

Labeling Guide - Testability Refactoring Patterns

This task is an important step of a scientific study on refactoring for testability. One of the researchers has already labelled several hundreds GitHub Pull Requests (PRs) with common refactoring patterns that are listed below checking whether the specific pattern is present in the Pull Request or not. To ensure that the results are reliable and minimise errors in the labelling process, we need a second (independent) opinion.

Refactoring for testability is essentially about changing the structure of production code without changing its behaviour in order to make it possible to write unit-tests for it or to improve or extend existing unit tests Here two examples of refactoring the production code for the purpose of testability:

Example 1: It would make sense to inject an instance of the dependent class from a unit-test instead of creating a new instance of the dependent class in the constructor: you can use a fake/mock implementation of the dependent class from the unit-test and therefore control the dependencies of the class under test.

Example 2: a class uses `System.currentTimeMillis()` to take date/time one hour ago and provides a string formatted in a certain way. You cannot use `assertEquals("03:54", timeCalculator.getTimeOneHourAgo())` in a unit-test because system time changes constantly, but you can inject `java.time.Clock` into the class, use fake implementation of `Clock` and use fixed time in the unit-test.

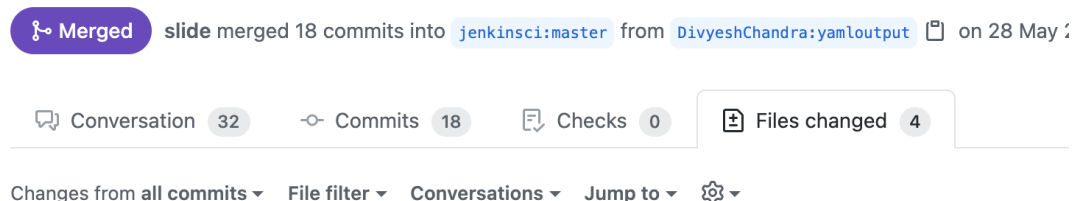
Process

Please carefully read and follow the following process. The validity of the study is dependent on how well you label the data.

1. Take a PR from the spreadsheet provided to you.
2. If you haven't done labelling of PRs with refactoring patterns for testability before, you can label 10 PRs and ask the researchers for feedback.
3. The URL of the PR provided to you in the spreadsheet in the **url** column, will open the **Files** corresponding to the PR: for example

<https://github.com/jenkinsci/email-ext-plugin/pull/207/files?diff=split&w=1>

Add additional parameter to FAILED_TESTS token to



The screenshot shows a GitHub Pull Request interface. At the top, it says "Merged" in a purple button, followed by "slide merged 18 commits into jenkinsci:master from DivyeshChandra:yamloutput" and "on 28 May". Below this, there are tabs for "Conversation" (32), "Commits" (18), "Checks" (0), and "Files changed" (4). At the bottom, there are filters for "Changes from all commits", "File filter", "Conversations", "Jump to", and a settings icon.

4. Please open the Conversation of the the PR in a separate browser tab (with URL <https://github.com/jenkinsci/email-ext-plugin/pull/207>) and navigate between Files/Conversation tabs to better understand the changes/their context.
5. Every PR can contain none, one or eventually more refactoring patterns for testability.
6. It helps sometimes to read the description of the PR (in Conversation tab). But be aware that it will not always provide much information about what exactly has been refactored to improve testability.
7. Every PR needs to be labelled in the **pr_group** column as irrelevant/**incl_ref_for_test/only_ref_for_test**, see **Types of PRs** below. If the PR contains more than one refactoring for testability, all rows in the table should have the same pr_group (or empty for second)
8. PRs that are relevant (**incl_ref_for_test/only_ref_for_test**) should contain refactoring patterns from the list of **Testability refactoring patterns**. If you think the PR includes a refactoring pattern that is not listed below, you can write **others** and mention in the **comment** column why you think it is a pattern.
9. In the **prod_file** column you can mention the filename of the production class (system under test) that has been modified, you can click on the clipboard icon next to the filename.

197 src/main/java/hudson/plugins/emailext/plugins/content/FailedTestsContent.java

@@ -1,5 +1,10 @@

package hudson.plugins.emailext.plugins.content;

10. In the **test_location** column you can mention the location of the test code that uses the refactored prod code: invokes it or overrides/mocks it, etc.. Die URL endet auf R[0-9]+, wie im Beispiel unten R90 fuer die Zeile 90 der Testdatei.

github.com/apache/bookkeeper/pull/2333/files?diff=split&w=1#diff-2824c64a13c57f42fd58dbb382db0ba701e3d56850aaaae1f2e3062c2af0ce35R90

QuorumCoverage should only count unknown nodes #2333

Changes from all commits File filter Conversations Jump to

bookkeeper-server/src/test/java/org/apache/bookkeeper/client/ReadLastConfirmedOpTest.java

```

64 +
65 + /**
66 +  * Test for specific bug that was introduced with dcdle88.
67 +  */
68 + @Test
69 + public void testBookieFailsAfterLedgerMissingOnFirst() throws E
70 +     long ledgerId = 0xf00b;
71 +     List<BookieSocketAddress> ensemble = Lists.newArrayList(boo
72 +     byte[] ledgerKey = new byte[0];
73 +
74 +     MockBookieClient bookieClient = new MockBookieClient(execut
75 +     DistributionSchedule schedule = new RoundRobinDistribution5
76 +     DigestManager digestManager = DigestManager.instantiate(led
77 +
78 +
79 +     true
80 +
81 +     CompletableFuture<Void> blocker = new CompletableFuture<>()
82 +     bookieClient.setPreReadHook((bookie, lId, entryId) -> {
83 +         if (bookie.equals(bookie1)) {
84 +             return CompletableFuture.completedFuture(null);
85 +         } else {
86 +             return blocker;
87 +         }
88 +     });
89 +     CompletableFuture<DigestManager.RecoveryData> promise = new
90 +     ReadLastConfirmedOp op = new ReadLastConfirmedOp(
91 +         bookieClient, schedule

```

11. In the **comment** column you can provide brief details that support your labeling decision: method XX made protected so it is overridden in junit, etc.
12. A PR can contain more than one refactoring pattern, in which case you can insert a new line in the spreadsheet with the same url and details of the second (or third, etc). refactoring: **ref_pattern**, **prod_file**, **location**, **comment**.

13. PRs with only added code **do not** include refactoring.
14. Very few PRs may contain code in scala/Kotlin/clojure since classes written in any JVM-based language can interact with each other. Try to understand the code changes, especially if you think they are relevant to refactoring for testability. It is fine to proceed to the next PR if you don't understand some of the scala/etc. code.
15. Sometimes a PR contains (relevant for us) refactorings for testability without test code, it is however less common than a PR with general refactorings in production code. If you identify refactorings for testability without relevant test code, just write **none/none relevant** in **test_location** column.
16. Try to not spend more than 10min per PR (or 30min for the few larg PRs with more than 10 files). You would usually see patterns sooner than later, if there are any. You can scroll the source code back-and-forward, often it helps to look at the test code first.
17. Feel free to use prefix **borderline** or **unsure** in the comment section if the changes in the PR are not so obvious.
18. Some PRs/projects may have been deleted meanwhile, please write **irrelevant** in pr_group und **deleted** in comment section.
19. Column **java_files_count** tells how many java files have been changed in the PR and gives you an estimate how large the PR is.
20. Go to step 1 if you have time for another PR :)

Types of PRs

- A. **only_ref_for_test (only refactorings for testability)**: such PRs contain *only* refactoring in classes under test aka production classes (under src/main/java and similar) and relevant test classes. Refactoring in such PRs should be relevant to testability: make it possible to unit-test a piece of functionality, improve code coverage, make it simpler to unit-test.
- B. **incl_ref_for_test**: same as above, but may also include additional bugfixes, new features, and other functional changes in production code.
- C. **irrelevant**: PRs that do not contain refactorings for testability. This category also includes bugfixes with relevant unit-tests, for which no production code has been refactored.

Testability refactoring patterns

The following testability refactoring patterns have been identified and can be used for categorisation based on the following rules/examples. A single PR can contain multiple refactoring patterns in the same or different production classes.

1. extract_method_for_override

In this testability refactoring pattern, a section of code is extracted into a method that is defined as protected/package/public and is then overridden in a subclass defined in a unit-test/near unit-test (in src/test/java and similar directories) or using Mockito.when(classUnderTest.getExtractedMethod()).thenReturn(mockedObject).

Examples:

1. <https://github.com/apache/incubator-heron/pull/536/files#diff-a053ee8dc2fc86c17a5f5a900a7b1d0de8763658f4068b603c3779e85c28a83aR73> Here startExecutorProcess method is overridden using Mockito
2. <https://github.com/apache/incubator-heron/pull/691/files#diff-9e584d73f147296cc864a9a993a8bfb8c8527e71c7d9b8e83b741c42a89605af0R117> here callRuntimeManagerRunner has been overridden using Mockito7
3. From the same PR, but another method:
<https://github.com/apache/incubator-heron/pull/691/files#diff-9e584d73f147296cc864a9a993a8bfb8c8527e71c7d9b8e83b741c42a89605af0R109> here getSchedulerClient method is overridden.
4. A more unusual case:
<https://github.com/codice/ddf/pull/433/files#diff-a5b614fe548edd0534f140f085578dda6dd0d3e41382197e89e049f2010efeb1R228> here a section of code has been replaced with FileUtils.forceMkdir and then overridden using PowerMockito to emulate an IOException thrown

2. extract_method_for_invocation

In this pattern a method is extracted in order to test it from a unit-test. It would also include a method that sets an attribute in a class under test to bring it into a state necessary for testing.

Examples:

1. <https://github.com/CIRDLES/Topsoil/pull/162/files#diff-7b9bf92f128762a2fad88b40bf74c65572d54460dd6a7ec7a135fe04c46ab59cR62> Here writeSVGToOutputStream method was extracted from save() method in order to test it separately from a unit-test.
2. <https://github.com/hmcts/idam-web-public/pull/196/files#diff-c155b5d2e77d82406ac20d0180b9f4a68371814c3671cb36d4a471c5692b73faR1499> here makeCookieSecure method is extracted and then the result of invocation validated using assertThat.
3. <https://github.com/kiegroup/kie-wb-common/pull/367/files#diff-723a9b8d886a2a19624adcf31c8857c22d33200a86b7105f6b9a9dae45529565R44> here get() method was extracted and invoked from junit.
4. A trickier, borderline case:
<https://github.com/GoogleCloudPlatform/google-cloud-eclipse/pull/1969/files?diff=split&w=1#diff-eeb575b530450ac30e8b3cc876ee53da8e03b9a77da6c8765e4f885e85f68da8R59> here findLocationField() has been extracted instead of accessing attributes of the class under test.
5. <https://github.com/pentaho/pentaho-platform/pull/2793/files#diff-80ca59ef80434e0f56b69ddd3f60ae731636696ac952675cc909e5d232ed7e13R50> here addHandlerParamJson has been extracted and commented as // visible for testing purposes. Similar comments or @VisibleForTesting annotation are relatively common.

3. widen_access_for_invocation

In this pattern access to an existing method, attribute or class is widened: from private to protected/public/package, from protected to package/public. A method/other member can also be made static in order to avoid instantiation of the class under test:

MyClass.calculateSomething() instead of new MyClass().calculateSomething(). The purpose of such refactoring is to invoke the method from a unit-test.

Examples:

1.

<https://github.com/OryxProject/oryx/pull/164/files#diff-e549a3311dcd3be5ddec5d29777121cc8bf7b9a1b45cb75fdf903fa71d96c7f6R68> here access to calcSilhouetteCoefficient is

widened from private to package.

2.

<https://github.com/pentaho/pentaho-platform/pull/3696/files#diff-aecd0c60cb6460753c921ec554d7998e1814b17af14684aa1c62f9d89a816686R112> here

setEnvironmentVariablesFolder is widened from private to public in order to invoke from junit.

3.

<https://github.com/jenkinsci/gitlab-plugin/pull/335/files#diff-26725b448b0d698fc3c3c3e1be7cc88015138cb25d6bb9bb89eba969364d8296R167> here access to NoOpAction has been

widened from private static to package static in order to access it from instanceof in the junit.

4.

<https://github.com/apache/druid/pull/2102/files?diff=split&w=1#diff-b9e00c64528c1ed053a08b2a9f8eafce558d3ddc75f503bd16ae6054180936dbR73> here access to attribute

CacheMonitor::cache has been widened from private to package in order to read it from a unit-test.

5.

<https://github.com/Alluxio/alluxio/pull/1629/files?diff=split&w=1#diff-05f6d61b3485c9cae64cdeb03bd18f3fadf5edc4c7eeb0a00846545fbc323dea1R89> a pretty unusual case of creating a

package-accessible PrivateClass class that provides access to private members of BlockMaster, such as private final mLostWorkers. Unusual, but it achieves the goal of widen_access_for_invocation.

4. widen_access_for_override

In this pattern, access to an existing method, attribute or class is widened from private to protected/public/package or from protected to package/public. Same as above, but for the purpose of overriding the member from a unit-test.

Examples:

1.

<https://github.com/Netflix/hollow/pull/219/files#diff-bebffa128a4d63e213c3189cfd6488a2035f161c712870be19d80e13448d24a4eR213> here publish() method has been widened to

protected in order to override its behaviour by throwing a RuntimeException using Mockito.doThrow.

2.

<https://github.com/OpenRefine/OpenRefine/pull/2839/files#diff-367c88cfad59847336f818d4276255a5c1c23cf155073ae8a5511d8521c3eeeaR59> here access to findValues method has been widened in order to override it using Mockito.thenReturn.

3.

<https://github.com/getodk/collect/pull/2489/files#diff-23f43ece96723536aea91a0b0073574fb4423e01eed6b2bc15214083e93a4cf8R29> here access to checkForRequiredSensors widened from private to public and overridden in MockBearingWidget subclass.

4.

<https://github.com/grpc/grpc-java/pull/3029/files?diff=split&w=1#diff-daee08ff4a4ba923599b66efc890057364c2e38046e102a5c968ffb72ea772dbR194> a more unusual case: a builder is extracted and builder.executorPool is overridden.

5.

<https://github.com/kielogroup/drools-wb/pull/184/files?diff=split&w=1#diff-90c28778706543a6f5e6929d07852b712a1238915f521944180de88c4dc9620fR102> access to ConstraintValueEditor::wrap method has been widened to package and overridden from an anonymous subclass in a junit.

5. extract_class_for_invocation

In this pattern, a piece of existing functionality is usually extracted into a (smaller) class that can be unit-tested.

Examples:

1. <https://github.com/jmxtrans/jmxtrans/pull/171/files#diff-d97fe6da56ad6b50374f4cbe77acdfb7b6db47e30070cb54936ca8638fcee6d3R191> here JmxResultProcessor class has been extracted and invoked from a unit-test.
2. <https://github.com/jmxtrans/jmxtrans/pull/291/files#diff-55a66325a926dfab96b606daf8b70a2d6f74d11b744519b5c98d116186a39cf1R22> here convertToDouble method has been extracted into a separate class ObjectToDouble and tested separately from ObjectToDoubleTest.
3. <https://github.com/azkaban/azkaban/pull/1765/files?diff=split&w=1#diff-d06231180a58579f8eedec31b2fa3cd65820a1dfb6917ac1be50cd1bb3cc7bf4R68> here code from shutdownAndAwaitTermination extracted into ExecutorServiceUtils and it is mocked (its behaviour overridden) in a unit-test.

6. extract_class_for_override

In this pattern, a class is extracted in order to override it from a unit-test.

Examples:

1.

<https://github.com/dropwizard/metrics/pull/516/files?diff=split&w=1#diff-77404fc98b2af4ff4f2d91472bb0cd19f1b6f5c86be4ce61acd1922638f6d54cR105> here ObjectNameFactory has been extracted in order to supply a pre-created ObjectName from a unit-test, instead of creating an ObjectName from JmxReporter::createName.

2.

<https://github.com/firebase/firebase-android-sdk/pull/217/files?diff=split&w=1#diff-c7d337444>

[4dbf8d5868c2367b77fd52fe7608c08880160b817b272fe396cf9e6R75](https://github.com/kiegroup/kie-wb-common/pull/277/files#diff-195b466598b765a6a5b7d806b12c6a434c4134715caad6a538ce10daee14b015R52) here ConnectivityMonitor interface has been extracted and subclassed as FakeConnectivityMonitor in a unit-test.

7. add_constructor_param

In this pattern an additional constructor is created or an extra parameter is added to an existing constructor to supply a dependency from a unit-test.

Examples:

1.

<https://github.com/kiegroup/kie-wb-common/pull/277/files#diff-195b466598b765a6a5b7d806b12c6a434c4134715caad6a538ce10daee14b015R52> here lockRequired is supplied from a unit-test instead of injecting it via spring or other dependency injection framework.

2.

<https://github.com/openmrs/openmrs-contrib-android-client/pull/349/files#diff-21e6dfb384292a19172eb526ff517140445ff7d3b3172b95eab5808b63db773fR48> a typical example of replacing new XXXDependency() inside a constructor with an object supplied from a unit-test.

8. create_constructor

In this pattern a constructor is created in order to supply dependencies from a unit-test. It is frequently used with @Autowired and in spring context.

Examples:

1.

<https://github.com/dhis2/dhis2-core/pull/2892/files#diff-755ce64247e7039b26560990d290fe0bbd03bcc0ce6ed7cb9f65ba0ae7b73134R118> here SmsMessageSender constructor has been created and used from a unit-test. In production code it would be called by spring/other dependency injection framework.

2.

<https://github.com/apache/incubator-heron/pull/691/files#diff-9e584d73f147296cc864a9a993a8bfbcb8527e71c7d9b8e83b741c42a89605af0R45> Constructor RuntimeManagerMain created to inject config and runtime.

9. override_system_time

It is a rather rare pattern. Here developers usually try to override System.currentTimeMillis() or date returned by new Date by using java.time.Clock or supplier of time.

Examples:

1. <https://github.com/firebase/firebase-android-sdk/pull/1333/files#diff-329ea845fe9ddff6aa413e47e733c53cba6df39b0905223ee9890badf53a4e5fR154> here Clock interface is created with one real and one fake implementation, the latter is used in the unit-test to override system time.
2. <https://github.com/azkaban/azkaban/pull/1975/files#diff-871283916aae52436d516c9da613d7b527846a89210fc110c36da3879a0ec81eR103> here joda DateTimeUtils is

Coding guide ref_for_test

used to set current time and production code uses `DateTime.now()` instead of `System.currentTimeMillis()`.

10. Other

Other testability patterns are possible and acceptable, please mark them as `ref_pattern=other` and write in comment why the changes in a PR are relevant to testability.