei, and J. Xu, "Mining API Patterns As Partial Orders from Source Code: From Usage Scenarios to Specifications," in *Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, 2007, pp. 25–34.

[39] A. Pandel, M. Gupta, and A. Tripathi, "DNIT – A new approach for design pattern detection," in *Proceedings of the International Conference on Computer and Communication Technology*, 2010, pp. 545–550.

[40] G. Sridhara, L. Pollock, and K. Vijay-Shanker, "Generating parameter comments and integrating with method summaries," in *Proceedings of the IEEE International Conference on Program Comprehension*, 2011, pp. 71–80.

[41] M. Sulír and J. Porubän, "Generating Method Documentation Using Concrete Values from Executions," in *Symposium on Languages, Applications and Technologies*, 2017, pp. 3:1–3:13.

[42] P. W. McBurney and C. McMillan, "Automatic documentation generation via source code summarization of method context," in *Proceedings of the 22nd International Conference on Program Comprehension*, 2014, pp. 279–290.

[43] S. Shoham, E. Yahav, S. J. Fink, and M. Pistoia, "Static Specification Mining Using Automata-Based Abstractions," *IEEE Transactions on Software Engineering*, vol. 34, no. 5, pp. 651–666, 2008.

[44] C. Le Goues and W. Weimer, "Specification mining with few false positives," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 2009, pp. 292–306.

[45] L. Moreno, J. Aponte, G. Sridhara, A. Marcus, L. Pollock, and K. Vijay-Shanker, "Automatic generation of natural language summaries for java classes," in *Proceedings of the 21st International Conference on Program Comprehension*, 2013, pp. 23–32.

[46] G. Sridhara, E. Hill, D. Muppaneni, L. Pollock, and K. Vijay-Shanker, "Towards automatically generating summary comments for java methods," in *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, 2010, pp. 43–52.

[47] G. Ammons, R. Bodík, and J. R. Larus, "Mining specifications," in *Proceedings of the 29th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 2002, pp. 4–16.

[48] M. D. Ernst, W. G. Griswold, Y. Kataoka, and D. Notkin, "Dynamically discovering pointer-based program invariants," in *Proceedings of the International Conference on Software Engineering*, vol. 373, 1999.

[49] S. Panichella, A. Panichella, M. Beller, A. Zaidman, and H. C. Gall, "The impact of test case summaries on bug fixing performance: An empirical investigation," in *Proceedings of the 38th International Conference on Software Engineering*, 2016, pp. 547–558.

[50] B. Zhang, E. Hill, and J. Clause, "Towards automatically generating descriptive names for unit tests," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, 2016, pp. 625–636.

[51] A. Qusef, G. Bavota, R. Oliveto, A. De Lucia, and D. Binkley, "Scotch: Test-to-code traceability using slicing and conceptual coupling," in *Proceedings of the 27th IEEE International Conference on Software Maintenance*, 2011, pp. 63–72.

[52] M. Soeken, R. Wille, and R. Drechsler, "Assisted behavior driven development using natural language processing," in *Proceedings of the International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, 2012, pp. 269–287.

[53] JBehave.org. (2017) What is JBehave? [Online]. Available: https://jbehave.org/

[54] E. Ben Charrada, A. Koziolek, and M. Glinz, "Identifying outdated requirements based on source code changes," in *IEEE International Requirements Engineering Conference*, 2012, pp. 61–70.

[55] Y. Zhou, R. Gu, T. Chen, Z. Huang, S. Panichella, and H. Gall, "Analyzing APIs Documentation and Code to Detect Directive Defects," in *IEEE/ACM International Conference on Software Engineering*, 2017, pp. 27–37.

[56] L. Tan, D. Yuan, G. Krishna, and Y. Zhou, "/*iComment: Bugs or Bad Comments?*/," in *Proceedings of the ACM Symposium on Operating Systems Principles*, 2007, pp. 145–158.

[57] J. Offutt and A. Abdurazik, "Generating Tests from UML Specifications," in *UML'99 – The Unified Modeling Language*, 1999, pp. 416–429.

[58] S. Thummalapenta, T. Xie, N. Tillmann, J. De Halleux, and W. Schulte, "Mseqgen: Object-oriented unit-test generation via mining source code," in *Proceedings of the the*

7th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT symposium on The Foundations of Software Engineering, 2009, pp. 193–202.

[59] K. Taneja and T. Xie, "Diffgen: Automated regression unit-test generation," in *Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering*, 2008, pp. 407–410.

Appendix A

Documents Related to the Sample Application of DScribe

This appendix contains the files related to the sample application of DScribe to two test classes of the Weka system, `AlgVectorTest` and `UtilsTest`, described in Section 4.6.

The appendix shows three files for each test class: the original version of the test files, a version manually modified by the author to integrate constructural documentation, and the production files, that show the generated documentation.

Both the original and modified version of `UtilsTest.java` contain the ten data sets used in the tests at the end of the file. Because these data sets contain several thousands of real numbers, and are not relevant to understand the essence of the unit tests, only the first two data sets are included, as an example of the others.

The appendix include only the focal methods documented by DScribe, rather than the entire production files, to highlight the generated documentation (in the documentation comment of the methods, prefixed by the tag `@dscribe`).

Finally, the appendix include the configuration file that DScribe used to produce the documentation shown.

Original Test Files

AlgVectorTest.java

```
1  /*
2   *    This program is free software: you can redistribute it and/or modify
3   *    it under the terms of the GNU General Public License as published by
4   *    the Free Software Foundation, either version 3 of the License, or
5   *    (at your option) any later version.
6   *
7   *    This program is distributed in the hope that it will be useful,
8   *    but WITHOUT ANY WARRANTY; without even the implied warranty of
9   *    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
10  *    GNU General Public License for more details.
11  *
12  *    You should have received a copy of the GNU General Public License
13  *    along with this program.  If not, see <http://www.gnu.org/licenses/>.
14  */
15
16 /*
17  * Copyright (C) 2007 University of Waikato
18  */
19
20 package weka.core;
21
22 import java.util.Random;
23
24 import junit.framework.Test;
25 import junit.framework.TestCase;
26 import junit.framework.TestSuite;
27
28 /**
29  * Tests AlgVector. Run from the command line with:<p/>
30  * java weka.core.AlgVectorTest
31  *
32  * @author FracPete (fracpete at waikato dot ac dot nz)
33  * @version $Revision: 8034 $
34  */
35 public class AlgVectorTest
36   extends TestCase {
37
38   /** for generating the datasets */
39   protected Random m_Random;
40
41   /**
42    * Constructs the <code>AlgVectorTest</code>.
43    *
44    * @param name        the name of the test class
45    */
46   public AlgVectorTest(String name) {
47     super(name);
48   }
49
50   /**
51    * Called by JUnit before each test method.
52    *
53    * @throws Exception  if an error occurs
54    */
55   protected void setUp() throws Exception {
56     super.setUp();
57
58     m_Random = new Random(1);
```

```
59    }
60
61    /**
62     * Called by JUnit after each test method
63     *
64     * @throws Exception  if an error occurs
65     */
66    protected void tearDown() throws Exception {
67      super.tearDown();
68
69      m_Random = null;
70    }
71
72    /**
73     * generates data with the given amount of nominal and numeric attributes
74     *
75     * @param nominal      the number of nominal attributes
76     * @param numeric      the number of numeric attributes
77     * @param rows         the number of rows to generate
78     * @return             the generated data
79     */
80    protected Instances generateData(int nominal, int numeric, int rows) {
81      Instances           result;
82      TestInstances       test;
83
84      test = new TestInstances();
85      test.setClassIndex(TestInstances.NO_CLASS);
86      test.setNumNominal(nominal);
87      test.setNumNumeric(numeric);
88      test.setNumInstances(rows);
89
90      try {
91        result = test.generate();
92      }
93      catch (Exception e) {
94        result = null;
95      }
96
97      return result;
98    }
99
100   /**
101    * tests constructing a vector with a given length
102    */
103   public void testLengthConstructor() {
104     int len = 22;
105     AlgVector v = new AlgVector(len);
106     assertEquals("Length␣differs", len, v.numElements());
107   }
108
109   /**
110    * tests constructing a vector from an array
111    */
112   public void testArrayConstructor() {
113     double[] data = {2.3, 1.2, 5.0};
114     AlgVector v = new AlgVector(data);
115     assertEquals("Length␣differs", data.length, v.numElements());
116     for (int i = 0; i < data.length; i++)
117       assertEquals((i+1) + ".␣value␣differs", data[i], v.getElement(i));
118   }
119
120   /**
121    * runs tests with the given data
122    *
123    * @param data         the data to test with
```

```
124     */
125    protected void runTestOnData(Instances data) {
126      // count numeric atts
127      int numeric = 0;
128      for (int n = 0; n < data.numAttributes(); n++) {
129        if (data.attribute(n).isNumeric())
130          numeric++;
131      }
132
133      // perform tests
134      for (int n = 0; n < data.numInstances(); n++) {
135        try {
136          AlgVector v = new AlgVector(data.instance(n));
137
138          // 1. is length correct?
139          assertEquals((n+1) + ": length differs", numeric, v.numElements());
140
141          // 2. are values correct?
142          int index = 0;
143          for (int i = 0; i < data.numAttributes(); i++) {
144            if (!data.attribute(i).isNumeric())
145              continue;
146            assertEquals((n+1) + "/" + (i+1) + ": value differs", data.instance(n).value(i
                 ), v.getElement(index));
147            index++;
148          }
149
150          // 3. is instance returned correct?
151          Instance inst = v.getAsInstance(data, new Random(1));
152          for (int i = 0; i < data.numAttributes(); i++) {
153            if (!data.attribute(i).isNumeric())
154              continue;
155            assertEquals((n+1) + "/" + (i+1) + ": returned value differs", data.instance(n
                 ).value(i), inst.value(i));
156          }
157        }
158        catch (Exception e) {
159          if (!(e instanceof IllegalArgumentException))
160            fail(e.toString());
161        }
162      }
163    }
164
165    /**
166     * tests constructing a vector from a purely numeric instance
167     */
168    public void testNumericInstances() {
169      runTestOnData(generateData(0, 5, 5));
170    }
171
172    /**
173     * tests constructing a vector from a purely nominal instance
174     */
175    public void testNominalInstances() {
176      runTestOnData(generateData(5, 0, 5));
177    }
178
179    /**
180     * tests constructing a vector from a mixed instance
181     */
182    public void testMixedInstances() {
183      runTestOnData(generateData(5, 5, 5));
184    }
185
186    public static Test suite() {
```

```
187       return new TestSuite(AlgVectorTest.class);
188     }
189
190     public static void main(String[] args){
191       junit.textui.TestRunner.run(suite());
192     }
193   }
```

UtilsTest.java

```
1  /*
2   *    This program is free software: you can redistribute it and/or modify
3   *    it under the terms of the GNU General Public License as published by
4   *    the Free Software Foundation, either version 3 of the License, or
5   *    (at your option) any later version.
6   *
7   *    This program is distributed in the hope that it will be useful,
8   *    but WITHOUT ANY WARRANTY; without even the implied warranty of
9   *    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
10  *    GNU General Public License for more details.
11  *
12  *    You should have received a copy of the GNU General Public License
13  *    along with this program.  If not, see <http://www.gnu.org/licenses/>.
14  */
15
16 /*
17  * Copyright (C) 2007 University of Waikato
18  */
19
20 package weka.core;
21
22 import weka.filters.unsupervised.attribute.StringToWordVector;
23
24 import junit.framework.Test;
25 import junit.framework.TestCase;
26 import junit.framework.TestSuite;
27
28 /**
29  * Tests Utils. Run from the command line with:<p/>
30  * java weka.core.UtilsTest
31  *
32  * @author  FracPete (fracpete at waikato dot ac dot nz)
33  * @version $Revision: 11410 $
34  */
35 public class UtilsTest
36   extends TestCase {
37
38   private static final double TOLERANCE = 1e-14;
39
40   /**
41    * Constructs the <code>UtilsTest</code>.
42    *
43    * @param name        the name of the test class
44    */
45   public UtilsTest(String name) {
46     super(name);
47   }
48
49   /**
50    * Called by JUnit before each test method.
51    *
52    * @throws Exception  if an error occurs
53    */
54   protected void setUp() throws Exception {
55     super.setUp();
56   }
57
58   /**
59    * Called by JUnit after each test method
60    *
61    * @throws Exception  if an error occurs
62    */
```

```
63    protected void tearDown() throws Exception {
64      super.tearDown();
65    }
66
67    /**
68     * Tests the accuracy and behavior of the {@link Utils.variance} method.
69     */
70    public void testVariance() {
71
72      assertTrue("Incorrect␣behaviour␣when␣count␣<=␣1!",
73          Double.isNaN(Utils.variance(new double[]{})));
74      assertTrue("Incorrect␣behaviour␣when␣count␣<=␣1!",
75          Double.isNaN(Utils.variance(new double[]{3})));
76
77      checkAccuracy(generatedVar1, generatedValues1);
78      checkAccuracy(generatedVar2, generatedValues2, 1e-2);
79      checkAccuracy(generatedVar3, generatedValues3);
80      checkAccuracy(generatedVar4, generatedValues4);
81      checkAccuracy(generatedVar5, generatedValues5);
82      checkAccuracy(generatedVar6, generatedValues6, 2);
83      checkAccuracy(generatedVar7, generatedValues7);
84      checkAccuracy(generatedVar8, generatedValues8, 2);
85      checkAccuracy(generatedVar9, generatedValues9);
86      checkAccuracy(generatedVar10, generatedValues10);
87    }
88
89    private void checkAccuracy(long var, long[] values) {
90      checkAccuracy(var, values, TOLERANCE);
91    }
92
93    private void checkAccuracy(long var, long[] values, double tol) {
94      double ref = Double.longBitsToDouble(var);
95      double test = Utils.variance(convert(values));
96      assertEquals("Inaccurate␣variance␣calculation!", ref, test, Math.abs(tol*ref));
97    }
98
99    private double[] convert(long[] values) {
100     double[] valuesDouble = new double[values.length];
101     for (int i = 0; i < values.length; i++) {
102       valuesDouble[i] = Double.longBitsToDouble(values[i]);
103     }
104     return valuesDouble;
105   }
106
107   /**
108    * tests splitOptions and joinOptions
109    *
110    * @see Utils#splitOptions(String)
111    * @see Utils#joinOptions(String[])
112    */
113   public void testSplittingAndJoining() {
114     String[]    options;
115     String[]    newOptions;
116     String      joined;
117     int         i;
118
119     try {
120       options    = new StringToWordVector().getOptions();
121       joined     = Utils.joinOptions(options);
122       newOptions = Utils.splitOptions(joined);
123       assertEquals("Same␣number␣of␣options", options.length, newOptions.length);
124       for (i = 0; i < options.length; i++) {
125         if (!options[i].equals(newOptions[i]))
126           fail("Option␣" + (i+1) + "␣differs");
127       }
```

```
128       }
129       catch (Exception e) {
130         fail("Exception:␣" + e);
131       }
132     }
133
134     /**
135      * tests quote and unquote
136      *
137      * @see Utils#quote(String)
138      * @see Utils#unquote(String)
139      */
140     public void testQuoting() {
141       String      input;
142       String      output;
143
144       input  = "blahblah";
145       output = Utils.quote(input);
146       assertTrue("No␣quoting␣necessary", !output.startsWith("'") && !output.endsWith("'"))
              ;
147
148       input  = "";
149       output = Utils.quote(input);
150       assertTrue("Empty␣string␣quoted", output.startsWith("'") && output.endsWith("'"));
151       assertTrue("Empty␣string␣restored", input.equals(Utils.unquote(output)));
152
153       input  = "␣";
154       output = Utils.quote(input);
155       assertTrue("Blank␣quoted", output.startsWith("'") && output.endsWith("'"));
156       assertTrue("Blank␣restored", input.equals(Utils.unquote(output)));
157
158       input  = "{";
159       output = Utils.quote(input);
160       assertTrue(">" + input + "<␣quoted", output.startsWith("'") && output.endsWith("'"))
              ;
161       assertTrue(">" + input + "<␣restored", input.equals(Utils.unquote(output)));
162
163       input  = "}";
164       output = Utils.quote(input);
165       assertTrue(">" + input + "<␣quoted", output.startsWith("'") && output.endsWith("'"))
              ;
166       assertTrue(">" + input + "<␣restored", input.equals(Utils.unquote(output)));
167
168       input  = ",";
169       output = Utils.quote(input);
170       assertTrue(">" + input + "<␣quoted", output.startsWith("'") && output.endsWith("'"))
              ;
171       assertTrue(">" + input + "<␣restored", input.equals(Utils.unquote(output)));
172
173       input  = "?";
174       output = Utils.quote(input);
175       assertTrue(">" + input + "<␣quoted", output.startsWith("'") && output.endsWith("'"))
              ;
176       assertTrue(">" + input + "<␣restored", input.equals(Utils.unquote(output)));
177
178       input  = "\r\n\t'\"%";
179       output = Utils.quote(input);
180       assertTrue(">" + input + "<␣quoted", output.startsWith("'") && output.endsWith("'"))
              ;
181       assertTrue(">" + input + "<␣restored", input.equals(Utils.unquote(output)));
182     }
183
184     /**
185      * tests backQuoteChars and unbackQuoteChars
186      *
```

```
187     * @see Utils#backQuoteChars(String)
188     * @see Utils#unbackQuoteChars(String)
189     */
190    public void testBackQuoting() {
191      String      input;
192      String      output;
193
194      input  = "blahblah";
195      output = Utils.backQuoteChars(input);
196      assertTrue("No␣backquoting␣necessary", input.equals(output));
197
198      input  = "\r\n\t'\"%";
199      output = Utils.backQuoteChars(input);
200      assertTrue(">" + input + "<␣restored", input.equals(Utils.unbackQuoteChars(output)))
             ;
201
202      input  = "\\r\\n\\t\\'\\\"\\%";
203      output = Utils.backQuoteChars(input);
204      assertTrue(">" + input + "<␣restored", input.equals(Utils.unbackQuoteChars(output)))
             ;
205
206      input  = Utils.joinOptions(new StringToWordVector().getOptions());
207      output = Utils.backQuoteChars(input);
208      assertTrue(">" + input + "<␣restored", input.equals(Utils.unbackQuoteChars(output)))
             ;
209    }
210
211    public static Test suite() {
212      return new TestSuite(UtilsTest.class);
213    }
214
215    public static void main(String[] args){
216      junit.textui.TestRunner.run(suite());
217    }
218
219    //Generated values with parameters:
220    //Count = 100
221    //values_mean = 1.000000e+08; values_stdDev = 1.000000e+08; values_ordering = Random
222    private static final long generatedVar1 = 48463797972488880081L; // approx ~ 1.002040e
           +16
223    private static final long[] generatedValues1 = {-4502108233396493964L,
           4723728974583900862L, 4724352188189308360L, 4728115190960860804L,
           4729651617148795053L, 4725637613972895537L, 4730983312112713362L,
           4732984965466902608L, 4733348328003874764L, 4721867348739709532L,
           -4515915348804967712L, -4504475538151643424L, 4720620978861520689L,
           4729253181485441422L, 4728924315319343309L, 4730860883401456904L,
           4729624414008534920L, 4721944814894706417L, -4502718654540998012L,
           -4505760338916381024L, 4732325452438913325L, -4503995118402924976L,
           -4499171420353650292L, 4701274311466297152L, 4728911828384797328L,
           -4504173214362173328L, -4500037646772938368L, 4722659557372032140L,
           4721662633398364149L, 4726865972667491760L, 4726785560101146141L,
           -4508794473662502776L, 4729003032045369746L, 4731873093211098583L,
           4703857113551828096L, 4722843885751083471L, 4713949557851769080L,
           -4501334979361403136L, 4726971853446339409L, -4501110925457140188L,
           4728810432137718965L, 4728195121463203742L, 4729725337545897655L,
           4720378352228018261L, 4727690759409108414L, 4732055580878571254L,
           4727132649379861353L, 4726514625839732792L, 4719931711826427412L,
           4725825440236714718L, -4502653151060615824L, 4732441086397906553L,
           -4499679870917995464L, 4726125560497166743L, 4732288574983369587L,
           -4503495536230517904L, 4731674699063872490L, -4509686744296375680L,
           -4502573192816411356L, 4729128740459696730L, 4725207374250699774L,
           4729338924827077800L, 4732668192408843973L, 4732931336961679433L,
           -4504326372865511840L, -4499658508274424552L, -4499458222199525156L,
           4727707496571224177L, 4730997225583310543L, -4515381666377829712L,
           4729539307593296511L, 4704104525801341344L, 4732145306997181814L,
```

```
        4715809951873630108L, 4729311235884579739L, 4731345354127246072L,
        4725707175693491160L, 4730630464779307854L, 4730133941199409854L,
        4731328290963192479L, 4731361316139689106L, 4714698125210484040L,
        4732133740508962216L, 4729140741926166706L, -4508102649237681760L,
        4719218275732095672L, 4721787841851815648L, 4729112873433246780L,
        -45065468519586452 44L, 4733246230484764212L, 4730842947051650822L,
        -4508306899685325664L, 4729152439012063403L, 4732892633054335396L,
        4732062715198881078L, 4722256675761560790L, 4721095628932846213L,
        4718724670462699848L, 4728211531090703109L, 4704495152248023264L};

    //Generated values with parameters:
    //Count = 100
    //values_mean = 1.000000e+08; values_stdDev = 1.000000e-08; values_ordering = Random
    private static final long generatedVar2 = 43692822704839639 40L; // approx ~ 1.305129e
        -16
    private static final long[] generatedValues2 = {4726483295884279809L,
        4726483295884279808L, 4726483295884279809L, 4726483295884279807L,
        4726483295884279809L, 4726483295884279808L, 4726483295884279807L,
        4726483295884279809L, 4726483295884279808L, 4726483295884279808L,
        4726483295884279807L, 4726483295884279807L, 4726483295884279807L,
        4726483295884279807L, 4726483295884279809L, 4726483295884279809L,
        4726483295884279808L, 4726483295884279809L, 4726483295884279808L,
        4726483295884279808L, 4726483295884279809L, 4726483295884279808L,
        4726483295884279809L, 4726483295884279809L, 4726483295884279808L,
        4726483295884279809L, 4726483295884279807L, 4726483295884279809L,
        4726483295884279809L, 4726483295884279809L, 4726483295884279809L,
        4726483295884279808L, 4726483295884279807L, 4726483295884279808L,
        4726483295884279809L, 4726483295884279807L, 4726483295884279809L,
        4726483295884279809L, 4726483295884279807L, 4726483295884279807L,
        4726483295884279807L, 4726483295884279808L, 4726483295884279808L,
        4726483295884279808L, 4726483295884279808L, 4726483295884279808L,
        4726483295884279807L, 4726483295884279808L, 4726483295884279808L,
        4726483295884279807L, 4726483295884279808L, 4726483295884279809L,
        4726483295884279809L, 4726483295884279808L, 4726483295884279808L,
        4726483295884279808L, 4726483295884279807L, 4726483295884279807L,
        4726483295884279808L, 4726483295884279808L, 4726483295884279809L,
        4726483295884279807L, 4726483295884279809L, 4726483295884279809L,
        4726483295884279809L, 4726483295884279809L, 4726483295884279809L,
        4726483295884279808L, 4726483295884279808L, 4726483295884279808L,
        4726483295884279809L, 4726483295884279808L, 4726483295884279808L,
        4726483295884279807L, 4726483295884279808L, 4726483295884279809L,
        4726483295884279808L, 4726483295884279808L, 4726483295884279808L,
        4726483295884279807L, 4726483295884279807L, 4726483295884279808L,
        4726483295884279807L, 4726483295884279807L, 4726483295884279808L,
        4726483295884279808L, 4726483295884279808L, 4726483295884279809L,
        4726483295884279809L, 4726483295884279809L, 4726483295884279809L,
        4726483295884279808L, 4726483295884279809L, 4726483295884279808L,
        4726483295884279807L, 4726483295884279809L, 4726483295884279808L};

    //Generated values with parameters:
    //Count = 100
    //values_mean = -1.000000e-08; values_stdDev = 1.000000e+08; values_ordering = Random
    private static final long generatedVar3 = 484457087973051160 7L; // approx ~ 7.704880e
        +15
```

continues with the declaration of 8 more data sets

Modified Test Files

AlgVectorTest.java

```
1  /*
2   *    This program is free software: you can redistribute it and/or modify
3   *    it under the terms of the GNU General Public License as published by
4   *    the Free Software Foundation, either version 3 of the License, or
5   *    (at your option) any later version.
6   *
7   *    This program is distributed in the hope that it will be useful,
8   *    but WITHOUT ANY WARRANTY; without even the implied warranty of
9   *    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
10  *    GNU General Public License for more details.
11  *
12  *    You should have received a copy of the GNU General Public License
13  *    along with this program.  If not, see <http://www.gnu.org/licenses/>.
14  */
15
16 /*
17  * Copyright (C) 2007 University of Waikato
18  */
19
20 package weka.core.dscribe;
21
22 import java.util.Random;
23
24 import junit.framework.Test;
25 import junit.framework.TestCase;
26 import junit.framework.TestSuite;
27 import weka.core.AlgVector;
28 import weka.core.Instance;
29 import weka.core.Instances;
30 import weka.core.TestInstances;
31
32 /**
33  * Tests AlgVector. Run from the command line with:
34  * <p/>
35  * java weka.core.AlgVectorTest
36  *
37  * @author FracPete (fracpete at waikato dot ac dot nz)
38  * @version $Revision: 8034 $
39  */
40 public class AlgVectorTest extends TestCase {
41
42   /** for generating the datasets */
43   protected Random m_Random;
44
45   /**
46    * Constructs the <code>AlgVectorTest</code>.
47    *
48    * @param name the name of the test class
49    */
50   public AlgVectorTest(String name) {
51     super(name);
52   }
53
54   /**
55    * Called by JUnit before each test method.
56    *
57    * @throws Exception if an error occurs
58    */
```

```
59    @Override
60    protected void setUp() throws Exception {
61      super.setUp();
62
63      m_Random = new Random(1);
64    }
65
66    /**
67     * Called by JUnit after each test method
68     *
69     * @throws Exception if an error occurs
70     */
71    @Override
72    protected void tearDown() throws Exception {
73      super.tearDown();
74
75      m_Random = null;
76    }
77
78    /**
79     * generates data with the given amount of nominal and numeric attributes
80     *
81     * @param nominal the number of nominal attributes
82     * @param numeric the number of numeric attributes
83     * @param rows    the number of rows to generate
84     * @return the generated data
85     */
86    protected Instances generateData(int nominal, int numeric, int rows) {
87      Instances result;
88      TestInstances test;
89
90      test = new TestInstances();
91      test.setClassIndex(TestInstances.NO_CLASS);
92      test.setNumNominal(nominal);
93      test.setNumNumeric(numeric);
94      test.setNumInstances(rows);
95
96      try {
97        result = test.generate();
98      } catch (Exception e) {
99        result = null;
100     }
101     return result;
102   }
103
104   /**
105    * tests constructing a vector with a given length
106    */
107   public void testAlgVector_int() {
108     int input = 22;
109     AlgVector instance = new AlgVector(input);
110     assertEquals("Length␣differs", input, instance.numElements());
111   }
112
113   /**
114    * tests constructing a vector from an array
115    */
116   public void testAlgVector_doubleArray() {
117     double[] input = { 2.3, 1.2, 5.0 };
118     AlgVector instance = new AlgVector(input);
119     assertSameContent(input, instance);
120   }
121
122   private void assertSameContent(Instance input, AlgVector instance) {
123     assertEquals("length␣differs", input.numAttributes(), instance.numElements());
```

```
124       for (int i = 0; i < input.numAttributes(); i++) {
125         assertEquals("value␣differs", input.value(i), instance.getElement(i));
126       }
127     }
128
129     private void assertSameContent(double[] input, AlgVector instance) {
130       assertEquals("length␣differs", input.length, instance.numElements());
131       for (int i = 0; i < input.length; i++) {
132         assertEquals("value␣differs", input[i], instance.getElement(i));
133       }
134     }
135
136     /**
137      * tests constructing a vector from a purely numeric instance
138      */
139     public void testAlgVector_Instance_WithPurelyNumeric() throws Exception {
140       Instance input = generateData(0, 5, 1).instance(0);
141       AlgVector instance = new AlgVector(input);
142       assertSameContent(input, instance);
143     }
144
145     public void testGetAsInstance_WithPurelyNumeric() throws Exception {
146       Instances data = generateData(0, 5, 1);
147       Instance expected = data.instance(0);
148       AlgVector instance = new AlgVector(expected);
149       Instance output = instance.getAsInstance(data, new Random(1));
150       for (int i = 0; i < expected.numAttributes(); i++) {
151         assertEquals("returned␣value␣differs", expected.value(i), output.value(i));
152       }
153     }
154
155     /**
156      * tests constructing a vector from a purely nominal instance
157      */
158     public void testAlgVector_Instance_WithPurelyNominal() throws Exception {
159       try {
160         new AlgVector(generateData(5, 0, 1).instance(0));
161         fail("no␣exception");
162       } catch (IllegalArgumentException expected) {
163       }
164     }
165
166     public void testAlgVector_Instance_WithPurelyNominal_Repeat() throws Exception {
167       for (int i = 0; i < 5; i++) {
168         testAlgVector_Instance_WithPurelyNominal();
169       }
170     }
171
172     /**
173      * tests constructing a vector from a mixed instance
174      *
175      * @throws Exception
176      */
177     public void testAlgVector_Instance_WithMixed_ShouldRemoveNonNumericValues() throws
         Exception {
178       int numeric = 5;
179       Instance input = generateData(5, numeric, 1).instance(0);
180       AlgVector instance = new AlgVector(input);
181       assertEquals("length␣differs", numeric, instance.numElements());
182       int instanceIndex = 0;
183       for (int i = 0; i < input.numAttributes(); i++) {
184         if (!input.attribute(i).isNumeric()) {
185           continue;
186         }
187         assertEquals("value␣differs", input.value(i), instance.getElement(instanceIndex));
```

```
188         instanceIndex++;
189       }
190     }
191
192     public void testGetAsInstance_WithMixed() throws Exception {
193       Instances data = generateData(5, 5, 1);
194       Instance expected = data.instance(0);
195       AlgVector instance = new AlgVector(expected);
196       Instance output = instance.getAsInstance(data, new Random(1));
197       for (int i = 0; i < expected.numAttributes(); i++) {
198         if (!expected.attribute(i).isNumeric()) {
199           continue;
200         }
201         assertEquals("returned␣value␣differs", expected.value(i), output.value(i));
202       }
203     }
204
205     public void testAlgVector_Instance_Repeat() throws Exception {
206       for (int i = 0; i < 5; i++) {
207         testAlgVector_Instance_WithPurelyNumeric();
208         testGetAsInstance_WithPurelyNumeric();
209       }
210       for (int i = 0; i < 5; i++) {
211         testAlgVector_Instance_WithMixed_ShouldRemoveNonNumericValues();
212         testGetAsInstance_WithMixed();
213       }
214     }
215
216     public static Test suite() {
217       return new TestSuite(AlgVectorTest.class);
218     }
219
220     public static void main(String[] args) {
221       junit.textui.TestRunner.run(suite());
222     }
223 }
```

UtilsTest.java

```
1  /*
2   *    This program is free software: you can redistribute it and/or modify
3   *    it under the terms of the GNU General Public License as published by
4   *    the Free Software Foundation, either version 3 of the License, or
5   *    (at your option) any later version.
6   *
7   *    This program is distributed in the hope that it will be useful,
8   *    but WITHOUT ANY WARRANTY; without even the implied warranty of
9   *    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
10  *    GNU General Public License for more details.
11  *
12  *    You should have received a copy of the GNU General Public License
13  *    along with this program.  If not, see <http://www.gnu.org/licenses/>.
14  */
15
16 /*
17  * Copyright (C) 2007 University of Waikato
18  */
19
20 package weka.core.dscribe;
21
22 import junit.framework.Test;
23 import junit.framework.TestCase;
24 import junit.framework.TestSuite;
25 import weka.core.Utils;
26 import weka.filters.unsupervised.attribute.StringToWordVector;
27
28 /**
29  * Tests Utils. Run from the command line with:
30  * <p/>
31  * java weka.core.UtilsTest
32  *
33  * @author FracPete (fracpete at waikato dot ac dot nz)
34  * @version $Revision: 11410 $
35  */
36 public class UtilsTest extends TestCase {
37
38   private static final double TOLERANCE = 1e-14;
39
40   /**
41    * Constructs the <code>UtilsTest</code>.
42    *
43    * @param name the name of the test class
44    */
45   public UtilsTest(String name) {
46     super(name);
47   }
48
49   /**
50    * Called by JUnit before each test method.
51    *
52    * @throws Exception if an error occurs
53    */
54   @Override
55   protected void setUp() throws Exception {
56     super.setUp();
57   }
58
59   /**
60    * Called by JUnit after each test method
61    *
62    * @throws Exception if an error occurs
```

```
63     */
64    @Override
65    protected void tearDown() throws Exception {
66      super.tearDown();
67    }
68
69    /**
70     * Tests the accuracy and behavior of the {@link Utils.variance} method.
71     */
72    public void testTypicalVariance() {
73
74      checkAccuracy(generatedVar1, generatedValues1);
75      checkAccuracy(generatedVar2, generatedValues2, 1e-2);
76      checkAccuracy(generatedVar3, generatedValues3);
77      checkAccuracy(generatedVar4, generatedValues4);
78      checkAccuracy(generatedVar5, generatedValues5);
79      checkAccuracy(generatedVar6, generatedValues6, 2);
80      checkAccuracy(generatedVar7, generatedValues7);
81      checkAccuracy(generatedVar8, generatedValues8, 2);
82      checkAccuracy(generatedVar9, generatedValues9);
83      checkAccuracy(generatedVar10, generatedValues10);
84    }
85
86    public void testVariance_WithZeroValue() {
87      assertEquals("Incorrect␣behaviour␣when␣count␣<=␣1!", Double.NaN, Utils.variance(new
            double[] {}));
88    }
89
90    public void testVariance_WithOneValue() {
91      assertEquals("Incorrect␣behaviour␣when␣count␣<=␣1!", Double.NaN, Utils.variance(new
            double[] { 3 }));
92    }
93
94    private void checkAccuracy(long var, long[] values) {
95      checkAccuracy(var, values, TOLERANCE);
96    }
97
98    private void checkAccuracy(long var, long[] values, double tol) {
99      double ref = Double.longBitsToDouble(var);
100     double test = Utils.variance(convert(values));
101     assertEquals("Inaccurate␣variance␣calculation!", ref, test, Math.abs(tol * ref));
102   }
103
104   private double[] convert(long[] values) {
105     double[] valuesDouble = new double[values.length];
106     for (int i = 0; i < values.length; i++) {
107       valuesDouble[i] = Double.longBitsToDouble(values[i]);
108     }
109     return valuesDouble;
110   }
111
112   /**
113    * tests splitOptions and joinOptions
114    *
115    * @see Utils#splitOptions(String)
116    * @see Utils#joinOptions(String[])
117    */
118   public void testSplitOptionsAndJoinOptions() throws Exception {
119     String[] input;
120     String[] back;
121     String output;
122
123     input = new StringToWordVector().getOptions();
124     output = Utils.joinOptions(input);
125     back = Utils.splitOptions(output);
```

```
126       assertSameContent(input, back);
127     }
128
129     private void assertSameContent(String[] input, String[] back) {
130       int i;
131       assertEquals("Same␣number␣of␣options", input.length, back.length);
132       for (i = 0; i < input.length; i++) {
133         if (!input[i].equals(back[i]))
134           fail("Option␣" + (i + 1) + "␣differs");
135       }
136     }
137
138     /**
139      * tests quote and unquote
140      *
141      * @see Utils#quote(String)
142      * @see Utils#unquote(String)
143      */
144     public void testQuoteAndUnquote() {
145       String input;
146       String output;
147       String back;
148
149       input = "\r\n\t'\"%";
150       output = Utils.quote(input);
151       back = Utils.unquote(output);
152       assertEquals(input, back);
153     }
154
155     public void testTypicalQuoting() {
156       String input;
157       String output;
158
159       input = "blahblah";
160       output = Utils.quote(input);
161       assertTrue("No␣quoting␣necessary", !output.startsWith("'") && !output.endsWith("'"))
              ;
162
163       input = "";
164       output = Utils.quote(input);
165       assertTrue("Empty␣string␣quoted", output.startsWith("'") && output.endsWith("'"));
166       assertTrue("Empty␣string␣restored", input.equals(Utils.unquote(output)));
167
168       input = "␣";
169       output = Utils.quote(input);
170       assertTrue("Blank␣quoted", output.startsWith("'") && output.endsWith("'"));
171       assertTrue("Blank␣restored", input.equals(Utils.unquote(output)));
172
173       input = "{";
174       output = Utils.quote(input);
175       assertTrue(">" + input + "<␣quoted", output.startsWith("'") && output.endsWith("'"))
              ;
176       assertTrue(">" + input + "<␣restored", input.equals(Utils.unquote(output)));
177
178       input = "}";
179       output = Utils.quote(input);
180       assertTrue(">" + input + "<␣quoted", output.startsWith("'") && output.endsWith("'"))
              ;
181       assertTrue(">" + input + "<␣restored", input.equals(Utils.unquote(output)));
182
183       input = ",";
184       output = Utils.quote(input);
185       assertTrue(">" + input + "<␣quoted", output.startsWith("'") && output.endsWith("'"))
              ;
186       assertTrue(">" + input + "<␣restored", input.equals(Utils.unquote(output)));
```

```
187
188       input = "?";
189       output = Utils.quote(input);
190       assertTrue(">" + input + "<␣quoted", output.startsWith("'") && output.endsWith("'"))
              ;
191       assertTrue(">" + input + "<␣restored", input.equals(Utils.unquote(output)));
192
193       input = "\r\n\t'\"%";
194       output = Utils.quote(input);
195       assertTrue(">" + input + "<␣quoted", output.startsWith("'") && output.endsWith("'"))
              ;
196       assertTrue(">" + input + "<␣restored", input.equals(Utils.unquote(output)));
197     }
198
199     /**
200      * tests backQuoteChars and unbackQuoteChars
201      *
202      * @see Utils#backQuoteChars(String)
203      * @see Utils#unbackQuoteChars(String)
204      */
205     public void testBackQuoteCharsAndUnbackQuoteChars() {
206       String input;
207       String output;
208       String back;
209
210       input = Utils.joinOptions(new StringToWordVector().getOptions());
211       output = Utils.backQuoteChars(input);
212       back = Utils.unbackQuoteChars(output);
213       assertEquals(input, back);
214     }
215
216     public void testTypicalBackQuoting() {
217       String input;
218       String output;
219
220       input = "blahblah";
221       output = Utils.backQuoteChars(input);
222       assertTrue("No␣backquoting␣necessary", input.equals(output));
223
224       input = "\r\n\t'\"%";
225       output = Utils.backQuoteChars(input);
226       assertTrue(">" + input + "<␣restored", input.equals(Utils.unbackQuoteChars(output)))
              ;
227
228       input = "\\r\\n\\t\\'\\\"\\%";
229       output = Utils.backQuoteChars(input);
230       assertTrue(">" + input + "<␣restored", input.equals(Utils.unbackQuoteChars(output)))
              ;
231
232       input = Utils.joinOptions(new StringToWordVector().getOptions());
233       output = Utils.backQuoteChars(input);
234       assertTrue(">" + input + "<␣restored", input.equals(Utils.unbackQuoteChars(output)))
              ;
235     }
236
237     public static Test suite() {
238       return new TestSuite(UtilsTest.class);
239     }
240
241     public static void main(String[] args) {
242       junit.textui.TestRunner.run(suite());
243     }
244
245     // Generated values with parameters:
246     // Count = 100
```

```
247     // values_mean = 1.000000e+08; values_stdDev = 1.000000e+08; values_ordering =
248     // Random
249     private static final long generatedVar1 = 4846379797248880081L; // approx ~ 1.002040e
            +16
250     private static final long[] generatedValues1 = { -4502108233396493964L,
            4723728974583900862L, 4724352188189308360L, 4728115190960860804L,
            4729651617148795053L, 4725637613972895537L, 4730983312112713362L,
            4732984965466902608L, 4733348328003874764L, 4721867348739709532L,
            -4515915348804967712L, -4504475538151643424L, 4720620978861520689L,
            4729253181485441422L, 4728924315319343309L, 4730860883401456904L,
            4729624414008534920L, 4721944814894706417L, -4502718654540998012L,
            -4505760338916381024L, 4732325452438913325L, -4503995118402924976L,
            -4499171420353650292L, 4701274311466297152L, 4728911828384797328L,
            -4504173214362173328L, -4500037646772938368L, 4722659557372032140L,
            4721662633398364149L, 4726865972667491760L, 4726785560101146141L,
            -4508794473662502776L, 4729003032045369746L, 4731873093211098583L,
            4703857113551828096L, 4722843885751083471L, 4713949557851769080L,
            -4501334979361403136L, 4726971853446339409L, -4501110925457140188L,
            4728810432137718965L, 4728195121463203742L, 4729725337545897655L,
            4720378352228018261L, 4727690759409108414L, 4732055580878571254L,
            4727132649379861353L, 4726514625839732792L, 4719931711826427412L,
            4725825440236714718L, -4502653151060615824L, 4732441086397906553L,
            -4499679870917995464L, 4726125560497166743L, 4732288574983369587L,
            -4503495536230517904L, 4731674699063872490L, -4509686744296375680L,
            -4502573192816411356L, 4729128740459696730L, 4725207374250699774L,
            4729338924827077800L, 4732668192408843973L, 4732931336961679433L,
            -4504326372865511840L, -4499658508274424552L, -4499458222199525156L,
            4727707496571224177L, 4730997225583310543L, -4515381666377829712L,
            4729539307593296511L, 4704104525801341344L, 4732145306997181814L,
            4715809951873630108L, 4729311235884579739L, 4731345354127246072L,
            4725707175693491160L, 4730630464779307854L, 4730133941199409854L,
            4731328290963192479L, 4731361316139689106L, 4714698125210484040L,
            4732133740508962216L, 4729140741926166706L, -4508102649237681760L,
            4719218275732095672L, 4721787841851815648L, 4729112873433246780L,
            -4506546851958645244L, 4733246230484764212L, 4730842947051650822L,
            -4508306899685325664L, 4729152439012063403L, 4732892633054335396L,
            4732062715198881078L, 4722256675761560790L, 4721095628932846213L,
            4718724670462699848L, 4728211531090703109L, 4704495152248023264L };
251
252     // Generated values with parameters:
253     // Count = 100
254     // values_mean = 1.000000e+08; values_stdDev = 1.000000e-08; values_ordering =
255     // Random
256     private static final long generatedVar2 = 4369282270483963940L; // approx ~ 1.305129e
            -16
257     private static final long[] generatedValues2 = { 4726483295884279809L,
            4726483295884279808L, 4726483295884279809L, 4726483295884279807L,
            4726483295884279809L, 4726483295884279808L, 4726483295884279807L,
            4726483295884279809L, 4726483295884279808L, 4726483295884279808L,
            4726483295884279807L, 4726483295884279807L, 4726483295884279807L,
            4726483295884279807L, 4726483295884279809L, 4726483295884279809L,
            4726483295884279808L, 4726483295884279809L, 4726483295884279808L,
            4726483295884279808L, 4726483295884279809L, 4726483295884279808L,
            4726483295884279809L, 4726483295884279809L, 4726483295884279808L,
            4726483295884279809L, 4726483295884279807L, 4726483295884279809L,
            4726483295884279809L, 4726483295884279809L, 4726483295884279809L,
            4726483295884279808L, 4726483295884279807L, 4726483295884279808L,
            4726483295884279809L, 4726483295884279807L, 4726483295884279809L,
            4726483295884279809L, 4726483295884279807L, 4726483295884279807L,
            4726483295884279807L, 4726483295884279807L, 4726483295884279808L,
            4726483295884279808L, 4726483295884279808L, 4726483295884279808L,
            4726483295884279807L, 4726483295884279808L, 4726483295884279808L,
            4726483295884279807L, 4726483295884279808L, 4726483295884279809L,
            4726483295884279809L, 4726483295884279808L, 4726483295884279808L,
            4726483295884279808L, 4726483295884279807L, 4726483295884279807L,
```

```
            4726483295884279808L, 4726483295884279808L, 4726483295884279809L,
            4726483295884279808L, 4726483295884279807L, 4726483295884279808L,
            4726483295884279807L, 4726483295884279809L, 4726483295884279809L,
            4726483295884279809L, 4726483295884279809L, 4726483295884279809L,
            4726483295884279808L, 4726483295884279808L, 4726483295884279808L,
            4726483295884279809L, 4726483295884279808L, 4726483295884279808L,
            4726483295884279807L, 4726483295884279808L, 4726483295884279809L,
            4726483295884279808L, 4726483295884279808L, 4726483295884279808L,
            4726483295884279807L, 4726483295884279807L, 4726483295884279808L,
            4726483295884279807L, 4726483295884279807L, 4726483295884279808L,
            4726483295884279808L, 4726483295884279808L, 4726483295884279809L,
            4726483295884279809L, 4726483295884279809L, 4726483295884279809L,
            4726483295884279808L, 4726483295884279809L, 4726483295884279808L,
            4726483295884279807L, 4726483295884279809L, 4726483295884279808L };
258
259     // Generated values with parameters:
260     // Count = 100
261     // values_mean = -1.000000e-08; values_stdDev = 1.000000e+08; values_ordering =
262     // Random
263     private static final long generatedVar3 = 4844570879730511607L; // approx ~ 7.704880e
            +15
```

continues with the declaration of 8 more data sets

Production File

AlgVector.java

```
43    /**
44     * Constructs a vector and initializes it with default values.
45     * @param n                    the number of elements
46     * @autogen AFTER calling this method, the value of "numElements" is n.
47     */
48    public AlgVector(int n) {
49
50      m_Elements = new double[n];
51      initialize();
52    }
```

...

```
54   /**
55   * Constructs a vector using a given array.
56   * @param array          the values of the matrix
57   * @autogen AFTER calling this method, the content of the object is the same as array.
58   */
59    public AlgVector(double[] array) {
60
61      m_Elements = new double[array.length];
62      for (int i = 0; i < array.length; i++) {
63        m_Elements[i] = array[i];
64      }
65    }
```

...

```
89    /**
90     * Constructs a vector using an instance. The vector has an element for each numerical
           attribute. The other attributes (nominal, string) are ignored.
91     * @param instance    with numeric attributes, that AlgVector gets build from
92     * @throws Exception          if instance doesn't have access to the data format or no
           numeric attributes in the data
93     * @autogen AFTER calling this method, the content of the object is the same as
         instance.
94     * @autogen IF the input is mixed, THEN remove non numeric values.
95     * @autogen IF instance is purely nominal, THEN it throws a IllegalArgumentException.
96     */
97    public AlgVector(Instance instance) throws Exception {
98
99      int len = instance.numAttributes();
100     for (int i = 0; i < instance.numAttributes(); i++) {
101       if (!instance.attribute(i).isNumeric())
102         len--;
103     }
104     if (len > 0) {
105       m_Elements = new double[len];
106       int n = 0;
107       for (int i = 0; i < instance.numAttributes(); i++) {
108         if (!instance.attribute(i).isNumeric())
109           continue;
110         m_Elements[n] = instance.value(i);
111         n++;
112       }
```

```
113       }
114       else {
115         throw new IllegalArgumentException("No␣numeric␣attributes␣in␣data!");
116       }
117     }
```

Utils.java

```
640   /**
641    * Quotes a string if it contains special characters. The following rules are applied:
           A character is backquoted version of it is one of <tt>" ' % \ \n \r \t</tt> . A
         string is enclosed within single quotes if a character has been backquoted using
         the previous rule above or contains <tt>{ }</tt> or is exactly equal to the
         strings <tt>, ? space or ""</tt> (empty string). A quoted question mark
         distinguishes it from the missing value which is represented as an unquoted
         question mark in arff files.
642    * @param string    the string to be quoted
643    * @return    the string (possibly quoted)
644    * @see #unquote(String)
645    * @autogen IF the input to unquote is the output of quote, THEN its output is the
         original input of quote.
646    */
647   public static/* @pure@ */String quote(String string) {
648     boolean quote = false;
649
650     // backquote the following characters
651     if ((string.indexOf('\n') != -1) || (string.indexOf('\r') != -1)
652       || (string.indexOf('\'') != -1) || (string.indexOf('"') != -1)
653       || (string.indexOf('\\') != -1) || (string.indexOf('\t') != -1)
654       || (string.indexOf('%') != -1) || (string.indexOf('\u001E') != -1)) {
655       string = backQuoteChars(string);
656       quote = true;
657     }
658
659     // Enclose the string in 's if the string contains a recently added
660     // backquote or contains one of the following characters.
661     if ((quote == true) || (string.indexOf('{') != -1)
662       || (string.indexOf('}') != -1) || (string.indexOf(',') != -1)
663       || (string.equals("?")) || (string.indexOf('␣') != -1)
664       || (string.equals(""))) {
665       string = ("'".concat(string)).concat("'");
666     }
667
668     return string;
669   }
```

...

```
671   /**
672    * unquotes are previously quoted string (but only if necessary), i.e., it removes the
         single quotes around it. Inverse to quote(String).
673    * @param string    the string to process
674    * @return    the unquoted string
675    * @see #quote(String)
676    * @autogen IF the input to unquote is the output of quote, THEN its output is the
         original input of quote.
677    */
678   public static String unquote(String string) {
679     if (string.startsWith("'") && string.endsWith("'")) {
680       string = string.substring(1, string.length() - 1);
681
682       if ((string.indexOf("\\n") != -1) || (string.indexOf("\\r") != -1)
683         || (string.indexOf("\\'") != -1) || (string.indexOf("\\\"") != -1)
684         || (string.indexOf("\\\\") != -1) || (string.indexOf("\\t") != -1)
685         || (string.indexOf("\\%") != -1) || (string.indexOf("\\u001E") != -1)) {
686         string = unbackQuoteChars(string);
687       }
688     }
689
```

```
690       return string;
691     }
```

...

```
842     /**
843      * The inverse operation of backQuoteChars(). Converts back-quoted carriage returns
             and new lines in a string to the corresponding character ('\r' and '\n'). Also "
             un"-back-quotes the following characters: ` " | \t and %
844      * @param string    the string
845      * @return    the converted string
846      * @see #backQuoteChars(String)
847      * @autogen IF the input to unbackQuoteChars is the output of backQuoteChars, THEN its
             output is the original input of backQuoteChars.
848      */
849     public static String unbackQuoteChars(String string) {
850
851       String charsFind[] = { "\\\\", "\\'", "\\t", "\\n", "\\r", "\\\"", "\\%", "\\u001E"
             };
852       char charsReplace[] = { '\\', '\'', '\t', '\n', '\r', '"', '%', '\u001E' };
853       return replaceStrings(string, charsFind, charsReplace);
854     }
```

...

```
901     /**
902      * Split up a string containing options into an array of strings, one for each option.
903      * @param quotedOptionString     the string containing the options
904      * @return     the array of options
905      * @throws Exception     in case of an unterminated string, unknown character or a
             parse error
906      * @autogen IF the input to splitOptions is the output of joinOptions, THEN its output
             is the original input of joinOptions.
907      */
908     public static String[] splitOptions(String quotedOptionString)
909             throws Exception {
910
911       return splitOptions(quotedOptionString, null, null);
912     }
913
914     /**
915      * Split up a string containing options into an array of strings, one for each option.
             If either the second or the third argument are null, the method unbackQuoteChars
             () is applied to each individual option string. Otherwise, the method
             replaceStrings() is applied to each individual option string, using the second
             and third argument of this method as parameters.
916      * @param quotedOptionString     the string containing the options
917      * @param toReplace     strings to replace in each option (e.g., backquoted characters
             )
918      * @param replacements     the characters to replace the strings with
919      * @return     the array of options
920      * @throws Exception     in case of an unterminated string, unknown character or a
             parse error
921      * @autogen IF the input to splitOptions is the output of joinOptions, THEN its output
             is the original input of joinOptions.
922      */
923     public static String[] splitOptions(String quotedOptionString, String[] toReplace,
         char[] replacements)
924             throws Exception {
925
926       Vector<String> optionsVec = new Vector<String>();
927       String str = new String(quotedOptionString);
```

```
928       int i;
929
930       while (true) {
931
932         // trimLeft
933         i = 0;
934         while ((i < str.length()) && (Character.isWhitespace(str.charAt(i)))) {
935           i++;
936         }
937         str = str.substring(i);
938
939         // stop when str is empty
940         if (str.length() == 0) {
941           break;
942         }
943
944         // if str start with a double quote
945         if (str.charAt(0) == '"') {
946
947           // find the first not anti-slached double quote
948           i = 1;
949           while (i < str.length()) {
950             if (str.charAt(i) == str.charAt(0)) {
951               break;
952             }
953             if (str.charAt(i) == '\\') {
954               i += 1;
955               if (i >= str.length()) {
956                 throw new Exception("String␣should␣not␣finish␣with␣\\");
957               }
958             }
959             i += 1;
960           }
961           if (i >= str.length()) {
962             throw new Exception("Quote␣parse␣error.");
963           }
964
965           // add the founded string to the option vector (without quotes)
966           String optStr = str.substring(1, i);
967           if ((toReplace != null) && (replacements != null)) {
968             optStr = replaceStrings(optStr, toReplace, replacements);
969           } else {
970             optStr = unbackQuoteChars(optStr);
971           }
972           optionsVec.addElement(optStr);
973           str = str.substring(i + 1);
974         } else {
975           // find first whiteSpace
976           i = 0;
977           while ((i < str.length()) && (!Character.isWhitespace(str.charAt(i)))) {
978             i++;
979           }
980
981           // add the founded string to the option vector
982           String optStr = str.substring(0, i);
983           optionsVec.addElement(optStr);
984           str = str.substring(i);
985         }
986       }
987
988       // convert optionsVec to an array of String
989       String[] options = new String[optionsVec.size()];
990       for (i = 0; i < optionsVec.size(); i++) {
991         options[i] = optionsVec.elementAt(i);
992       }
```

```
993       return options;
994     }
```

...

```
996     /**
997      * Joins all the options in an option array into a single string, as might be used on
             the command line.
998      * @param optionArray      the array of options
999      * @return      the string containing all options.
1000     * @autogen IF the input to splitOptions is the output of joinOptions, THEN its output
             is the original input of joinOptions.
1001     */
1002    public static String joinOptions(String[] optionArray) {
1003
1004      String optionString = "";
1005      for (String element : optionArray) {
1006        if (element.equals("")) {
1007          continue;
1008        }
1009        boolean escape = false;
1010        for (int n = 0; n < element.length(); n++) {
1011          if (Character.isWhitespace(element.charAt(n))
1012            || element.charAt(n) == '"' || element.charAt(n) == '\'') {
1013            escape = true;
1014            break;
1015          }
1016        }
1017        if (escape) {
1018          optionString += '"' + backQuoteChars(element) + '"';
1019        } else {
1020          optionString += element;
1021        }
1022        optionString += "␣";
1023      }
1024      return optionString.trim();
1025    }
```

...

```
1605    /**
1606     * Computes the variance for an array of doubles.
1607     * @param vector                          the array
1608     * @return                          the variance
1609     * @autogen IF vector is one value or zero value, THEN returned value is Double.NaN.
1610     */
1611    public static/* @pure@ */double variance(double[] vector) {
1612
1613      if (vector.length <= 1)
1614        return Double.NaN;
1615
1616      double mean = 0;
1617      double var = 0;
1618
1619      for (int i = 0; i < vector.length; i++) {
1620        double delta = vector[i] - mean;
1621        mean += delta / (i + 1);
1622        var += (vector[i] - mean) * delta;
1623      }
1624
1625      var /= vector.length - 1;
1626
```

```
1627      // We don't like negative variance
1628      if (var < 0) {
1629        return 0;
1630      } else {
1631        return var;
1632      }
1633    }
```

Configuration File

config.dscribe

```
FOCAL_METHOD

[package].dscribe.[class]Test#test[method]_[argtype]_[state]_[expected]
[package].dscribe.[class]Test#test[method]_[argtype]_[state]
[package].dscribe.[class]Test#test[method]_[argtype]
[package].dscribe.[class]Test#test[method]_[state]
[package].dscribe.[class]Test#test[method]And[method]

SHAPES

oneArg(Signature): method.argnames=[value]

withInput(Name): state=With[value]
expectedInName(Name): expected=Should[value]

tryFail(Code):
        try {
                new Constructor/*method*/(/*...*/);
                fail(/*...*/);
        }
        catch (Exception/*type*/ expected/*e*/) {}

inputBack(Code):
        // ...
        assertEquals(/*...*/ input, instance.get/*getter*/());

sameContent(Code):
        // ...
        assertSameContent(input, instance);

directValue(Code):
        assertEquals(/*...*/ x/*value*/, focal/*method*/(/*...*/));

doUndo(Code):
        // ...
        output = call1/*do*/(input);
        back = call2/*undo*/(output);
        assertSameContent(input, back);

doUndo(Code):
        // ...
        output = call1/*do*/(input);
        back = call2/*undo*/(output);
        assertEquals(input, back);

CONSTRUCTS

oneArg + inputBack -> (,, the value of "[inputBack.getter]", [oneArg.value])
withInput + oneArg + tryFail -> ([oneArg.value], [withInput.value], , it throws a [
    tryFail.type])
withInput + expectedInName -> (the input, [withInput.value], , [expectedInName.value])
oneArg + sameContent -> (,, the content of the object, the same as [oneArg.value])
oneArg + withInput + directValue -> ([oneArg.value], [withInput.value], returned value,
    [directValue.value])
doUndo -> (the input to [doUndo.undo], the output of [doUndo.do], its output, the
    original input of [doUndo.do])

PLACEHOLDERS
```

```
56
57  state: (With|When)[A-Z][A-Za-z0-9]*
58  expected: Should[A-Z][A-Za-z0-9]*
59  value: .+
60  modes: (Structure)?(NoClass)?
```

Appendix B

Codes Assigned to Each Test from the Case Study

This appendix presents the codes produced during the second phase of our case study, assigned to each unit test in our sample. The closed set of codes is the following, distributed along eight dimensions:

Unit: What is the "unit" under test?

- *Mo* Interaction between members of two or more classes
- *C* Whole class, or two or more methods of the same class
- *Me* Single method, in different contexts
- *P* Single method, in a single context
- *A* Static or instance variable

Purpose: What is the general purpose or objective of the test? One code must be selected. In the case of ambiguity, make the best appxoximation.

- *N* Behavior under typical circumstances
- *C* Corner case or unintuitive behavior
- *E* Exception handling or other unintended cases

I Interaction between the unit and other components of the system

B Addresses a bug described in an issue tracker

P Performance testing

S Stress testing

R General regression test

Name: Is the name of the unit test well-structured?

S Well-delimited structure including focal unit and other elements

F Well-delimited structure indicating only the focal unit

A Close to a well-delimited structure, but with small variations, such as abbreviations

H No discernible structure

Variables: Are there recurrent local variable names that could be used to understand the unit test? Select as many codes as necessary. Indicate near matches with ~.

I input

O output

E expected

B back

R result

In instance

Helpers: Are helper methods used in the test? Choose as many as applicable.

N No helper method used in the test

As Helper method encapsulating an assertion

Ac Helper method encapsulating the action of the scenario under test

S Helper method encapsulating the setup phase

Op Helper method encapsulating a complex operation as part of the Arrange or Act phase

C Helper method performing both an assertion and a complex operation or a setup

M Helper method encapsulating the creation of a mock or stub object

1 "One-size-fits-all" helper method: performs a complete test, with different scenarios depending on the arguments passed to the method

Assertions: How are the assertions performed? Select only the most complex (farthest in the list).

N No assertion used in the test

1 Single assertion

S Multiple assertions in a sequence

L Assertions in a control structure such as a loop

M Assertions in a mock or stub object

La Assertions in a lambda expression

Setup: Are assertions used in the setup phase of the test?

Y Yes

N No

P Yes, but only immediately before and after an assertion

Resources: Are constrained resources explicitly manipulated for the test? Select all that apply.

N None

So Connection to or manipulation of a socket

Se Connection to or manipulation of a server

F Manipulation of local files

D Manipulation of a database

P Injection into, or use of a specific platform, framework, or environment not managed by the unit under test (e.g., the operating system)

T Manipulation of threads

The following shows the codes for all methods in our sample. The left column contains the qualified names of the tests. Packages are indicated in italics above the unit tests they contain. The eight columns at the right show the codes attributed to the unit test, using the abbreviations described above.

Freemind	Unit	Purpose	Name	Variables	Helpers	Assertions	Setup	Resources
tests.freemind								
Base64Tests.testDifferentBase64ers	C	N	H	I/O/E/B	N	S	N	N
CalendarMarkingTests.testCalendarMarkingEmpty	C	C	S	R	N	1	N	N
CalendarMarkingTests.testCalendarMarkingSingle	C	N	S	R	N	S	N	N
CalendarMarkingTests.testCalendarMarkingDouble	C	N	S	R	N	S	N	N
CalendarMarkingTests.testCalendarMarkingRepeatWeekly	C	N	S	R/~I	N	S	N	N
CalendarMarkingTests.testCalendarMarkingRepeatBeWeekly	C	N	S	R	N	S	N	N
CalendarMarkingTests.testCalendarMarkingRepeatDaily	C	N	S	R	N	S	N	N
CalendarMarkingTests.testCalendarMarkingRepeatMonthly	C	N	S	R	N	S	N	N
CalendarMarkingTests.testCalendarMarkingRepeatYearly	C	N	S	R	N	S	N	N
CalendarMarkingTests.testCalendarMarkingRepeatYearlyEveryNthDay	C	N	S	R	N	S	N	N
CalendarMarkingTests.testCalendarMarkingRepeatYearlyEveryNthDayStartAfterFirstOccurrence	C	N	S	R	N	S	N	N
CalendarMarkingTests.testCalendarMarkingRepeatWeeklyEveryNthDay	C	N	S	R	N	S	N	N
CalendarMarkingTests.testCalendarMarkingRepeatYearlyEveryNthWeek	C	N	S	R/~I	N	S	N	N
CalendarMarkingTests.testCalendarMarkingRepeatYearlyEveryNthWeekStrangeDates	C	N	S	R/~I	N	S	N	N
CollaborationTests.testNormalStartup	Mo	I	H		As	S	N	So/F
CollaborationTests.testNormalStartupWithTwoClients	Mo	I	H		As	S	N	So/F
CollaborationTests.testPublishExistingMapStartup	Mo	I	H		As/M	S	Y	So/F
CollaborationTests.testPublishExistingMapWrongName	Mo	I	H		As/M	S	Y	So/F
DontShowAgainDialogTests.testDialog	C	N	H		N	N	N	N
ExportTests.testExportPng	Me	N	H		N	N	N	F
FreeMindTaskTests.testTestTask	C	N	F		N	1	N	T
HtmlConversionTests.testSetHtml	C	C	H		N	S	N	N
HtmlConversionTests.testEndContentMatcher	A	N	H		As	S	N	N
HtmlConversionTests.testNanoXmlContent	Me	N	H		N	1	N	N
HtmlConversionTests.testXHtmlToHtmlConversion	Me	N	H		N	S	N	N
HtmlConversionTests.testWellFormedXml	Me	N	A		N	S	N	N
HtmlConversionTests.testBr	P	E	H	I/R	N	1	N	N
HtmlConversionTests.testSpaceHandling	P	E	H	I	S	1	N	N
HtmlConversionTests.testSpaceHandlingInShtmlIdempotency	P	E	H	I	S	1	N	N
HtmlConversionTests.testSpaceRemovalInShtml	P	E	H	I	S	1	N	N
HtmlConversionTests.testUnicodeHandling	C	N	H	I/B	N	1	N	N
HtmlConversionTests.testHtmlBodyExtraction	C	N	H	I/~E	S	S	N	N
HtmlConversionTests.testIllegalXmlChars	Me	N	A		N	1	N	N
HtmlConversionTests.testSpaceReplacements	Me	N	H		N	S	N	N
HtmlConversionTests.testListDetection	C	N	H	In	N	S	N	N
HtmlConversionTests.testDetermineNodeAmount	Me	N	A		N	S	N	N
LastOpenedTests.testStrangeCharsInList	C	E	H		N	1	N	N
LastStorageManagementTests.testGetXml	Me	C	F		N	1	N	N

LastStorageManagementTests.testChangeOrAdd	Me	C	F		Op	L	N	N
LastStorageManagementTests.testGetList	Me	C	A		Op	S	Y	N
LastStorageManagementTests.testChangeOrAdd2	Me	C	F		Op	L	N	N
LayoutTests.testYShift	Me	N	A		Op	S	N	N
LayoutTests.testYShiftNegative	Me	N	A		Op	S	N	N
LayoutTests.testYShiftNegativeWith3Childs	Me	N	A		Op	S	N	N
LayoutTests.testScrollMap	Me	N	A		Op	S	N	N
LayoutTests.testYShiftNegativeWith3ChildsWithRootMovement	Me	N	A		Op	S	N	N
LayoutTests.testYShiftNegativeWith3ChildsYCalcToToor	Me	N	A		Op	S	N	N
MarshallerTests.testNewLines	C	E	H		N	S	Y	N
MarshallerTests.testStringEncoder	C	N	F	I/O	N	S	N	N
MarshallerTests.testOsmNominatimConversion	Me	C	H	~R	N	S	N	N
MarshallerTests.testNominatimReverse	Me	C	H	R	N	S	N	N
ScriptEditorPanelTest.testErrorLineNumbers	Me	N	H		N	1	N	N
SignedScriptTests.testSignedInitialization	Me	C	H		N	S	N	N
SignedScriptTests.testScriptContents	C	N	F		N	S	N	N
SplashTests.testLightBuldSplash	C	N	H		N	N	N	P
StandaloneMapTests.testStandalineCreation	C	I	H	E	Op	1	N	N
StandaloneMapTests.testXmlChangeWithoutModeController	C	I	H		Op	S	N	N
ToolsTests.testArgsToUrlConversion	C	N	H		N	1	N	N
ToolsTests.testRichContentConversion	C	N	H	I/R	N	1	N	N
ToolsTests.testUrlConversion	C	N	H	I/R	N	1	N	N
ToolsTests.testRelativeUrlsWindows	C	C	H	I/E	C	1	N	F
ToolsTests.testGetPrefix	Me	N	F		N	1	N	N
ToolsTests.testRelativeUrls	C	N	H	I/E	C	1	N	F
ToolsTests.testRelativeUrls2	C	N	H	I/E	C	1	N	F
ToolsTests.testRelativeUrls3	C	N	H	I/E	C	1	N	F
ToolsTests.testRelativeUrls4	C	N	H	I/E	C	1	N	F
ToolsTests.testRelativeUrlsSpaces	C	E	H	I/E	C	1	N	F
ToolsTests.testOccurrences	Me	N	A		N	S	N	N
ToolsTests.testUpdate	Me	N	H		C	1	N	N
ToolsTests.testUpdateWithSecurityManager	Me	E	H		C	1	N	P
ToolsTests.testPageFormatStorage	C	N	H		N	S	N	N
ToolsTests.testKeyDocumentationPathConversion	C	N	H		N	N	N	N
ToolsTests.testChangedProperties	Me	N	A		N	1	N	N
ToolsTests.testNumberRegexp	A	N	H		N	S	N	N
ToolsTests.testMakeFileHidden	Me	N	F		N	N	N	N
ToolsTests.testHiddenNonExistingFile	Me	E	H		N	N	N	N
TransformTest.testExportHtml	Me	N	H		C	1	N	F
TransformTest.testExportHtmlWithImage	Me	E	H		C	1	N	F
TransformTest.testExportHtmlApplet	Me	E	H		C	1	N	F
TransformTest.testExportHtmlFlash	Me	E	H		C	1	N	F
TransformTest.testExportOoo	Me	E	H		C	1	N	F
tests.freemind.findreplace								
FindTextTests.testTagRemoval	Me	N	H	I	N	1	N	N
FindTextTests.testTagRemovalWithNewlines	Me	E	H	I	N	1	N	N
FindTextTests.testTagRemovalOnlyForHtmlText	Me	E	H	I	N	1	N	N
FindTextTests.testFlatNodeTableFilter	C	N	A		N	S	N	N
FindTextTests.testPositions	C	N	A		N	S	N	N
FindTextTests.testDirectReplace	Me	N	H		N	S	N	N
FindTextTests.testGetPureRegularExpression	Me	N	F		1	S	N	N
FindTextTests.testReplaceNodeText	Me	N	H		N	M	N	N

Eclipse	Unit	Purpose	Name	Variables	Helpers	Assertions	Setup	Resources
org.eclipse.core.tests.databinding								
UpdateValueStrategyTest.testDefaultValidatorForStringToInteger	P	C	H		1	1	N	N
UpdateValueStrategyTest.testDefaultValidatorForStringToIntegerPrimitive	P	C	H		1	1	N	N
UpdateValueStrategyTest.testDefaultValidatorForStringToLong	P	C	H		1	1	N	N
UpdateValueStrategyTest.testDefaultValidatorForStringToLongPrimitive	P	C	H		1	1	N	N
UpdateValueStrategyTest.testDefaultValidatorForStringToFloat	P	C	H		1	1	N	N
UpdateValueStrategyTest.testDefaultValidatorForStringToFloatPrimitive	P	C	H		1	1	N	N
UpdateValueStrategyTest.testDefaultValidatorForStringToDouble	P	C	H		1	1	N	N
UpdateValueStrategyTest.testDefaultValidatorForStringToDoublePrimitive	P	C	H		1	1	N	N
UpdateValueStrategyTest.testDefaultValidatorForStringToByte	P	C	H		1	1	N	N
UpdateValueStrategyTest.testDefaultValidatorForStringToBytePrimitive	P	C	H		1	1	N	N
UpdateValueStrategyTest.testDefaultValidatorForStringToShort	P	C	H		1	1	N	N
UpdateValueStrategyTest.testDefaultValidatorForStringToShortPrimitive	P	C	H		1	1	N	N
UpdateValueStrategyTest.testDefaultValidatorForStringToDate	P	C	H		1	1	N	N
UpdateValueStrategyTest.testDefaultValidatorForNumberToType	P	C	H		1	1	N	N
UpdateValueStrategyTest.testDefaultValidatorForNumberToShort	P	C	H		1	1	N	N
UpdateValueStrategyTest.testDefaultValidatorForNumberToInteger	P	C	H		1	1	N	N
UpdateValueStrategyTest.testDefaultValidatorForNumberToLong	P	C	H		1	1	N	N
UpdateValueStrategyTest.testDefaultValidatorForNumberToFloat	P	C	H		1	1	N	N
UpdateValueStrategyTest.testDefaultValidatorForNumberToDouble	P	C	H		1	1	N	N
UpdateValueStrategyTest.testDefaultValidatorForNumberToBigInteger	P	C	H		1	1	N	N
UpdateValueStrategyTest.testDefaultValidatorForNumberToBigDecimal	P	C	H		1	1	N	N
UpdateValueStrategyTest.testCachesDefaultedValidators	C	C	H		N	1	Y	N
UpdateValueStrategyTest. testFillDefaults_AssertSourceTypeExtendsConverterFromType	P	C	S		N	1	N	N
UpdateValueStrategyTest. testFillDefaults_AssertConverterToTypeExtendsDestinationType	P	C	S		N	1	N	N
ListBindingTest.testUpdateModelFromTarget	Me	N	A		N	S	P	N
ListBindingTest.testUpdateTargetFromModel	Me	N	A		N	S	P	N
ListBindingTest.testGetTarget	Me	N	F		N	1	N	N
ListBindingTest.testGetModel	Me	N	F		N	1	N	N
ListBindingTest.testStatusIsInstanceOfBindingStatus	Me	N	H		N	1	N	N
ListBindingTest.testAddValidationStatusContainsMultipleStatuses	Me	E	A		N	S	N	N
ListBindingTest.testRemoveValidationStatusContainsMultipleStatuses	Me	E	A		N	S	N	N
ListBindingTest.testAddOkValidationStatus	Me	E	A		N	S	N	N
ListBindingTest.testRemoveOkValidationStatus	Me	E	A		N	S	N	N
ListBindingTest.testErrorDuringConversionIsLogged	Me	E	H		N	1	N	N
ListBindingTest.testErrorDuringRemoveIsLogged	Me	E	H		N	1	N	N
ListBindingTest.testErrorDuringMoveIsLogged	Me	E	H		N	1	N	N
ListBindingTest.testErrorDuringReplaceIsLogged	Me	E	H		N	1	N	N
ListBindingTest.testAddListenerAndInitialSyncAreUninterruptable	C	B	H		N	N	N	N
ListBindingTest.testTargetValueIsSyncedToModelIfModelWasNotSyncedToTarget	Me	B	H		N	1	N	N
ListBindingTest.testConversion	C	B	H		N	S	N	N
AggregateValidationStatusTest.testAggregateValidationStatusValueType	Me	N	A		N	1	N	N
AggregateValidationStatusTest.testConstructor_DefaultRealm	Me	N	S		N	1	N	N
org.eclipse.core.tests.databinding.observable.map								
CompositeMapTest.testAddToFirstMap	C	N	H		N	S	Y	N
CompositeMapTest.testAddSharedToFirstMap	C	N	H		N	S	Y	N
CompositeMapTest.testRemoveFromFirstMap	C	N	H		N	S	N	N

CompositeMapTest.testRemoveSharedFromFirstMap	C	N	H	N	S	N	N
CompositeMapTest.testChangeInFirstMap	C	N	H	N	S	N	N
CompositeMapTest.testChangeInFirstMapToShared	C	N	H	N	S	N	N
CompositeMapTest.testChangedInFirstMapFromShared	C	N	H	N	S	N	N
CompositeMapTest.testChangeInSecondMap	C	N	H	N	S	N	N
CompositeMapTest.testDispose	Me	N	F	Op	1	P	N
ComputedObservableMapTest.testGet_ElementNotInKeySet	P	C	S	N	1	N	N
ComputedObservableMapTest.testGet_ElementInKeySet	P	C	S	N	1	N	N
ComputedObservableMapTest.testPut_ElementNotInKeySet	P	C	S	N	S	N	N
ComputedObservableMapTest.testPut_ElementInKeySet	P	C	S	N	S	N	N
ComputedObservableMapTest. testAddToKeySet_BeforeFirstListenerAdded_DoesNotAddListenerToKey	P	C	S	N	1	P	N
ComputedObservableMapTest. testAddToKeySet_AfterFirstListenerAdded_AddsListenerToKey	P	C	S	N	1	P	N
ComputedObservableMapTest.testRemoveFromKeySet_RemovesListenersFromKey	P	C	S	N	1	P	N
ComputedObservableMapTest.testRemoveLastListener_DoNotDiscardKeySet	P	C	S	N	1	N	N
ComputedObservableMapTest.testDispose_RemoveListenersFromKeySetElements	P	C	S	N	1	P	N
ComputedObservableMapTest.testDisposeKeySet_DisposesMap	P	C	S	N	1	P	N
ComputedObservableMapTest. testDisposeKeySet_RemoveListenersFromKeySetElements	P	C	S	N	1	P	N
BidiObservableMapTest.testConstructor_NullArgument	P	C	S	N	1	N	N
BidiObservableMapTest.testGetKeys_Empty	P	C	S	1	La	N	N
BidiObservableMapTest.testGetKeys_NullValue	P	C	S	1	La	N	N
BidiObservableMapTest.testGetKeys_SinglePut	P	C	S	1	La	N	N
BidiObservableMapTest.testGenKeys_ReplaceValue	P	C	S	1	La	N	N
BidiObservableMapTest.testGetKeys_MultipleKeysWithSameValue	P	C	S	1	La	N	N
BidiObservableMapTest.testContainsValue_PutAndRemove	P	C	S	1	La	N	N
org.eclipse.ui.tests.intro							
IntroPartTest.testOpenAndClose	C	I	H	Ac	S	N	P
IntroPartTest.testImage	C	I	H	Ac/S	S	N	P
NoIntroPartTest.testOpenAndClose	C	I	H	Ac	1	N	P
IntroTest2.testPerspectiveChangeWith33StickyBehavior	C	I	H	N	S	Y	P
org.eclipse.ui.tests.operations							
WorkbenchOperationStressTests.test115761	Mo	S	H	S	N	N	P
WorkspaceOperationsTests.testCreateSingleMarkerUndoRedo	C	I	H	Ac/As	S	N	P
WorkspaceOperationsTests.testCreateMultipleMarkersSingleTypeUndoRedo	C	I	H	Ac/As	L	N	P
WorkspaceOperationsTests.testCreateMultipleMarkerTypesUndoRedo	C	I	H	Ac/As	L	N	P
WorkspaceOperationsTests.testUpdateSingleMarkerUndoRedo	C	I	H	Ac/As	S	N	P
WorkspaceOperationsTests.testUpdateMultipleMarkerUndoRedo	C	I	H	Ac/As	S	N	P
WorkspaceOperationsTests.testUpdateAndMergeSingleMarkerUndoRedo	C	I	H	Ac/As	S	N	P
WorkspaceOperationsTests.testUpdateAndMergeMultipleMarkerUndoRedo	C	I	H	Ac/As	S	N	P
WorkspaceOperationsTests.testDeleteMarkersUndoRedo	C	I	H	Ac/As	L	N	P
WorkspaceOperationsTests.testCreateMarkerUndoInvalid	C	I	H	Ac	1	N	P
WorkspaceOperationsTests.testCreateMarkerUndoInvalid2	C	I	H	Ac	1	N	P
WorkspaceOperationsTests.testUpdateMarkersInvalid	C	I	H	Ac	1	N	P
WorkspaceOperationsTests.testUpdateMarkersInvalid2	C	I	H	Ac	1	N	P
WorkspaceOperationsTests.testProjectCreateUndoRedo	C	I	H	Ac	S	N	P/F
WorkspaceOperationsTests.testProjectMoveUndoRedo	C	I	H	Ac	S	N	P/F
WorkspaceOperationsTests.testProjectMoveInvalidLocationUndoRedo	C	I	H	C	1	N	P/F
WorkspaceOperationsTests.testProjectCopyUndoRedo	C	I	H	Ac	S	N	P/F
WorkspaceOperationsTests.testProjectClosedCopyUndoRedo	C	I	H	C	1	N	P/F
WorkspaceOperationsTests.testProjectCopyAndChangeLocationUndoRedo	C	I	H	Ac	S	N	P/F
WorkspaceOperationsTests.testProjectClosedCopyAndChangeLocationUndoRedo	C	I	H	C	1	N	P/F

WorkspaceOperationsTests.testProjectCopyAndChangeToInvalidLocationUndoRedo	C	I	H	C	1	N	P/F
WorkspaceOperationsTests.testProjectRenameUndoRedo	C	I	H	Ac	S	N	P/F
WorkspaceOperationsTests.testProjectDeleteUndoRedo	C	I	H	Ac	S	N	P/F
WorkspaceOperationsTests.test223956	C	B	H	Ac	S	N	P/F
WorkspaceOperationsTests.test201441	C	B	H	Ac	S	N	P/F
WorkspaceOperationsTests.testProjectClosedDeleteUndoRedo	C	I	H	C	1	N	P/F
WorkspaceOperationsTests.testProjectDeleteWithContentUndoRedo	C	I	H	Ac	S	N	P/F
WorkspaceOperationsTests.testProjectClosedDeleteWithContentUndoRedo	C	I	H	C	1	N	P/F
WorkspaceOperationsTests.testFolderCreateLeafUndoRedo	C	I	H	Ac	S	N	P/F
WorkspaceOperationsTests.testFolderCreateNestedInProjectUndoRedo	C	I	H	Ac	S	N	P/F
WorkspaceOperationsTests.testFolderCreateNestedInFolderUndoRedo	C	I	H	Ac	S	N	P/F
WorkspaceOperationsTests.testDeleteNestedResourcesUndoRedo	C	I	H	Ac	S	N	P/F
WorkspaceOperationsTests.testFolderCreateLinkedUndoRedo	C	I	H	Ac	S	N	P/F
WorkspaceOperationsTests.testFolderCreateLinkedNestedUndoRedo	C	I	H	Ac	S	N	P/F
WorkspaceOperationsTests.testFolderMoveUndoRedo	C	I	H	Ac	S	N	P/F
WorkspaceOperationsTests.testRedundantSubFolderMoveUndoRedo	C	I	H	Ac	S	N	P/F
WorkspaceOperationsTests.testRedundantFolderFileMoveUndoRedo	C	I	H	Ac	S	N	P/F
WorkspaceOperationsTests.testFolderCopyUndoRedo	C	I	H	Ac	S	N	P/F
WorkspaceOperationsTests.testFolderCopyLinkUndoRedo	C	I	H	Ac	S	N	P/F
WorkspaceOperationsTests.testFolderCopyRenameUndoRedo	C	I	H	Ac	S	N	P/F
WorkspaceOperationsTests.testFolderRenameUndoRedo	C	I	H	Ac	S	N	P/F
WorkspaceOperationsTests.testFolderDeleteUndoRedo	C	I	H	Ac	S	N	P/F
WorkspaceOperationsTests.testNestedRedundantFolderDeleteUndoRedo	C	I	H	Ac	S	N	P/F
WorkspaceOperationsTests.testNestedRedundantFileDeleteUndoRedo	C	I	H	Ac	S	N	P/F
WorkspaceOperationsTests.testFolderDeleteLinkedUndoRedo	C	I	H	Ac	S	N	P/F
WorkspaceOperationsTests.testFileCreateLeafUndoRedo	C	I	H	Ac	S	N	P/F
WorkspaceOperationsTests.testFileCreateNestedInProjectUndoRedo	C	I	H	Ac	S	N	P/F
WorkspaceOperationsTests.testFileCreateNestedInFolderUndoRedo	C	I	H	Ac	S	N	P/F
WorkspaceOperationsTests.testFileCreateLinkedUndoRedo	C	I	H	Ac	S	N	P/F
WorkspaceOperationsTests.testFileCreateLinkedNestedUndoRedo	C	I	H	Ac	S	N	P/F
WorkspaceOperationsTests.testFileMoveUndoRedo	C	I	H	Ac	S	N	P/F
WorkspaceOperationsTests.testFileMoveAndOverwriteUndoRedo	C	I	H	Ac	S	N	P/F
WorkspaceOperationsTests.testFileCopyUndoRedo	C	I	H	Ac	S	N	P/F
WorkspaceOperationsTests.testFileCopyLinkUndoRedo	C	I	H	Ac	S	N	P/F
WorkspaceOperationsTests.testFileCopyRenameUndoRedo	C	I	H	Ac	S	N	P/F
WorkspaceOperationsTests.testFileCopyAndOverwriteUndoRedo	C	I	H	Ac	S	N	P/F
WorkspaceOperationsTests.testFileRenameUndoRedo	C	I	H	Ac	S	N	P/F
WorkspaceOperationsTests.testFileDeleteUndoRedo	C	I	H	Ac	S	N	P/F
WorkspaceOperationsTests.testFileLinkedDeleteUndoRedo	C	I	H	Ac	S	N	P/F
WorkspaceOperationsTests.testFileAndFolderMoveSameDests	C	I	H	Ac	S	N	P/F
WorkspaceOperationsTests.testFileAndFolderCopyDifferentDests	C	I	H	Ac	S	N	P/F
WorkspaceOperationsTests.testFileAndFolderCopyDifferentNames	C	I	H	Ac	S	N	P/F
WorkspaceOperationsTests.testRedundantFileAndFolderCopy	C	I	H	Ac	S	N	P/F
WorkspaceOperationsTests.testFileAndFolderCopySameDests	C	I	H	Ac	S	N	P/F
WorkspaceOperationsTests.testWorkspaceUndoMonitor	C	I	H	Ac	S	N	P/F
WorkspaceOperationsTests.testProjectCopyUndoInvalid	C	I	H	C	S	N	P/F
WorkspaceOperationsTests.test162655	C	B	H	C	S	N	P/F
WorkspaceOperationsTests.test250125	C	B	H	Ac	S	N	P/F
OperationsAPITest.testContextDispose	Me	I	A	N	S	P	P
OperationsAPITest.testContextHistories	C	I	H	N	S	P	P
OperationsAPITest.testHistoryLimit	C	I	H	N	S	P	P
OperationsAPITest.testLocalHistoryLimit	Me	I	H	N	S	P	P
OperationsAPITest.testOpenOperation	Me	I	F	N	S	N	P

OperationsAPITest.testExceptionDuringOpenOperation	Mo	I	H		N	1	N	P
OperationsAPITest.test94459	Mo	B	H		N	S	N	P
OperationsAPITest.test944959	Mo	B	H		N	1	N	P
OperationsAPITest.test94400	Mo	B	H		N	S	N	P
OperationsAPITest.test123316	Mo	B	H		N	S	N	P
OperationsAPITest.testUnsuccessfulOpenOperation	Mo	I	H		N	S	N	P
OperationsAPITest.testNotAddedOpenOperation	Mo	I	H		N	S	N	P
OperationsAPITest.testMultipleOpenOperation	Mo	I	H		N	S	N	P
OperationsAPITest.testAbortedOpenOperation	Mo	I	H		N	1	N	P
OperationsAPITest.testOperationApproval	Mo	I	H		N	S	P	P
OperationsAPITest.testOperationFailure	Mo	I	H		N	S	N	P
OperationsAPITest.testOperationRedo	Mo	I	H		N	S	N	P
OperationsAPITest.testOperationUndo	Mo	I	H		N	S	N	P
OperationsAPITest.testHistoryFactory	Me	N	H		N	1	N	N
OperationsAPITest.testOperationChanged	Mo	I	H		N	1	N	P
OperationsAPITest.test87675_split	Mo	B	H		S	S	Y	P
OperationsAPITest.test87675_undoredo	Mo	B	H		S	S	Y	P
OperationsAPITest.testOperationApprover2	Mo	I	H		N	S	N	P
OperationsAPITest.testReplaceContext	Mo	I	H		N	S	N	P
OperationsAPITest.test128117simple	Mo	B	H		N	S	N	P
OperationsAPITest.test128117complex	Mo	B	H		N	S	N	P
OperationsAPITest.testStressTestAPI	Mo	S	H		N	N	N	P
OperationsAPITest.test159305	Mo	B	H		N	S	N	P
org.eclipse.jface.tests.databinding.viewers								
ObservableSetTreeContentProviderTest. testConstructor_NullArgumentThrowsException	P	C	S		Ac	1	N	N
ObservableSetTreeContentProviderTest. testGetElements_ChangesFollowObservedList	Me	N	S		S	S	N	N
ObservableSetTreeContentProviderTest. testViewerUpdate_RemoveElementAfterMutation	C	N	S		S	S	N	N
ObservableSetTreeContentProviderTest.testInputChanged_ClearsKnownElements	C	N	S		S	S	P	N
ObservableSetTreeContentProviderTest.testInputChanged_ClearsRealizedElements	C	N	S		S	S	P	N
ViewerSupportTest.testBindList_Twice	Me	C	S	~I	S	N	N	N
ViewerSupportTest.testBindSet_Twice	Me	C	S	~I	S	N	N	N
ViewerSupportTest.testBindListTree_Twice	Me	C	S	~I	S	N	N	N
ViewerSupportTest.testBindSetTree_Twice	Me	C	S	~I	S	N	N	N
ObservableListContentProviderTest.testKnownElements_Realm	Me	C	A		N	1	N	N
ObservableListContentProviderTest.testRealizedElements_Realm	Me	C	A		N	1	N	N
ObservableListContentProviderTest.testKnownElementsAfterSetInput	Me	N	A	~I	N	1	Y	N
ObservableListContentProviderTest. testViewerUpdate_RemoveElementAfterMutation	C	N	A	I	N	S	Y	N
ObservableListContentProviderTest.testInputChanged_ClearsKnownElements	C	N	A	I	N	1	P	N
ObservableListContentProviderTest.testInputChanged_ClearsRealizedElements	C	N	A	I	N	1	P	N
org.eclipse.ui.tests.api.workbenchpart								
RawIViewPartTest.testDefaults	C	N	H		1	S	N	N
LifecycleViewTest.testLifecycle	C	N	F		Ac	S	Y	P
DependencyInjectoinViewTest.testDependencyInjectionLifecycle	C	N	H		Ac	S	Y	P
org.eclipse.ui.tests.api								
IWorkbenchTest.testGetEditorRegistry	Me	C	F		N	1	N	P
IWorkbenchTest.testGetPerspectiveRegistry	Me	C	F		N	1	N	P
IWorkbenchTest.testGetPreferenceManager	Me	C	F		N	1	N	P
IWorkbenchTest.testGetSharedImages	Me	C	F		N	1	N	P
IWorkbenchTest.testGetWorkingSetManager	Me	N	F		N	S	N	P

IWorkbenchTest.testGetWorkbenchWindows	Me	N	F		Op	S	N	P
IWorkbenchWindowActionDelegateTest.testInit	C	N	H		C	S	N	P
IPageServiceTest.testLocalPageService	C	N	H		N	S	Y	P
org.eclipse.ui.tests.markers								
DeclarativeFilterActivityTest.testActivityEnablement	Mo	N	H		S/As	S	N	P
MarkerSupportRegistryTests.testMarkerCategories	Mo	N	H		S/As	S	N	P
MarkersViewColumnSizeTest.testColumnCreate	Mo	N	H		N	S	N	P
MarkersViewColumnSizeTest.testColumnRestore	Mo	N	H		N	S	N	P
org.eclipse.jface.tests.fieldassist								
FieldAssistAPITests.testSimpleContentProposal	Me	N	H		N	S	N	N
FieldAssistAPITests.testContentProposalWithCursor	Me	N	H		N	S	N	N
FieldAssistAPITests.testContentProposalWithLabel	Me	N	H		N	S	N	N
FieldAssistAPITests.testContentProposalWithDescription	Me	N	H		N	S	N	N
FieldAssistAPITests.testInitializationWithInvalidCursor	Me	N	H		N	S	N	N
FieldAssistTestCase.testAutoactivateNoDelay	Mo	I	H		S/As	1	N	P
FieldAssistTestCase.testAutoactivateWithDelay	Mo	I	H		S/As	1	N	P
FieldAssistTestCase.testExplicitActivate	Mo	I	H		S/As	1	N	P
FieldAssistTestCase.testPopupDeactivates	Mo	I	H		S/As	S	N	P
FieldAssistTestCase.testPropagateKeysOff	Mo	I	H		S/As	S	N	P
FieldAssistTestCase.testPropagateKeysOn	Mo	I	H		S/As	S	N	P
FieldAssistTestCase.testBug262022	Mo	B	H		S/As	1	N	P
FieldAssistTestCase.testBug279953	Mo	B	H		S/As	S	N	P
FieldAssistTestCase.testDecorationIsVisible	Mo	I	H		S/As	S	Y	P
FieldAssistTestCase.testPopupFocus	Mo	I	H		S/As	S	Y	P
FieldAssistTestCase.testPopupIsOpen	Mo	I	H		S/As	S	P	P
FieldAssistTestCase.testBug2566515	Mo	B	H		S/As	S	N	P
FieldAssistTestCase.testDefaultPopupPositioningReplaceMode	Mo	I	H		S/As	S	N	P
ControlDecorationTests.testDecorationIsVisible	Mo	I	H		S/As	S	N	P
ControlDecorationTests.testHoverVisibility	Mo	I	H		S/As	S	N	P
org.eclipse.ui.tests.contexts								
PartContextTest.testBasicContextActivation	Mo	I	H		As	S	Y	P
PartContextTest.testContextActivation	Mo	I	H		As	S	Y	P
PartContextTest.testWindowContextActivation	Mo	I	H		As	S	Y	P
PartContextTest.testPageBookPageContextActivation	Mo	I	H		As	S	Y	P/F
Bug84763Test.testWindowChildWhenDialog	Mo	B	H		N	S	N	P
Bug74990Test.testPartIdSubmission	Mo	B	H		N	S	N	P
org.eclipse.ui.tests.menus								
DynamicMenuTest.testDynamicMenu	Mo	I	H		S/Ac	S	N	P
DynamicMenuTest.testDynamicMenuMultiOpen	Mo	I	H		S/Ac	S	N	P
DynamicToolbarTest.testDynamicMenu	Mo	I	H		N	S	N	P
Bug410426Test.testToolbarContributionFromFactoryVisibility	Mo	B	H		S/As	S	N	P
org.eclipse.ui.tests.progress								
ProgressContantsTest.testCommandProperty	Mo	N	H		S/Ac	S	N	P
ProgressContantsTest.testCommandPropertyEnablement	Mo	N	H		S/Ac	S	N	P
ProgressViewTests.testClearTaskInfo	C	N	H		Ac	S	N	P
ProgressViewTests.testNoUpdatesIfHidden	C	C	H		Ac	S	N	P
AccumulatingProgressMonitorTest.testAccumulatingMonitorInUIThread	Mo	I	H		Ac	S	N	P
AccumulatingProgressMonitorTest.testCollector	Mo	I	H	E	Ac	S	Y	P
org.eclipse.ui.tests.propertyPages								
PropertyPageEnablementTest.testAndPage	Mo	I	H		N	L	Y	P
PropertyPageEnablementTest.testOrPage	Mo	I	H	0	N	L	Y	P
PropertyPageEnablementTest.testInstanceOfPage	Mo	I	H	0	N	L	Y	P
org.eclipse.ui.tests.preferences								

FontPreferenceTestCase.testGoodFontDefinition	P	C	H		N	S	N	P
FontPreferenceTestCase.testBadFirstFontDefinition	P	C	H		N	S	N	P
FontPreferenceTestCase.testNoFontDefinition	P	C	H		N	S	N	P
FontPreferenceTestCase.testNonUIThreadFontAccess	C	C	H		As	M	N	P
DeprecatedFontPreferenceTestCase.testGoodFontDefinition	P	C	H		N	S	N	P
DeprecatedFontPreferenceTestCase.testBadFirstFontDefinition	P	C	H		N	S	N	P
DeprecatedFontPreferenceTestCase.testNoFontDefinition	P	C	H		N	S	N	P
WorkingCopyPreferencesTestCase.testRemoveKey	Me	N	F		N	1	Y	P
WorkingCopyPreferencesTestCase.testRemoveNode	Me	N	F		N	1	Y	P
DialogSettingsCustomizationTest.testDialogSettingsContributedByBundle	P	N	H		As	1	Y	P
DialogSettingsCustomizationTest.testDialogSettingsContributedByFileUrl	P	N	H		As	1	Y	P/F
org.eclipse.core.tests.internal.databinding.property.value								
SetSimpleValueObservableMapTest.testGetKeyValueType	Me	C	A		N	S	N	N
ListSimpleValueObservableListTest.testBug301410	Me	B	H		N	N	N	N
MapSimpleValueObservableMapTest.testGetKeyValueType	Me	C	A		N	S	N	N
MapSimpleValueObservableMapTest.testPut_ReplaceOldValue	P	C	S		N	1	N	N
org.eclipse.ui.tsts.keys								
Bug36537Test.testForRedundantKeySequenceBindings	Mo	B	H		N	L	N	P
Bug43800Test.testTruncatingCast	Me	B	H		N	1	N	N
BindingManagerTest.testConstructor	P	C	F		N	1	N	N
BindingManagerTest.testAddBinding	Me	N	F		N	S	N	P
BindingManagerTest.testGetActiveBindingsDisregardingContext	Me	N	F		N	S	N	N
BindingManagerTest.testGetActiveBindingsDisregardingContextFlat	Me	N	F		N	S	N	N
BindingManagerTest.testGetActiveBindingsFor	Me	N	F		N	S	N	N
BindingManagerTest.testGetActiveScheme	Me	N	F		N	1	N	N
BindingManagerTest.testGetBindings	Me	N	F		N	S	N	N
BindingManagerTest.testGetDefinedSchemeIds	Me	N	A		N	S	N	N
BindingManagerTest.testGetLocale	Me	N	F		N	1	N	N
BindingManagerTest.testGetPartialMatches	Me	N	F		N	S	N	N
BindingManagerTest.testGetPerfectMatch	Me	N	F		N	S	N	N
BindingManagerTest.testGetPlatform	Me	N	F		N	1	N	N
BindingManagerTest.testGetScheme	Me	N	F		N	S	N	N
BindingManagerTest.testIsPartialMatch	Me	N	F		N	S	N	N
BindingManagerTest.testIsPerfectMatch	Me	N	F		N	S	N	N
BindingManagerTest.testRemoveBindings	Me	N	F		N	S	N	N
BindingManagerTest.testSetActiveScheme	Me	N	F		N	S	N	N
BindingManagerTest.testGetCurrentConflicts	Me	N	F		N	S	N	N
BindingManagerTest.testSetBindings	Me	N	F		N	S	N	N
BindingManagerTest.testSetLocale	Me	N	F		N	S	N	N
BindingManagerTest.testSetPlatform	Me	N	F		N	S	N	N
BindingManagerTest.testGetBestActiveBindingFor	Me	N	F		N	S	N	N

Weka	Unit	Purpose	Name	Variables	Helpers	Assertions	Setup	Resources
weka.core.tokenizers								
AbstractTokenizerTest.testSerialVersionUID	A	C	F		As	1	N	N
AbstractTokenizerTest.testBuildInitialization	Me	N	H		N	1	N	N
AbstractTokenizerTest.testRegression	C	R	S		Ac	S	N	F
AbstractTokenizerTest.testListOptions	Me	N	F		1	1	N	N
AbstractTokenizerTest.testSetOptions	Me	N	F		1	1	N	N
AbstractTokenizerTest.testDefaultOptions	Me	N	F		1	1	N	N
AbstractTokenizerTest.testRemainingOptions	Me	N	F		1	1	N	N

AbstractTokenizerTest.testCanonicalUserOptions	Me	N	F	1	1	N	N
AbstractTokenizerTest.testResettingOptions	Me	N	F	1	1	N	N
AbstractTokenizerTest.testGlobalInfo	Me	N	F	1	1	N	N
AbstractTokenizerTest.testToolTips	Me	N	F	1	1	N	N
AlphabeticTokenizerTest.testNumberOfGeneratedTokens	Me	N	H	N	S	N	N
CharacterNGramTokenizerTest.testNumberOfGeneratedTokens	Me	N	H	N	S	N	N
NGramTokenizerTest.testNumberOfGeneratedTokens	Me	N	H	N	S	N	N
WordTokenizerTest.testNumberOfGeneratedTokens	Me	N	H	N	S	N	N
weka.classifiers							
AbstractClassifierTest.testToString	Me	N	F	1	1	N	N
AbstractClassifierTest.testSerialVersionUID	Me	N	F	1	1	N	N
AbstractClassifierTest.testAttributes	C	N	H	1	S	N	N
AbstractClassifierTest.testInstanceWeights	Me	N	F	1	N	N	N
AbstractClassifierTest.testOnlyClass	C	N	H	1	L	N	N
AbstractClassifierTest.testNClasses	C	N	H	1	L	N	N
AbstractClassifierTest.testClassAsNthAttribute	C	N	H	As	N	N	N
AbstractClassifierTest.testZeroTraining	C	N	H	1	L	N	N
AbstractClassifierTest.testMissingPredictors	C	N	H	As	L	N	N
AbstractClassifierTest.testMissingClass	C	N	H	As	L	N	N
AbstractClassifierTest.testBuildInitialization	C	N	H	1	L	N	N
AbstractClassifierTest.testDatasetIntegrity	C	N	H	1	L	N	N
AbstractClassifierTest.testUseOfTestClassValue	C	N	H	1	L	N	N
AbstractClassifierTest.testUpdatingEquality	C	N	H	1	L	N	N
AbstractClassifierTest.testRegression	C	R	S	S/1/Ac	L	N	F
AbstractClassifierTest.testListOptions	Me	N	F	1	1	N	N
AbstractClassifierTest.testSetOptions	Me	N	F	1	1	N	N
AbstractClassifierTest.testDefaultOptions	Me	N	F	1	1	N	N
AbstractClassifierTest.testRemainingOptions	Me	N	F	1	1	N	N
AbstractClassifierTest.testCanonicalUserOptions	Me	N	F	1	1	N	N
AbstractClassifierTest.testResettingOptions	Me	N	F	1	1	N	N
AbstractClassifierTest.testGlobalInfo	Me	N	F	1	1	N	N
AbstractClassifierTest.testToolTips	Me	N	F	1	1	N	N
weka.core.neighboursearch							
AbstractNearestNeighbourSearchTest.testSerialVersionUID	A	N	F	1	1	N	N
AbstractNearestNeighbourSearchTest.testListOptions	Me	N	F	1	1	N	N
AbstractNearestNeighbourSearchTest.testSetOptions	Me	N	F	1	1	N	N
AbstractNearestNeighbourSearchTest.testDefaultOptions	Me	N	F	1	1	N	N
AbstractNearestNeighbourSearchTest.testRemainingOptions	Me	N	F	1	1	N	N
AbstractNearestNeighbourSearchTest.testCanonicalUserOptions	Me	N	F	1	1	N	N
AbstractNearestNeighbourSearchTest.testResettingOptions	Me	N	F	1	1	N	N
AbstractNearestNeighbourSearchTest.testGlobalInfo	Me	N	F	1	1	N	N
AbstractNearestNeighbourSearchTest.testToolTips	Me	N	F	1	1	N	N
AbstractNearestNeighbourSearchTest.testNumberOfNeighbors	C	N	H	N	L	Y	N
AbstractNearestNeighbourSearchTest.testBuildInitialization	C	N	H	N	L	N	N
AbstractNearestNeighbourSearchTest.testRegression	C	R	S	Ac	S	N	F
weka.classifiers							
CostMatrixTest.testIncorrectSize	P	C	H	S	1	N	N
CostMatrixTest.test2ClassCostMatrixNoExpressions	P	N	H	S	S	N	N
CostMatrixTest.test2ClassCostMatrixOneSimpleExpression	P	N	H	S	S	N	N
CostMatrixTest.test2ClassCostMatrixOneExpression	P	N	H	S	S	N	N
CostMatrixTest.test2ClassCostMatrixTwoExpressions	P	N	H	S	S	N	N
weka.filters.unsupervised.attribute							
TimeSeriesDeltaTest.testInverted	C	C	H	S	L	N	N

RemoveByNameTests.testTypical	C	N	H	S/Ac	S	N	N
RemoveByNameTests.testTypicalWithClass	C	N	H	S/Ac	S	N	N
RemoveByNameTests.testTypicalInverted	C	N	H	S/Ac	S	N	N
NumericToNominalTest.testTypical	C	N	H	Ac	S	N	N
MathExpressionTest.testTypical	C	N	H	S/Ac	S	N	N
MathExpressionTest.testStats	C	N	H	1	S	N	N
MathExpressionTest.testEquality	C	N	H	S/Ac	S	N	N
MathExpressionTest.testAbs	C	N	H	S/Ac	S	N	N
MathExpressionTest.testsqrt	C	N	H	S/Ac	S	N	N
MathExpressionTest.testLog	C	N	H	S/Ac	S	N	N
MathExpressionTest.testExp	C	N	H	S/Ac	S	N	N
MathExpressionTest.testSin	C	N	H	S/Ac	S	N	N
MathExpressionTest.testCos	C	N	H	S/Ac	S	N	N
MathExpressionTest.testTan	C	N	H	S/Ac	S	N	N
MathExpressionTest.testRint	C	N	H	S/Ac	S	N	N
MathExpressionTest.testFloor	C	N	H	S/Ac	S	N	N
MathExpressionTest.testPow2	C	N	H	S/Ac	S	N	N
MathExpressionTest.testCeil	C	N	H	S/Ac	S	N	N
DiscretizeTest.testTypical	C	N	H	S/Ac	L	N	N
DiscretizeTest.testTypical2	C	N	H	S/Ac	L	N	N
DiscretizeTest.testInverted	C	N	H	S/Ac	L	N	N
DiscretizeTest.testNonInverted2	C	N	H	S/Ac	L	N	N
DiscretizeTest.testBins	Me	N	H	S/Ac	L	N	N
DiscretizeTest.testFindNumBins	Me	N	H	S/Ac	L	N	N
AddUserFieldsTest.testTypical	C	N	H	C	1	N	N
weka.filters.unsupervised.instance							
RemoveFoldsTest.testAllFolds	C	N	H	Ac	L	N	N
ReservoirSampleTest.testTypical	C	N	H	S/Ac	S	N	N
ReservoirSampleTest.testSubSample	C	N	H	S/Ac	S	N	N
ReservoirSampleTest.testHeaderOnlyInput	C	N	H	S/Ac	1	N	N
RemoveMisclassifiedTest.testNominal	C	N	H	S/Ac	1	N	N
RemoveMisclassifiedTest.testNumeric	C	N	H	S/Ac	1	N	N
RemoveMisclassifiedTest.testInverting	C	N	H	S/Ac	1	N	N
weka.attributeSelection							
AbstractAttributeSelectionTest.testAttributes	C	N	H	1	S	N	N
AbstractAttributeSelectionTest.testSerialVersionUID	A	N	H	1	1	N	N
AbstractAttributeSelectionTest.testInstanceWeights	Me	N	F	1	N	N	N
AbstractAttributeSelectionTest.testNClasses	C	N	F	1	N	N	N
AbstractAttributeSelectionTest.testClassAsNthAttribute	C	N	F	As	N	N	N
AbstractAttributeSelectionTest.testZeroTraining	C	N	F	1	L	N	N
AbstractAttributeSelectionTest.testMissingPredictors	C	N	F	As	L	N	N
AbstractAttributeSelectionTest.testMissingClass	C	N	F	As	L	N	N
AbstractAttributeSelectionTest.testBuildInitialization	C	N	F	1	L	N	N
AbstractAttributeSelectionTest.testDatasetIntegrity	C	N	F	1	L	N	N
AbstractAttributeSelectionTest.testRegression	C	R	S	Ac	S	N	F
AbstractAttributeSelectionTest.testListOptions	Me	N	F	1	1	N	N
AbstractAttributeSelectionTest.testSetOptions	Me	N	F	1	1	N	N
AbstractAttributeSelectionTest.testDefaultOptions	Me	N	F	1	1	N	N
AbstractAttributeSelectionTest.testRemainingOptions	Me	N	F	1	1	N	N
AbstractAttributeSelectionTest.testCanonicalUserOptions	Me	N	F	1	1	N	N
AbstractAttributeSelectionTest.testResettingOptions	Me	N	F	1	1	N	N
AbstractAttributeSelectionTest.testGlobalInfo	Me	N	F	1	1	N	N
AbstractAttributeSelectionTest.testToolTips	Me	N	F	1	1	N	N

weka.clusterers								
AbstractClustererTest.testAttributes	C	N	H		1	S	N	N
AbstractClustererTest.testSerialVersionUID	A	N	H		1	1	N	N
AbstractClustererTest.testInstanceWeights	Me	N	F		1	N	N	N
AbstractClustererTest.testZeroTraining	Me	N	F		1	L	N	N
AbstractClustererTest.testMissingPredictors	Me	N	F		As	L	N	N
AbstractClustererTest.testUpdatingEquality	Me	N	F		1	L	N	N
AbstractClustererTest.testBuildInitialization	C	N	F		1	L	N	N
AbstractClustererTest.testDatasetIntegrity	C	N	F		1	L	N	N
AbstractClustererTest.testRegression	C	R	S		Ac	S	N	F
AbstractClustererTest.testListOptions	Me	N	F		1	1	N	N
AbstractClustererTest.testSetOptions	Me	N	F		1	1	N	N
AbstractClustererTest.testDefaultOptions	Me	N	F		1	1	N	N
AbstractClustererTest.testRemainingOptions	Me	N	F		1	1	N	N
AbstractClustererTest.testCanonicalUserOptions	Me	N	F		1	1	N	N
AbstractClustererTest.testResettingOptions	Me	N	F		1	1	N	N
AbstractClustererTest.testGlobalInfo	Me	N	F		1	1	N	N
AbstractClustererTest.testToolTips	Me	N	F		1	1	N	N

Tomcat	Unit	Purpose	Name	Variables	Helpers	Assertions	Setup	Resources
org.apache.tomcat.jdbc.test								
TestSlowQueryReport.testSlowSql	C	N	H		S	S	N	D
TestSlowQueryReport.testSlowSqlJmx	C	N	H		S	S	N	D
TestSlowQueryReport.testFastSql	C	N	H		S	S	N	D
EqualsHashCodeTest.testEquals	Me	N	F		N	S	N	D
EqualsHashCodeTest.testHashCode	Me	N	F		N	S	N	D
CheckOutThreadTest.testDBCPThreads10Connections10	Mo	N	H		S	N	N	T/D
CheckOutThreadTest.testPoolThreads10Connections10	Mo	N	H		S	N	N	T/D
CheckOutThreadTest.testPoolThreads10Connections10Fair	Mo	N	H		S	N	N	T/D
CheckOutThreadTest.testDBCPThreads10Connections10	Mo	N	H		S	N	N	T/D
CheckOutThreadTest.testPoolThreads20Connections10	Mo	N	H		S	N	N	T/D
CheckOutThreadTest.testPoolThreads20Connections10Fair	Mo	N	H		S	N	N	T/D
CheckOutThreadTest.testDBCPThreads10Connections10Validate	Mo	N	H		S	N	N	T/D
CheckOutThreadTest.testPoolThreads10Connections10Validate	Mo	N	H		S	N	N	T/D
CheckOutThreadTest.testPoolThreads10Connections10ValidateFair	Mo	N	H		S	N	N	T/D
CheckOutThreadTest.testDBCPThreads20Connections10Validate	Mo	N	H		S	N	N	T/D
CheckOutThreadTest.testPoolThreads10COnnections20Validate	Mo	N	H		S	N	N	T/D
CheckOutThreadTest.testPoolThreads10Connections20ValidateFair	Mo	N	H		S	N	N	T/D
org.apache.jasper.compiler								
TestELParser.testText	P	N	H		1	1	N	N
TestELParser.testLiteral	P	N	H		1	1	N	N
TestELParser.testVariable	P	N	H		1	1	N	N
TestELParser.testFunction01	P	N	H		1	1	N	N
TestELParser.testFunction02	P	N	H		1	1	N	N
TestELParser.testFunction03	P	N	H		1	1	N	N
TestELParser.testFunction04	P	N	H		1	1	N	N
TestELParser.testFunction05	P	N	H		1	1	N	N
TestELParser.testCompound01	P	N	H		1	1	N	N
TestELParser.testCompound02	P	N	H		1	1	N	N
TestELParser.testCompound03	P	N	H		1	1	N	N
TestELParser.testTernary01	P	N	H		1	1	N	N

TestELParser.testTernary02	P	N	H	1	1	N	N
TestELParser.testTernary03	P	N	H	1	1	N	N
TestELParser.testTernary04	P	N	H	1	1	N	N
TestELParser.testTernary05	P	N	H	1	1	N	N
TestELParser.testTernary06	P	N	H	1	1	N	N
TestELParser.testTernary07	P	N	H	1	1	N	N
TestELParser.testTernary08	P	N	H	1	1	N	N
TestELParser.testTernary09	P	N	H	1	1	N	N
TestELParser.testTernary10	P	N	H	1	1	N	N
TestELParser.testTernary11	P	N	H	1	1	N	N
TestELParser.testTernary12	P	N	H	1	1	N	N
TestELParser.testTernary13	P	N	H	1	1	N	N
TestELParser.testTernaryBug56031	P	B	H	1	1	N	N
TestELParser.testQuotes01	P	N	H	1	1	N	N
TestELParser.testQuotes02	P	N	H	1	1	N	N
TestELParser.testQuotes03	P	N	H	1	1	N	N
TestELParser.testEscape01	P	N	H	1	1	N	N
TestELParser.testEscape02	P	N	H	1	1	N	N
TestELParser.testEscape03	P	N	H	1	1	N	N
TestELParser.testEscape04	P	N	H	1	1	N	N
TestELParser.testEscape05	P	N	H	1	1	N	N
TestELParser.testEscape07	P	N	H	1	1	N	N
TestELParser.testEscape08	P	N	H	1	1	N	N
TestELParser.testEscape09	P	N	H	1	1	N	N
TestELParser.testEscape10	P	N	H	1	1	N	N
TestELParser.testEscape11	P	N	H	1	1	N	N
TestJspConfig.testServlet22NoEL	Mo	N	H	Op	S	N	Se
TestJspConfig.testServlet23NoEL	Mo	N	H	Op	S	N	Se
TestJspConfig.testServlet24NoEL	Mo	N	H	Op	S	N	Se
TestJspConfig.testServlet25NoEL	Mo	N	H	Op	1	N	Se
TestJspConfig.testServlet30NoEL	Mo	N	H	Op	1	N	Se
TestJspConfig.testServlet31NoEL	Mo	N	H	Op	1	N	Se
TestJspConfig.testServlet40NoEL	Mo	N	H	Op	1	N	Se
TestELInterpreterFactory.testBug54239	Mo	B	H	Op	S	N	Se
org.apache.catalina.authenticator.jaspic							
TestSimpleServerAuthConfig.testConfigOnServerAuthConfig	C	N	A	As	1	N	N
TestSimpleServerAuthConfig.testConfigOnGetAuthContext	C	N	A	As	1	N	N
TestSimpleServerAuthConfig.testConfigNone	C	N	A	N	1	N	N
TestAuthConfigFactoryImpl.testRegistrationNullLayer	C	N	H	1	1	N	N
TestAuthConfigFactoryImpl.testRegistrationNullAppContext	C	N	H	1	1	N	N
TestAuthConfigFactoryImpl.testRegistrationNullLayerAndNullAppContext	C	N	H	1	1	N	N
TestAuthConfigFactoryImpl.testSearchNoMatch01	C	N	H	1	1	N	N
TestAuthConfigFactoryImpl.testSearchNoMatch02	C	N	H	1	1	N	N
TestAuthConfigFactoryImpl.testSearchNoMatch03	C	N	H	1	1	N	N
TestAuthConfigFactoryImpl.testSearchNoMatch04	C	N	H	1	1	N	N
TestAuthConfigFactoryImpl.testSearchOnlyAppContextMatch01	C	N	H	1	1	N	N
TestAuthConfigFactoryImpl.testSearchOnlyAppContextMatch02	C	N	H	1	1	N	N
TestAuthConfigFactoryImpl.testSearchOnlyAppContextMatch03	C	N	H	1	1	N	N
TestAuthConfigFactoryImpl.testSearchOnlyLayerMatch01	C	N	H	1	1	N	N
TestAuthConfigFactoryImpl.testSearchOnlyLayerMatch02	C	N	H	1	1	N	N
TestAuthConfigFactoryImpl.testSearchOnlyLayerMatch03	C	N	H	1	1	N	N
TestAuthConfigFactoryImpl.testSearchBothMatch	C	N	H	1	1	N	N
TestAuthConfigFactoryImpl.testRegistrationInsertExact01	C	N	H	1	1	N	N

TestAuthConfigFactoryImpl.testRegistrationInsertExact02	C	N	H		1	1	N	N
TestAuthConfigFactoryImpl.testRegistrationInsertExact03	C	N	H		1	1	N	N
TestAuthConfigFactoryImpl.testRegistrationInsertAppContext01	C	N	H		1	1	N	N
TestAuthConfigFactoryImpl.testRegistrationInsertAppContext02	C	N	H		1	1	N	N
TestAuthConfigFactoryImpl.testRegistrationInsertLayer01	C	N	H		1	1	N	N
TestAuthConfigFactoryImpl.testDetachListenerNonexistingRegistration	C	N	S		N	1	N	N
TestAuthConfigFactoryImpl.testDetachListener	C	N	F		N	1	N	N
TestAuthConfigFactoryImpl.testRegistrationNullListener	C	N	H		N	1	N	N
TestAuthConfigFactoryImpl.testAllRegistrationIds	C	N	H		N	S	N	N
TestAuthConfigFactoryImpl.testRemovePersistentRegistration	C	N	H		N	L	N	N
TestAuthConfigFactoryImpl.testRegistrationNullClassName	C	N	H		1	1	N	N
TestAuthConfigFactoryImpl.testRegistrationNullClassOverrideExisting	C	N	H		1	1	N	N
TestAuthConfigFactoryImpl.testRegistrationNullClassNullLayerNullAppContext	C	N	H		1	1	N	N
TestPersistentProviderRegistration.testLoadEmpty	Me	C	A		N	1	N	F
TestPersistentProviderRegistration.testLoadSimple	Me	N	A		As	1	N	F
TestPersistentProviderRegistration.testSaveSimple	Me	N	A		As	1	N	F
TestPersistentProviderRegistration.testLoadProviderWithoutLayerAndAC	Me	N	A		As	1	N	F
TestPersistentProviderRegistration.testSaveProviderWithoutLayerAndAC	Me	N	A		As	1	Y	F
TestMapperWebapps.testContextRoot_Bug53339	Mo	B	H		Op	1	N	Se
TestMapperWebapps.testContextReload_Bug56658_Bug56882	Mo	B	H		Op	S	N	Se
TestMapperWebapps.testWelcomeFileNotStrict	Mo	N	H		Op	S	N	Se
TestMapperWebapps.testWelcomeFileStrict	Mo	N	H		Op	S	N	Se
TestMapperWebapps.testRedirect	Mo	N	H		Op	S	N	Se
TestMapperPerformance.testPerformance	Mo	P	S		C	L	N	Se
TestMapper.testAddHost	Me	N	F		Op	S	N	Se
TestMapper.testRemoveHost	Me	N	F		Op	L	N	Se
TestMapper.testMap	Me	N	F		N	S	N	Se
TestMapper.testAddRemoveContextVersion	Me	N	A		Op	S	N	Se
TestMapper.testReloadContextVersion	Me	N	A		N	S	N	Se
TestMapper.testContextListConcurrencyBug56653	C	B	H		Op	L	N	Se
org.apache.catalina.tribes.test.channel								
TestRemoveProcessException.testDataSendSYNCACK	P	N	H		N	L	N	Se
TestDataIntegrity.testDataSendNO_ACK	P	N	H		N	1	N	T/Se
TestDataIntegrity.testDataSendASYNCM	P	N	H		N	1	N	T/Se
TestDataIntegrity.testDataSendASYNC	P	N	H		N	1	N	Se
TestDataIntegrity.testDataSendACK	P	N	H		N	1	N	Se
TestDataIntegrity.testDataSendSYNCACK	P	N	H		N	1	N	Se
TestChannelConfig.testIntInput	P	N	H		N	1	N	N
TestChannelConfig.testStringInputSimple	P	N	H		N	1	N	N
TestChannelConfig.testStringInputCompound	P	N	H		N	1	N	N
TestChannelConfig.testStringRepresentationOfIntValue	P	N	H		N	1	N	N
TestChannelConfig.testStringInputForMapSendOptions	P	N	H		N	1	N	N

Hibernate	Unit	Purpose	Name	Variables	Helpers	Assertions	Setup	Resources
org.hibernate.test.fetchstrategyhelper								
BatchFetchStrategyHelperTest.testManyToOneDefaultFetch	C	C	H		Op	S	N	N
BatchFetchStrategyHelperTest.testManyToOneJoinFetch	C	C	H		Op	S	N	N
BatchFetchStrategyHelperTest.testManyToOneSelectFetch	C	C	H		Op	S	N	N
BatchFetchStrategyHelperTest.testCollectionDefaultFetch	C	C	H		Op	S	N	N
BatchFetchStrategyHelperTest.testCollectionJoinFetch	C	C	H		Op	S	N	N
BatchFetchStrategyHelperTest.testCollectionSelectFetch	C	C	H		Op	S	N	N

BatchFetchStrategyHelperTest.testCollectionSubselectFetch	C	C	H	Op	S	N	N
FetchStrategyHelperTest.testManyToOneDefaultFetch	C	C	H	Op	S	N	N
FetchStrategyHelperTest.testManyToOneJoinFetch	C	C	H	Op	S	N	N
FetchStrategyHelperTest.testManyToOneSelectFetch	C	C	H	Op	S	N	N
FetchStrategyHelperTest.testCollectionDefaultFetch	C	C	H	Op	S	N	N
FetchStrategyHelperTest.testCollectionJoinFetch	C	C	H	Op	S	N	N
FetchStrategyHelperTest.testCollectionSelectFetch	C	C	H	Op	S	N	N
FetchStrategyHelperTest.testCollectionSubselectFetch	C	C	H	Op	S	N	N
NoProxyFetchStrategyHelperTest.testManyToOneDefaultFetch	C	C	H	Op	S	N	N
NoProxyFetchStrategyHelperTest.testManyToOneJoinFetch	C	C	H	Op	S	N	N
NoProxyFetchStrategyHelperTest.testManyToOneSelectFetch	C	C	H	Op	S	N	N
org.hibernate.envers.test.integration.data							
Lobs.testRevisionsCounts	Mo	I	H	S	1	N	D
Lobs.testHistoryOfId1	Mo	I	H	S	S	N	D
Lobs.testRevisionsCountsForAuditedArraysWithNoChanges	Mo	B	H	S	1	N	D
Lobs.testHistoryOfId2	Mo	B	H	S	1	N	D
Enums.testRevisionsCounts	Mo	I	H	S	1	N	D
Enums.testHistoryOfId1	Mo	I	H	S	S	N	D
LobSerializables.testRevisionsCounts	Mo	I	H	S	1	N	D
LobSerializables.testHistoryOfId1	Mo	I	H	S	S	N	D
org.hibernate.test.resource.transaction.jta							
InaccessibleJtaPlatformTests.testInaccessibleTransactionManagerHandling	C	C	H	N	S	N	N
InaccessibleJtaPlatformTests.testInaccessibleUserTransactionHandling	C	C	H	N	S	N	N
AbstractBasicJtaTestScenarios.basicBmtUsageTest	C	N	H	S	S	Y	D
AbstractBasicJtaTestScenarios.rollbackBmtUsageTest	C	N	H	S	S	Y	D
AbstractBasicJtaTestScenarios.basicCmtUsageTest	C	N	H	S	S	Y	D
AbstractBasicJtaTestScenarios.basicCmtUsageWithPulseTest	C	N	H	S	S	Y	D
AbstractBasicJtaTestScenarios.rollbackCmtUsageTest	C	N	H	S	S	Y	D
AbstractBasicJtaTestScenarios.explicitJointingTest	C	N	H	S	S	Y	D
AbstractBasicJtaTestScenarios.jpaExplicitJoiningTest	C	N	H	S	S	Y	D
AbstractBasicJtaTestScenarios.assureMultipleJoinCallsNoOp	C	N	H	S	S	Y	D
AbstractBasicJtaTestScenarios.explicitJoinOutsideTxnTest	C	N	H	S	S	Y	D
AbstractBasicJtaTestScenarios.basicThreadCheckingUsage	C	N	H	N	S	Y	D
AbstractBasicJtaTestScenarios.testMarkRollbackOnly	C	N	H	N	S	Y	D
AbstractBasicJtaTestScenarios.testSynchronizationFailure	C	N	H	N	S	Y	D
org.hibernate.test.annotations.derivedidentities.e4.a							
DerivedIdentitySimpleParentSimpleDepTest.testOneToOneExplicitJoinColumn	Mo	I	H	S	S	Y	D
DerivedIdentitySimpleParentSimpleDepTest.testManyToOneExplicitJoinColumn	Mo	I	H	S	S	Y	D
DerivedIdentitySimpleParentSimpleDepTest.testSimplePkValueLoading	Mo	I	H	S	1	N	D
org.hibernate.test.metamodel							
WildcardTypeAttributeMetaModelTest.testWildcardGenericAttributeCanBeResolved	C	B	H	C	1	N	N
EmbeddableMetaModelTest.testEmbeddableCanBeResolvedWhenUsedAsInterface	C	B	H	C	La	N	N