Pontifícia Universidade Católica
do Rio de Janeiro

Raphael Cabral

Upgrade Configuration

Programming Conclusion Projec

Project presented to the Programa de Pós–graduação em Informática, do Departamento de Informática da PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Informática.

Advisor: Prof. Marcos Kalinowski

Rio de Janeiro
November 2022

Pontifícia Universidade Católica
do Rio de Janeiro

Raphael Cabral

Upgrade Configuration

Project presented to the Programa de Pós–graduação em Informática da PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Informática. Approved by the Examination Committee:

Prof. Marcos Kalinowski
Advisor
Departamento de Informática – PUC-Rio

Prof. Clarisse Sieckenius
Departamento de Informática – PUC-Rio

Rio de Janeiro, November 9th, 2022

Table of contents

List of figures

List of tables

List of codes

List of Abreviations

JPA – Java Persistence API.

SOLID – is a mnemonic acronym for the principles introduced by Robert C. Martin (also known as Uncle Bob): Single Responsibility; Open Closed; Liskov Substitution; Interface Segregation; and Dependence Inversion.

From a trace, architecture is born. And when it's beautiful and creates surprise, it can, if handled well, reach the top level of a work of art.

Oscar Niemeyer, .

1
Introduction

Normally Scripts executed in the database do not have the same level of organization as versioned code bases. Even if the Script content is versioned, there is no guarantee that its execution will be synchronized with the evolution of the code base. This process is often done manually, which can lead to a risk of human error, or automated by resources external to the application, which can bring difficulties for the developer.

In order to bring the same level of organization already observed in the versioned code bases, Upgrade Configuration is a tool that aims to bring order and organization to the Scripts that are executed in the database, working as a version control of the same. A tool like this allows you to:

- Synchronize the database with the application version;
- Know which Scripts were executed or not;
- Automate the execution of scripts;
- Create a database from scratch;
- Allows to create a rollback of changes to the database.

The remainder of this document is organized as follows. In Section 2 we provide the specification of Upgrade Configuration. In Section 3, we describe the project, the architecture of application. Section 4 will present the source code repository, the result of its static analysis, Software Design Patterns and Good practices adopted. Next, Tests composed of Unit Tests and Integration Tests, are introduced in Section 5. Lastly, an User Manual by Upgrade Configuration is presented on Section 6.

2
Specification

This section presents the specification of the system developed in this work. Thus, the goals, requirements and use cases are provided.

2.1
Goal

The Goal of the application is to create a version control for the database, so that the developer has the power to migrate it easily and with security. Where database evolutions are shipped along with the application and run automatically at startup.

The software facilitates continuous delivery process and favors the expansion of the agile development culture and devops.

2.2
Requirements

This subsection presents the functional and non-functional requirements of the project.

2.2.1
Functional Requirements

[FR1] The software must allow the developer to upgrade its base of data, together with its codebase.

[FR2] The software must allow the developer to perform regression of your database, together with your codebase.

[FR3] The software must allow the developer to create a database from scratch;

[FR4] The software must allow the developer to Automate the execution of scripts;

[FR5] The software must allow the developer to Synchronize the database with the application version;

2.2.2
Non Functional Requirements

[NFR1] The software must support its use in java applications that make use of JPA.

[NFR2] The software must support its use as a Maven plugin for Java applications.

[NFR3] The software must support Upgrade and Downgrade control through the manipulation of an XML file by the Developer.

[NFR4] Users must be able to use the product after reading the user manual.

2.3 Use Cases

This section presents the use case diagram and describes each use case in the program.

2.3.1 Use Case Diagram

The use case diagram is shown in Figure 2.1.

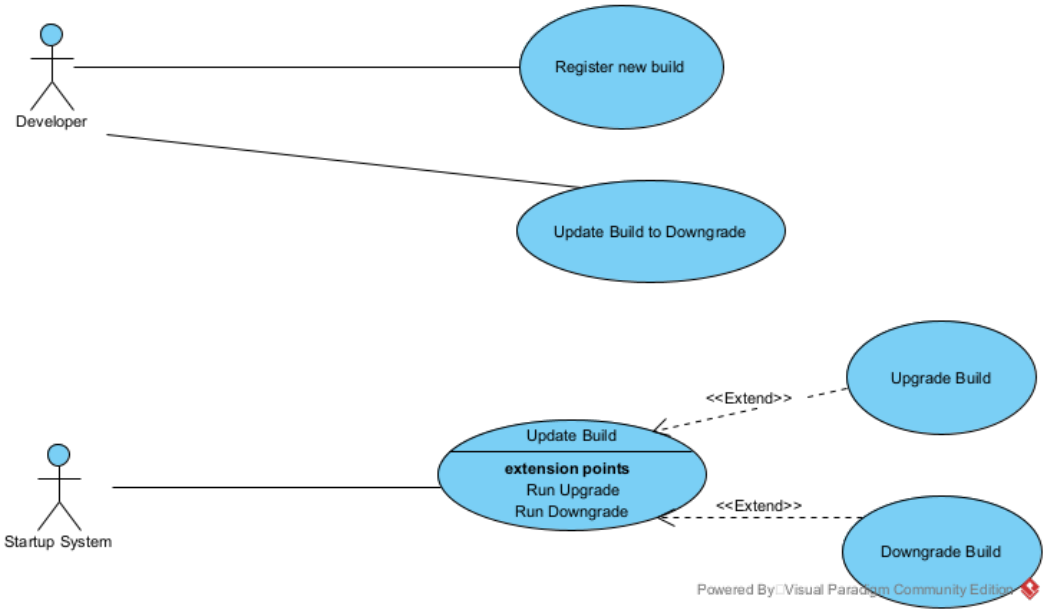


Figure 2.1: Use Case Diagram

2.3.2 Use Cases Description

This subsection further describes each use case found in Figure 2.1.

Table 2.1: Use Case 1: Register new build.

Use Case 1: Register new build	
Main Actor: Developer Goal: Register new build. Preconditions: - Existence of the upgrade-configuration.xml file in the project root. Postconditions: - New build is persisted	
Normal Flow	
Actor	System
1- Actor opens the upgrade-configuration.xml file. 2- Ator adds new build to upgrade-configuration.xml file. 3– Actor saves file.	

Table 2.2: Use Case 2: Update Build to Downgrade.

Use Case 2: Update Build to Downgrade	
Main Actor: Developer Goal: Mark existing build for Downgrade process. Preconditions: - Existence of the upgrade-configuration.xml file in the project root, with at least one build. Postconditions: - Build is marked for Downgrade process.	
Normal Flow	
Actor	System
1- Actor opens the upgrade-configuration.xml file. 2- Actor changes the build to have the downgrade attribute true. 3 – Actor saves file.	

Table 2.3: Use Case 3: Update Build.

Use Case 3: Update Build	
Main Actor: Startup System Goal: Analyze list of Builds to update the system. Preconditions: - Existence of the upgrade-configuration.xml file in the project root. Postconditions: - System is updated with the current Build.	
Normal Flow	
Actor	System
	1 – System loads list of Builds from the upgrade-configuration.xml file. 2 – System verifies that there is no Build marked for Downgrading. 3 – System <<extend>>“Upgrade Build”.
Alternative Flows	
2 – System verifies that there is no Build marked for Downgrading.	
Actor	System
	2.1 - System verifies that there is a Build marked for Downgrading. 2.2 - System <<extend>>“Downgrade Build”.

Table 2.4: Use Case 4: Upgrade Build.

Use Case 4: Upgrade Build	
Main Actor: Startup System Goal: Update the system with the latest Build than the current one already applied to the system. Preconditions: - Existence of the upgrade-configuration.xml file in the project root. Postconditions: - System is updated with the latest Build than the current one. - Information of builds applied to the system is persisted.	
Normal Flow	
Actor	System
	1 – System loads list of Builds from the upgrade-configuration.xml file. 2– System loads the latest build applied to the system. 3 – System creates list of builds that have not yet been applied. 4 – System verifies that the list of builds to apply is not empty. 5 – System performs the upgrade steps for each build retrieved in step 3. 6 – System does not find an error in the performed upgrade steps. 7 - System persists information of Builds applied in the system.
Alternative Flows	
4 – System verifies that the list of builds to apply is not empty.	
Actor	System
	4.1 – System verifies that the list of builds to apply is empty. 4.2 – System interrupts the execution of the use case.
6 – System does not find an error in the performed upgrade steps.	
Actor	System
	6.1 – System detects error in the performed upgrade steps. 6.2 - System persists information of Builds applied in the system in a partial way.

Table 2.5: Use Case 5: Downgrade Build.

Use Case 5: Downgrade Build	
Main Actor: Startup System Goal: Remove builds applied to the system, up to the build marked for downgrade. Preconditions: - Existence of the upgrade-configuration.xml file in the project root. Postconditions: - System is downgraded to the build marked for downgrade. - Regressed Builds have persistence removed.	
Normal Flow	
Actor	System
	1 – System loads list of Builds from the upgrade-configuration.xml file. 2– System loads the latest build applied to the system. 3 – System creates list of builds for regression. 4 – System verifies that the list of builds for regression is not empty. 5 – System performs the downgrade steps of each build retrieved in step 3. 6 – System does not detect an error in the downgrade steps performed. 7 - System removes persistence of regressed builds.
Alternative Flows	
4 – System verifies that the list of builds for regression is not empty.	
Actor	System
	4.1 – System verifies that the list of builds for regression is empty. 4.2 – System interrupts the execution of the use case.
6 – System does not detect an error in the performed upgrade steps.	
Actor	System
	6.1 – System detects error in the downgrade steps performed. 6.2- System partially removes persistence of regressed builds.

3
Project

This section presents the Project of the system developed in this work. Thus, the Architecture and Diagrams are provided.

3.1
Architecture

The upgrade configuration is a monolithic application that can be used as a dependency of your application, which can be monolithic or distributed. All organization related to packages and relationship between classes, is described through the Class Diagram, available in subsection 3.2.2.

The organization of the persistence layer, is described through the Entity relationship Diagram, available in subsection 3.2.1.

The Figure 3.1 presents the architecture of the Upgrade Configuration when used in the initialization of a monolithic application.

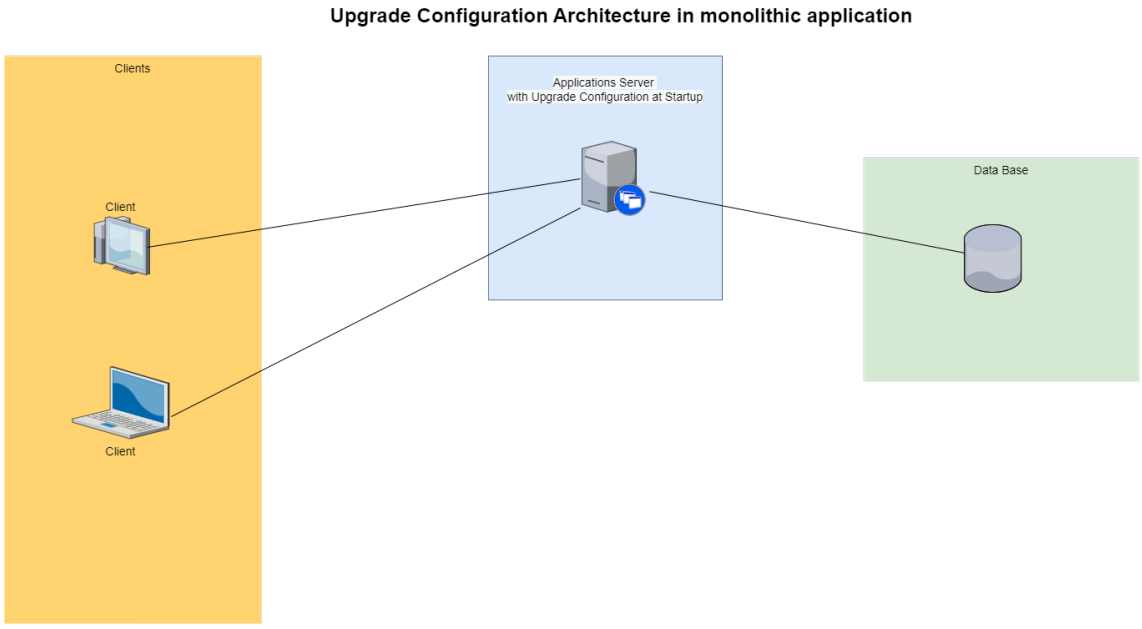


Figure 3.1: Upgrade Configuration Architecture in Monolithic Applications

The Figure 3.2 presents the architecture of the Upgrade Configuration when used in the initialization of a distributed application.

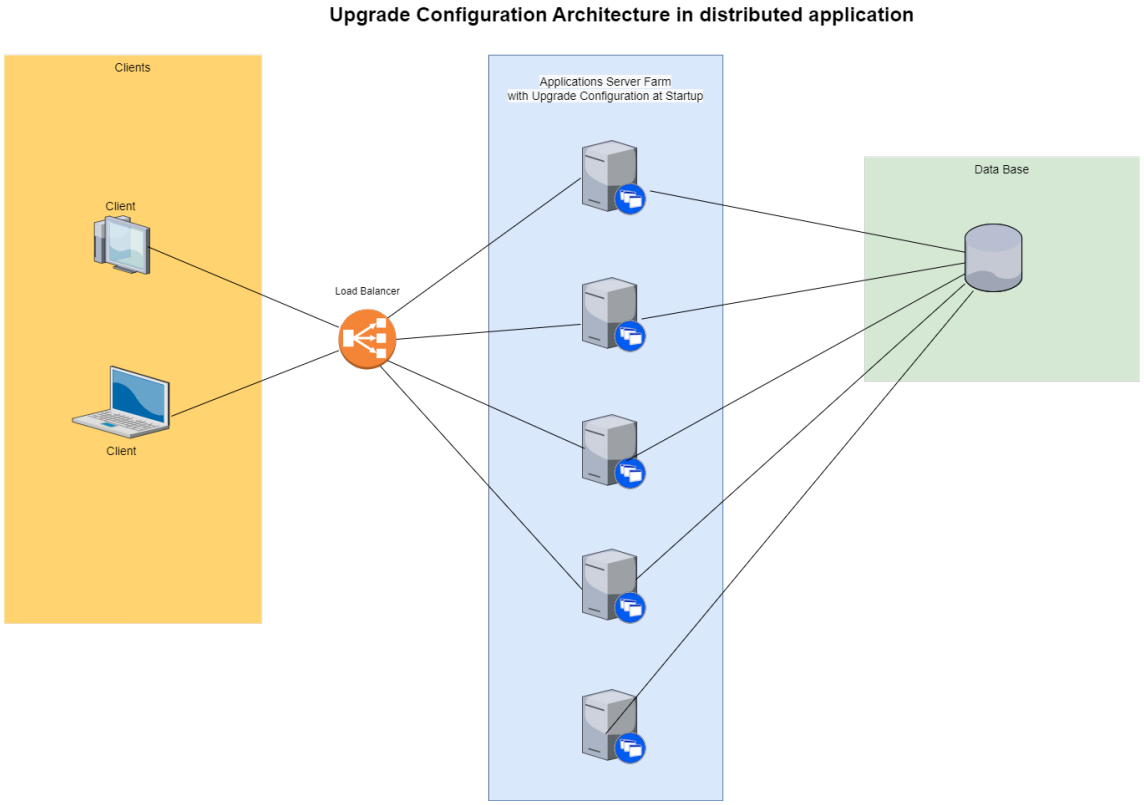


Figure 3.2: Upgrade Configuration Architecture in Distributed Applications

3.2 Diagrams

This section presents the Entity relationship Diagram and the Class Diagram.

3.2.1 Data Modeling: Entity relationship

The Entity relationship is shown in Figure 3.3.

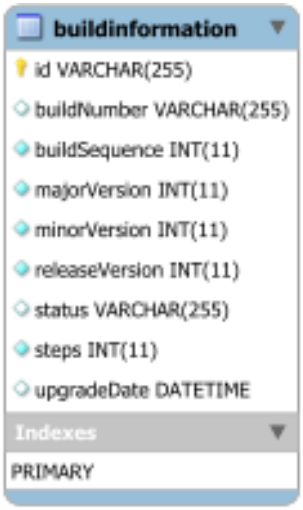


Figure 3.3: Entity Relationship

3.2.2 Class Diagram

The Class Diagram is shown in Figure 3.4 or available on GitHub at https://github.com/rcabral/upgrade-configuration/blob/main/source/upgrade-configuration/diagrams/class-diagram.svg for a better view.

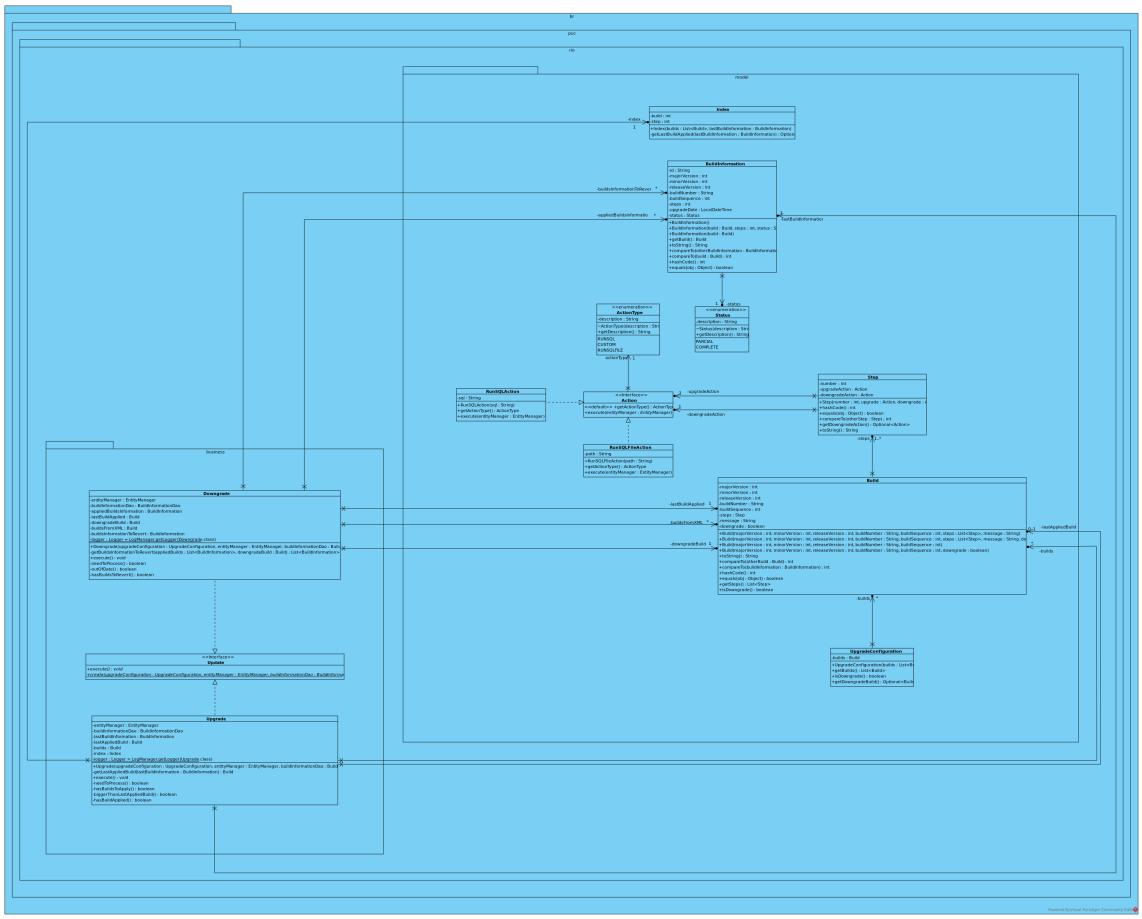


Figure 3.4: Class Diagram

4
Source Code

The source code repository, the static analysis execution report, design patterns used and good development practices adopted will be presented in this section.

4.1
Repository

The source code, properly commented, is available on the GitHub platform.

Repository URL: https://github.com/rcabral/upgrade-configuration.

4.2
Static Analysis

To perform the static analysis of the source code, the PMD Maven Plugin was used. PMD (PMD, 2022) is a source code analyzer, it finds common programming flaws like unused variables, empty catch blocks, unnecessary object creation, and so forth.

The static analysis of the code performed by PMD found no problems in the source code. The result can be seen through the report available in Figure 4.1 or through the original report available on GitHub at https://github.com/rcabral/upgrade-configuration/blob/main/source/upgrade-configuration/target/site/pmd.html.

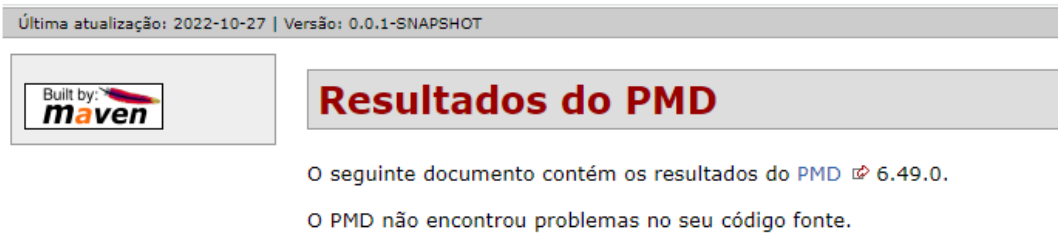


Figure 4.1: Static analysis generated by PMD Maven Plugin

4.3
Software Design Patterns and Good practices adopted

The application was developed using the object-oriented programming paradigm. And it took into account the following Design Patterns and Good practices: SOLID object-oriented programming principles (MARTIN, 1995), Design Patterns techniques (GAMMA et al., 1995), and clean code practices (MARTIN, 2009).

5
Test

This section presents the tests developed for the system,in this work. Thus, the details of Unit Tests and Integration Tests are provided.

5.1
Unit Tests

To mitigate the threat related to the presence of bugs in the source code, unit tests were developed.

The framework used to implement the unit tests, a execution report and code coverage are presented in this section.

5.1.1
Frameworks

The Frameworks adopted to develop the automatized Test Cases will be presented in this subsection.

5.1.1.1
JUnit 4

In order to take advantage of Java platform, a test framework was employed in this work. JUnit 4 (JUNIT, 2022) is a test framework for Java applications that support the development of test cases in a seamless way, such as providing primitives for stating pre and post conditions on each test case developed.

5.1.1.2
Mockito

Mockito (MOCKITO, 2022) is an open source testing framework for Java. The framework allows the creation of test double objects (mock and stubs) in automated unit tests.

It was the tool that made it possible to create mocks and stubs for implementing unit tests.

5.1.2 Execution Report

No human intervention was needed in order to test application logic. The Surefire Plugin (SUREFIRE, 2022) was used during the Test Phase of the build lifecycle to execute the unit tests and generate the report.

The execution report is composed of Summary, Package List and Test Cases. It´s presented through the Figures 5.1, 5.2, 5.3, 5.4, 5.5, 5.6, 5.7 or through the original report available on GitHub at https://github.com/rcabral/upgrade-configuration/blob/main/source/upgrade-configuration/target/site/surefire-report.html.

Surefire Report

Summary

[Summary] [Package List] [Test Cases]

Tests	Errors	Failures	Skipped	Success Rate	Time
70	0	0	0	100%	1,074

Note: failures are anticipated and checked for with assertions while errors are unanticipated.

Package List

[Summary] [Package List] [Test Cases]

Package	Tests	Errors	Failures	Skipped	Success Rate	Time
br.puc.rio.sample	1	0	0	0	100%	0
br.puc.rio.dao	12	0	0	0	100%	0
br.puc.rio.business	12	0	0	0	100%	1,067
br.puc.rio.model	41	0	0	0	100%	0,002
br.puc.rio.converter	4	0	0	0	100%	0,005

Note: package statistics are not computed recursively, they only sum up all of its testsuites numbers.

br.puc.rio.sample

	Class	Tests	Errors	Failures	Skipped	Success Rate	Time
	HelloCustomActionTest	1	0	0	0	100%	0

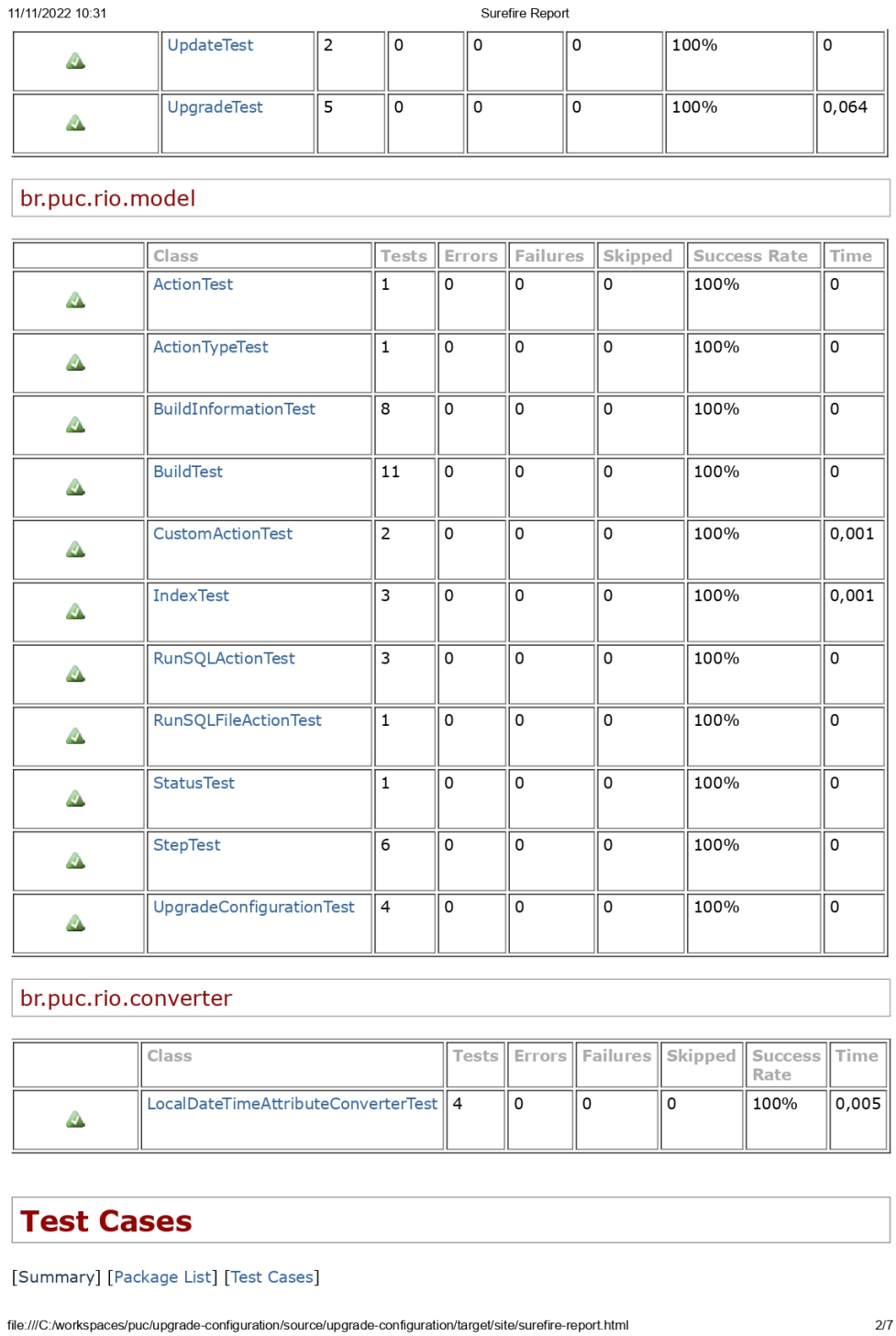
br.puc.rio.dao

	Class	Tests	Errors	Failures	Skipped	Success Rate	Time
	BuildInformationDaoTest	12	0	0	0	100%	0

br.puc.rio.business

	Class	Tests	Errors	Failures	Skipped	Success Rate	Time
	DowngradeTest	5	0	0	0	100%	1,003

Figure 5.1: Execution Report - Part 1



11/11/2022 10:31 Surefire Report

	UpdateTest	2	0	0	0	100%	0
	UpgradeTest	5	0	0	0	100%	0,064

br.puc.rio.model

	Class	Tests	Errors	Failures	Skipped	Success Rate	Time
	ActionTest	1	0	0	0	100%	0
	ActionTypeTest	1	0	0	0	100%	0
	BuildInformationTest	8	0	0	0	100%	0
	BuildTest	11	0	0	0	100%	0
	CustomActionTest	2	0	0	0	100%	0,001
	IndexTest	3	0	0	0	100%	0,001
	RunSQLActionTest	3	0	0	0	100%	0
	RunSQLFileActionTest	1	0	0	0	100%	0
	StatusTest	1	0	0	0	100%	0
	StepTest	6	0	0	0	100%	0
	UpgradeConfigurationTest	4	0	0	0	100%	0

br.puc.rio.converter

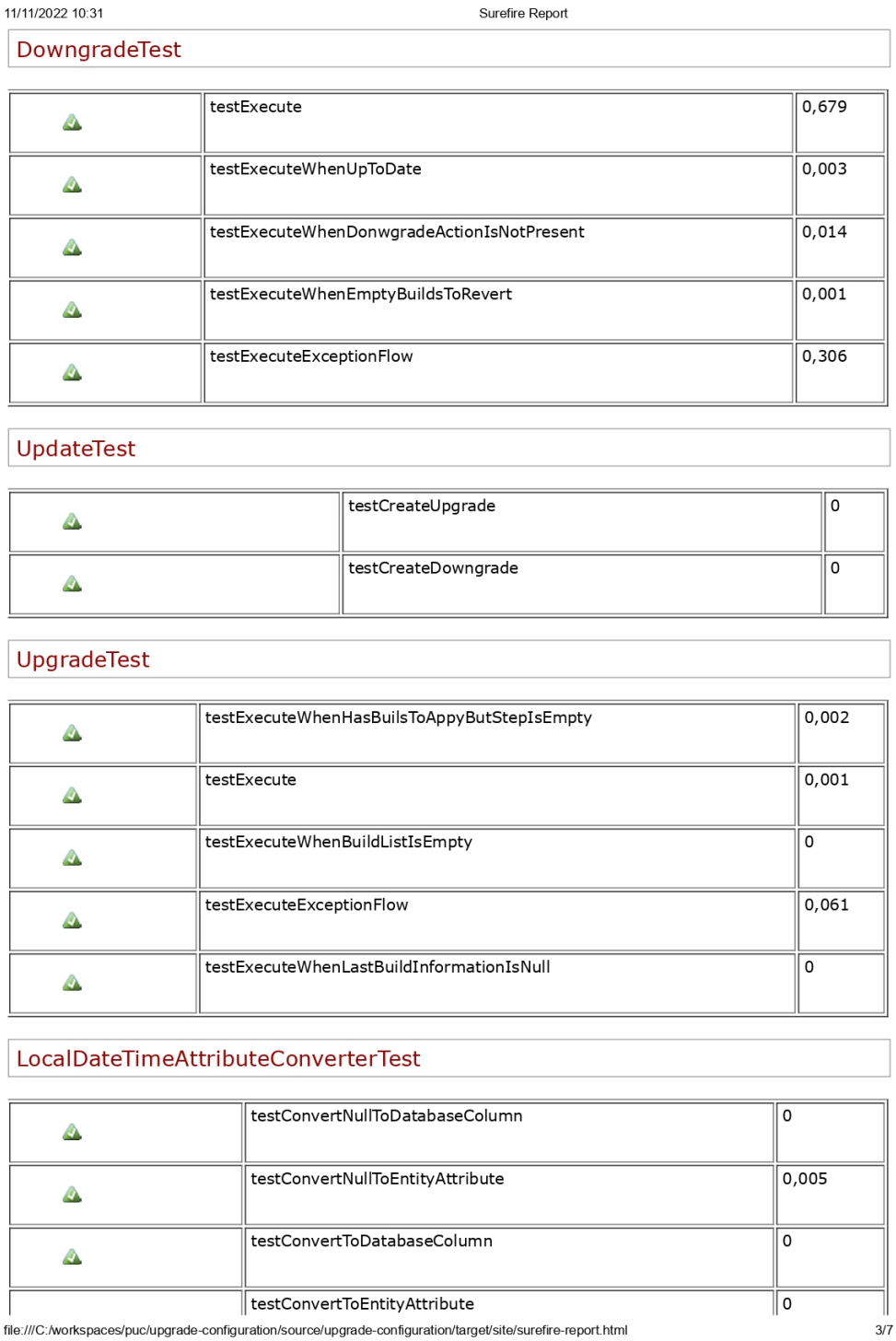
	Class	Tests	Errors	Failures	Skipped	Success Rate	Time
	LocalDateTimeAttributeConverterTest	4	0	0	0	100%	0,005

Test Cases

[Summary] [Package List] [Test Cases]

file:///C:/workspaces/puc/upgrade-configuration/source/upgrade-configuration/target/site/surefire-report.html 2/7

Figure 5.2: Execution Report - Part 2



11/11/2022 10:31 Surefire Report

DowngradeTest

	testExecute	0,679
	testExecuteWhenUpToDate	0,003
	testExecuteWhenDonwgradeActionIsNotPresent	0,014
	testExecuteWhenEmptyBuildsToRevert	0,001
	testExecuteExceptionFlow	0,306

UpdateTest

	testCreateUpgrade	0
	testCreateDowngrade	0

UpgradeTest

	testExecuteWhenHasBuilsToAppyButStepIsEmpty	0,002
	testExecute	0,001
	testExecuteWhenBuildListIsEmpty	0
	testExecuteExceptionFlow	0,061
	testExecuteWhenLastBuildInformationIsNull	0

LocalDateTimeAttributeConverterTest

	testConvertNullToDatabaseColumn	0
	testConvertNullToEntityAttribute	0,005
	testConvertToDatabaseColumn	0
	testConvertToEntityAttribute	0

file:///C:/workspaces/puc/upgrade-configuration/source/upgrade-configuration/target/site/surefire-report.html 3/7

Figure 5.3: Execution Report - Part 3

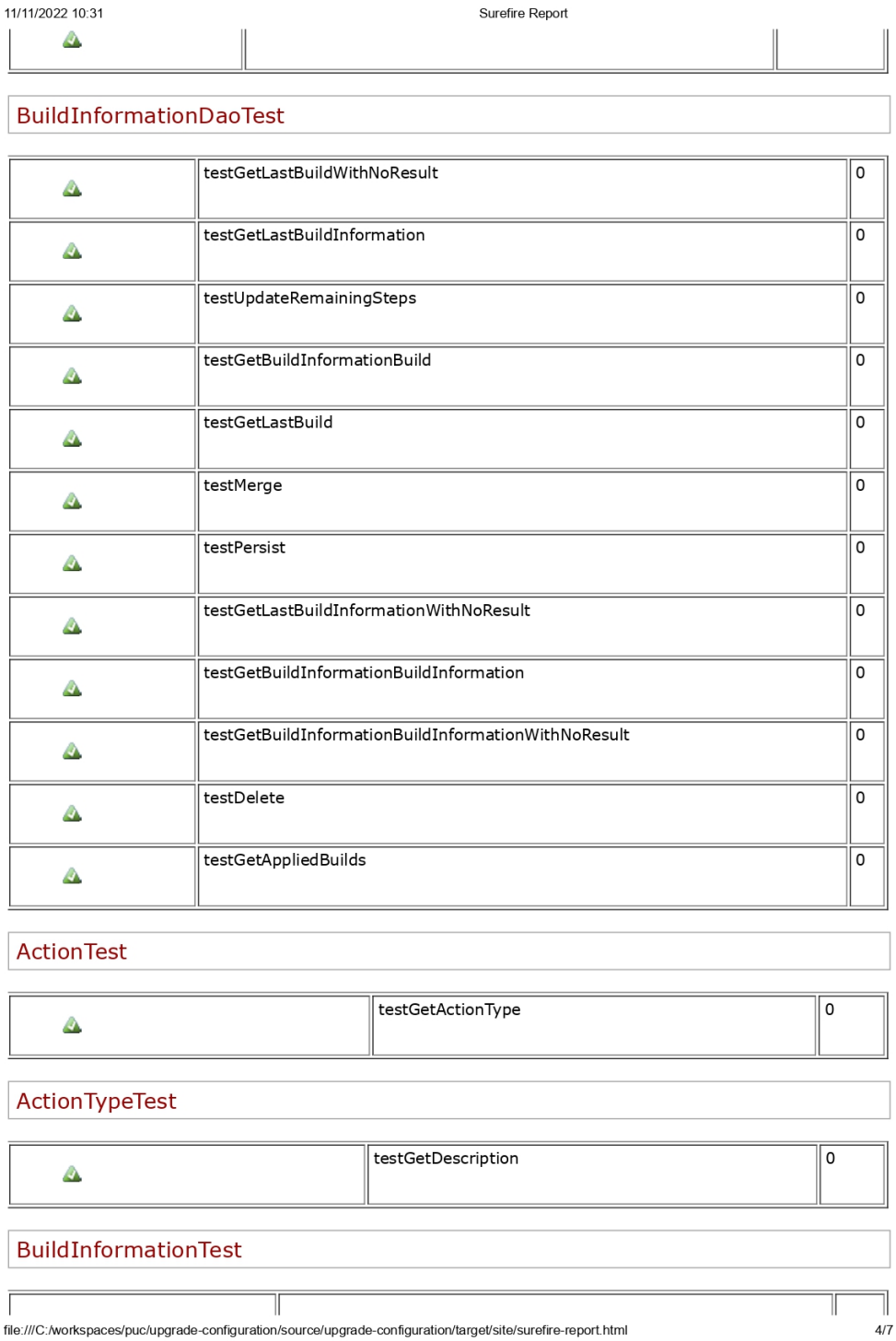
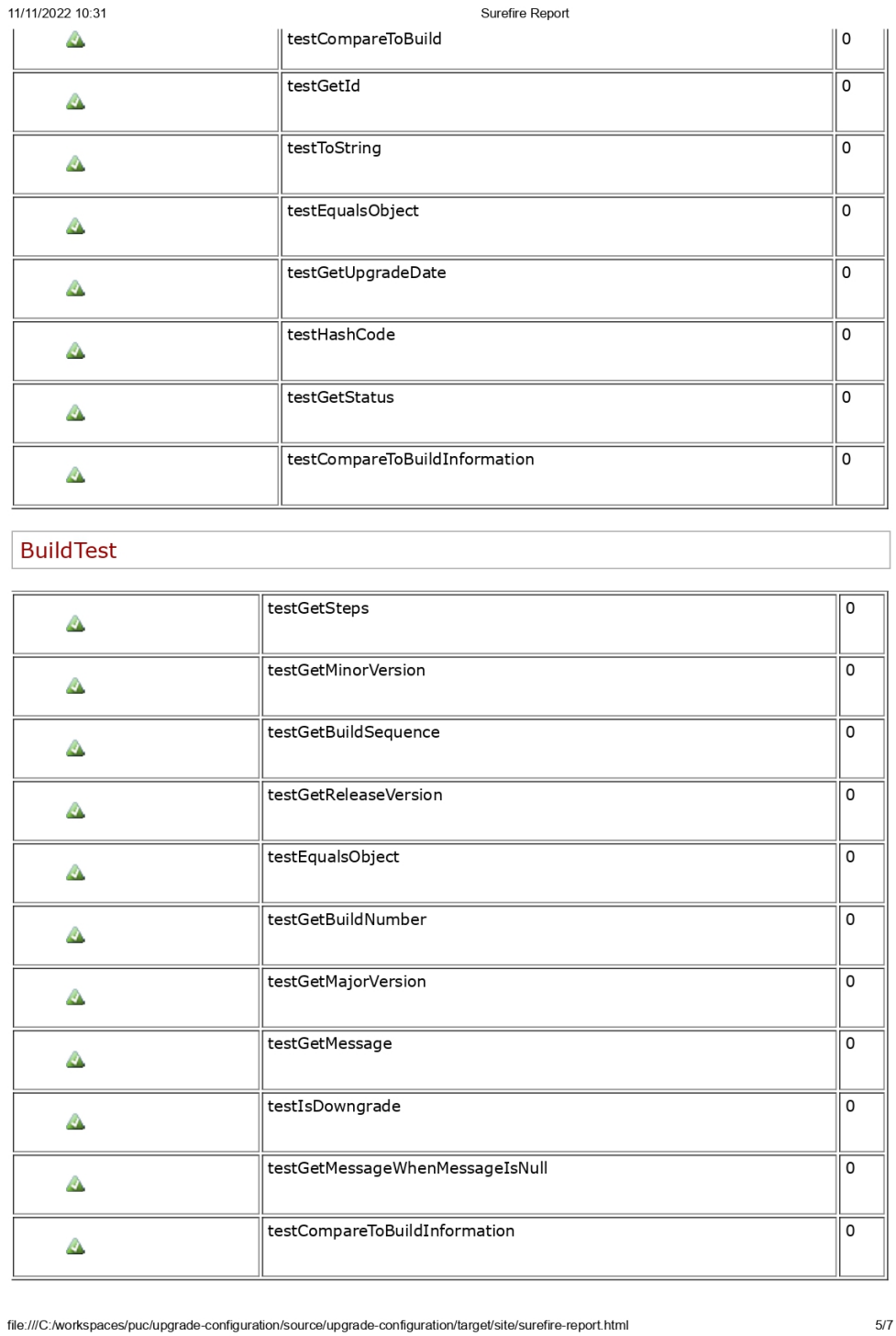


Figure 5.4: Execution Report - Part 4



11/11/2022 10:31 Surefire Report

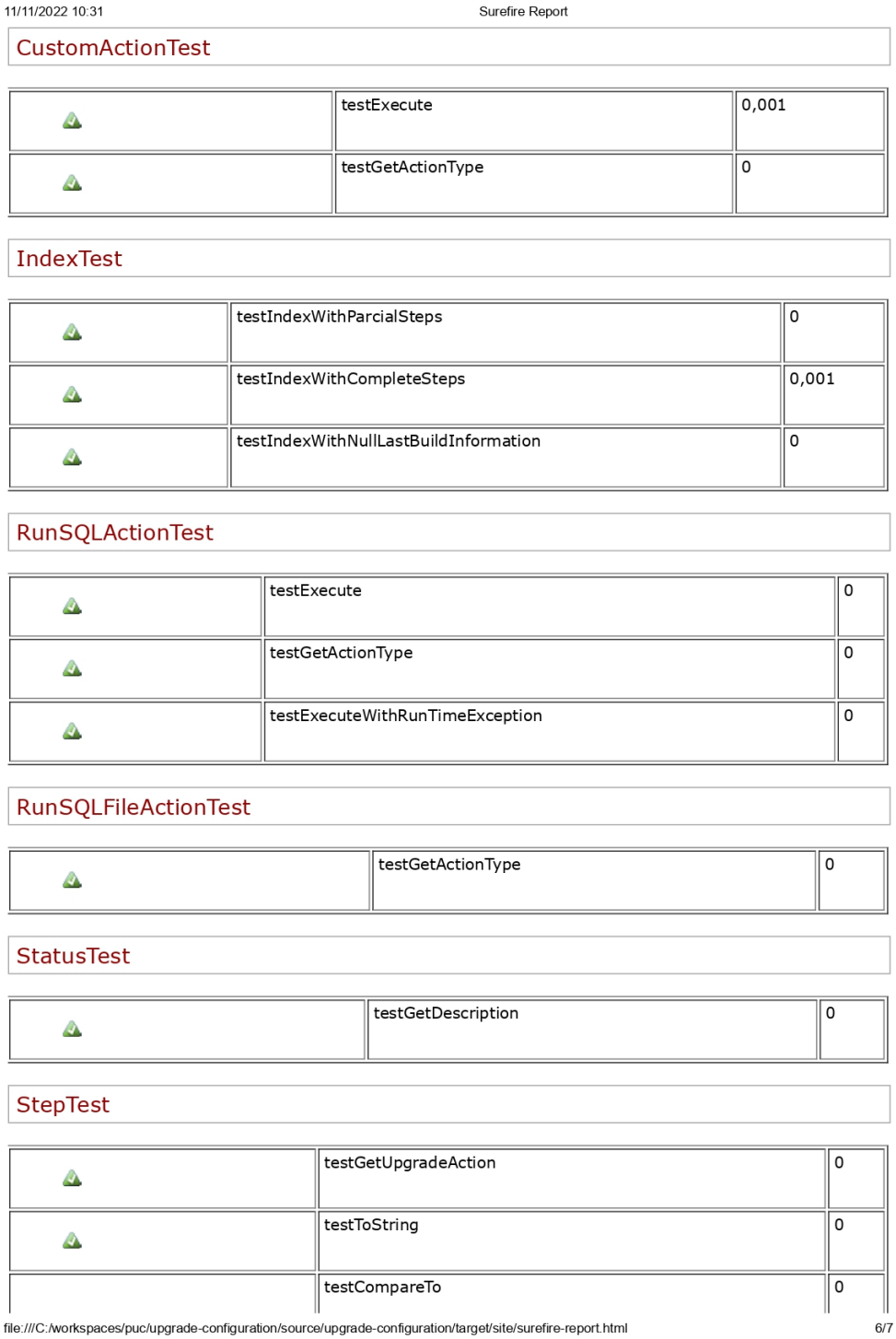
	testCompareToBuild	0
	testGetId	0
	testToString	0
	testEqualsObject	0
	testGetUpgradeDate	0
	testHashCode	0
	testGetStatus	0
	testCompareToBuildInformation	0

BuildTest

	testGetSteps	0
	testGetMinorVersion	0
	testGetBuildSequence	0
	testGetReleaseVersion	0
	testEqualsObject	0
	testGetBuildNumber	0
	testGetMajorVersion	0
	testGetMessage	0
	testIsDowngrade	0
	testGetMessageWhenMessageIsNull	0
	testCompareToBuildInformation	0

file:///C:/workspaces/puc/upgrade-configuration/source/upgrade-configuration/target/site/surefire-report.html 5/7

Figure 5.5: Execution Report - Part 5



11/11/2022 10:31 Surefire Report

CustomActionTest

	testExecute	0,001
	testGetActionType	0

IndexTest

	testIndexWithParcialSteps	0
	testIndexWithCompleteSteps	0,001
	testIndexWithNullLastBuildInformation	0

RunSQLActionTest

	testExecute	0
	testGetActionType	0
	testExecuteWithRunTimeException	0

RunSQLFileActionTest

	testGetActionType	0

StatusTest

	testGetDescription	0

StepTest

	testGetUpgradeAction	0
	testToString	0
	testCompareTo	0

file:///C:/workspaces/puc/upgrade-configuration/source/upgrade-configuration/target/site/surefire-report.html 6/7

Figure 5.6: Execution Report - Part 6

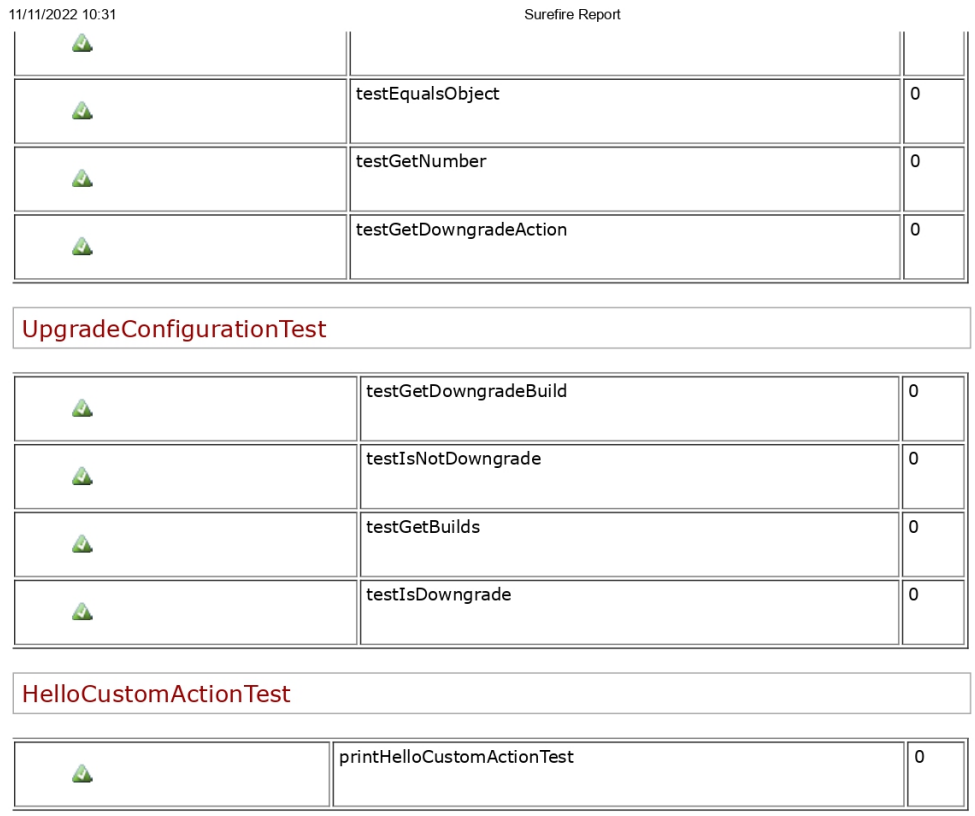


Figure 5.7: Execution Report - Part 7

5.1.3 Code Coverage Report

No human intervention was needed to generate the code coverage report. The Jacoco Maven Plugin (JACOCO, 2022) was used during the Test Phase of the build lifecycle to generate the report.The Plugin provides the runtime agent to the tests and allows the report creation during the build.

The Code Coverage carried out by Jacoco reached the result of 90%. The result can be seen through the report available in Figure 5.8 or through the original report available on GitHub at https://github.com/rcabral/upgrade-configuration/blob/main/source/upgrade-configuration/target/site/jacoco/index.html.

An observation is that we reached 90% code coverage with automated unit tests, where we were able to exercise all the application's business logic, without human intervention. Only two classes have not been tested with automated unit tests, the UpgradeConfigurationController and JPAUtil classes, they have static class calls that make it impossible to create stunt objects (mock and stubs) for unit tests. However, these two classes are tested in the integration test presented in section 5.2.

The Code Coverage Report report is presented through the Figure 5.8.

Upgradeconfiguration

Upgradeconfiguration

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
br.puc.rio.model		88%		61%	30	101	35	245	6	68	0	10
br.puc.rio.controller		0%		n/a	4	4	16	16	4	4	1	1
br.puc.rio.util		0%		n/a	2	2	3	3	2	2	1	1
br.puc.rio.business		100%		87%	7	43	0	91	0	16	0	3
br.puc.rio.dao		100%		100%	0	11	0	60	0	10	0	1
br.puc.rio.converter		100%		100%	0	5	0	3	0	3	0	1
br.puc.rio.sample		100%		n/a	0	2	0	3	0	2	0	1
Total	187 of 1.810	90%	33 of 126	74%	43	168	54	421	12	105	2	18

Figure 5.8: Code Coverage Report

5.2 Integration Tests

To mitigate the threat related to the presence of bugs after the integration of all parts of the system, upgrade and downgrade process integration test were performed.

5.2.1 Upgrade Build Process

This integration test exercises the Upgrade process, performing the build upgrade from version 14.0.0.180122 to version 25.0.0.101022. The video with the evidence of the entire execution

process is available on GitHub at https://github.com/rcabral/upgrade-configuration/raw/main/source/upgrade-configuration/src/test/resources/integration-tests-evidences/upgrade-integration-test.mp4, but can also be observed through Figure 5.9. And the Execution Log is available in Figure 5.10.

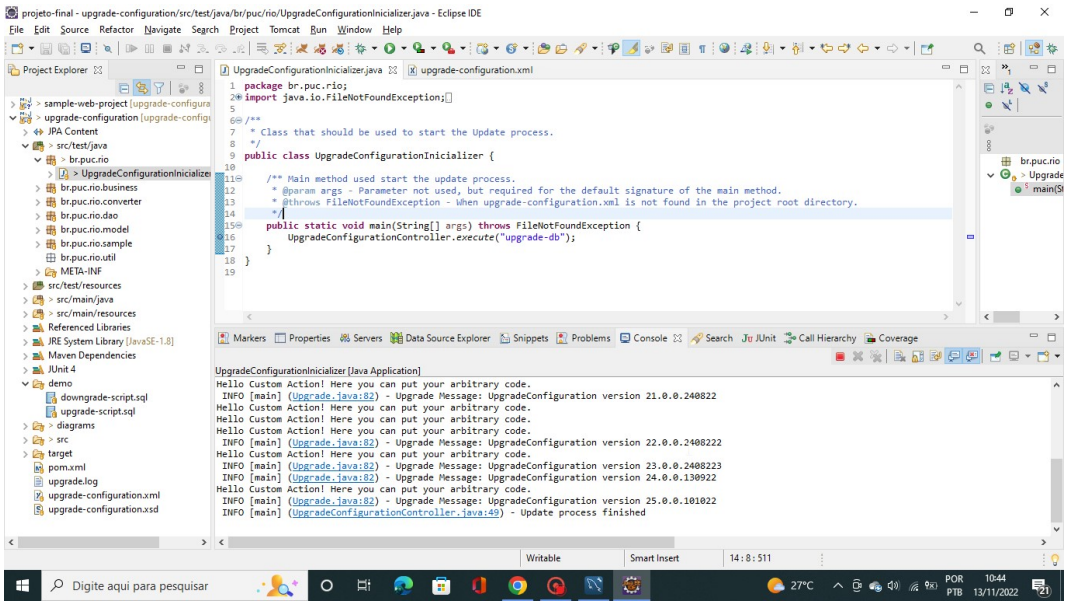


Figure 5.9: Upgrade Integration Test evidence

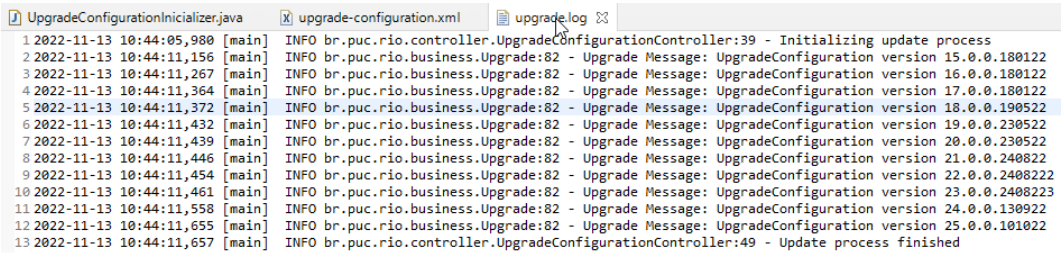


Figure 5.10: Upgrade Integration Test Log

5.2.2 Downgrade Build Process

This integration test exercises the Downgrade process, downgrading the build from version 25.0.0.101022 to version 14.0.0.180122. The video with the evidence of the entire execution process is available on GitHub at https://github.com/rcabral/upgrade-configuration/raw/main/source/upgrade-configuration/src/test/resources/integration-tests-evidences/downgrade-integration-test.mp4, but can also be observed through Figure 5.11. And the Execution Log is available in Figure 5.12.

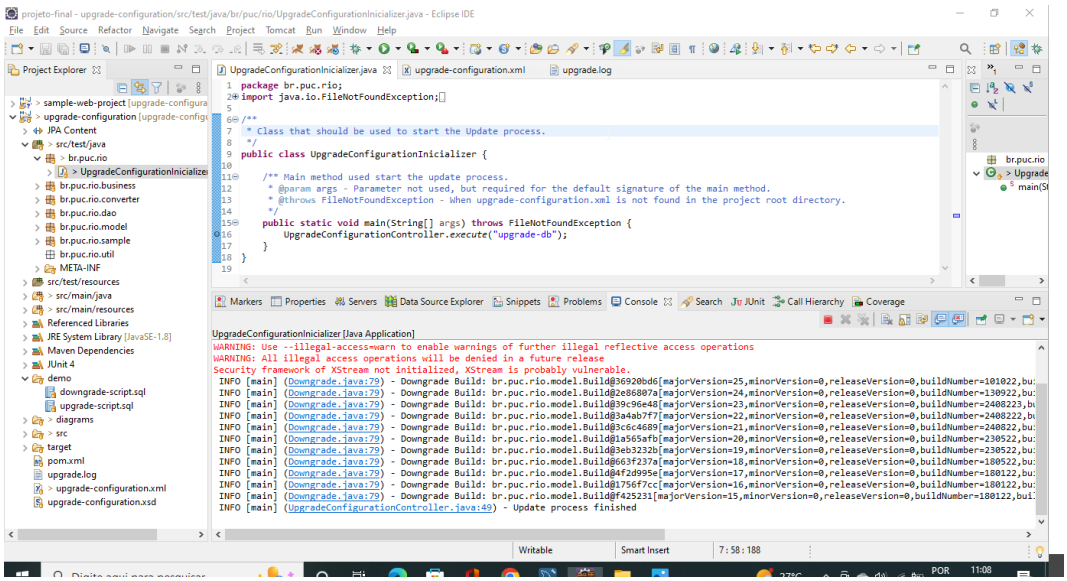


Figure 5.11: Downgrade Integration Test evidence

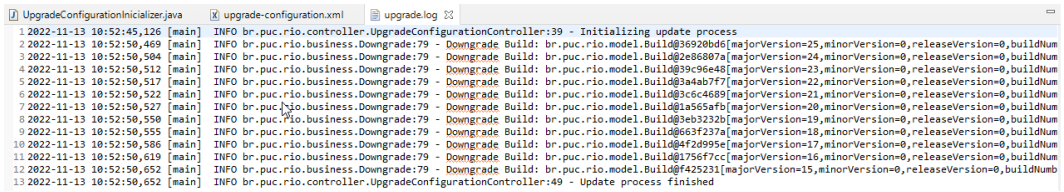


Figure 5.12: Downgrade Integration Test Log

6
User Manual

Upgrade Configuration is a version control for the database, so the developer has the power to migrate it easily and with security. Allows database evolutions to be shipped along with the application and run automatically at startup. This facilitates the continuous application delivery process and favors the expansion of the agile development culture and devops.

6.1
Prerequisites

List:

- JDK 8 or greater;
- JPA 2 or greater;
- Maven 3 or greater.

6.2
How to Configure

Upgrade Configuration uses Maven to manage its dependencies. So, to add Upgrade Configuration to your project, all you need to do is add the following dependency:

```
<dependency>
    <groupId>br.puc-rio</groupId>
    <artifactId>upgrade-configuration</artifactId>
    <version>1.0</version>
</dependency>
```

Maven will take care of downloading all required dependencies.

6.3 How to Use

6.3.1 Creating your first Build

By convention, to manage your Builds, just create the file upgrade-configuration.xml in the root of your project. The XML code 1, also present at https://github.com/rcabral/upgrade-configuration/raw/main/source/upgrade-configuration/demo/first-build/upgrade-configuration.xml, presents an example of creating your first Build. As can be seen, this Build is composed of 2 steps, with an upgrade and a downgrade action.

Code 1: First Build

```
<?xml version="1.0" encoding="UTF-8"?>
<UpgradeConfiguration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="upgrade-configuration.xsd">
<!-- List of Builds -->
<build majorVersion="1" minorVersion="0" releaseVersion="0" buildNumber="151122" buildSequence="0">
  <!-- List of Steps -->
    <steps>
      <Step number="0">
        <!-- Upgrade Action -->
        <upgrade class="br.puc.rio.model.RunSQLAction">
          <sql>CREATE TABLE MY_FIRST_TABLE(id int(4) AUTO_INCREMENT, name varchar(30) NOT NULL,PRIMARY KEY (id));</sql>
        </upgrade>
         <!-- Downgrade Action -->
        <downgrade class="br.puc.rio.model.RunSQLAction">
          <sql>DROP TABLE MY_FIRST_TABLE;</sql>
        </downgrade>
      </Step>
      <Step number="1">
        <!-- Upgrade Action -->
        <upgrade class="br.puc.rio.model.RunSQLAction">
          <sql>CREATE TABLE MY_SECOND_TABLE(id int(4) AUTO_INCREMENT ,name varchar(30) NOT NULL,PRIMARY KEY (id));</sql>
        </upgrade>
        <!-- Downgrade Action -->
        <downgrade class="br.puc.rio.model.RunSQLAction">
          <sql>DROP TABLE MY_SECOND_TABLE;</sql>
        </downgrade>
      </Step>
    </steps>
    <message>UpgradeConfiguration version 1.0.0.151122</message>
  </build>
</UpgradeConfiguration>
```

One note is that the downgrade action is not mandatory, only upgrade.

6.3.2 Running The Upgrade Configuration

To run the upgrade configuration and synchronize your builds declared in the XML file with the database, just run the method named execute, from the br.puc.rio.controller.UpgradeConfigurationController class.

It is recommended that this method be executed at server startup, so that every build change is synchronized. The code 2 presents an example of creating a component with Spring Boot, so that the method is executed at server startup.

Code 2: Running The Upgrade Configuration on Startup

```
1  package br.puc.rio;
2
3  import br.puc.rio.controller.UpgradeConfigurationController;
4  import javax.annotation.PostConstruct;
5  import javax.annotation.PreDestroy;
6  import java.io.FileNotFoundException;
7  import org.springframework.stereotype.Component;
8
9  @Component
10 public class StartupComponent {
11
12   @PostConstruct
13   private void init() throws FileNotFoundException {
14     UpgradeConfigurationController.execute();
15   }
16
17 }
```

6.3.3 Upgrade your Build

To create a new build and synchronize with your database. Add a new build to the upgrade-configuration.xml file and run the upgrade configuration as described in subsection 6.3.2.

An example of creating a new build in the upgrade-configuration.xml file can be seen in XML code 3. And it is also available on GitHub at https://github.com/rcabral/upgrade-configuration/raw/main/source/upgrade-configuration/demo/upgrade-build/upgrade-configuration.xml. Note that the creation of the new build starts from line 35.

Code 3: Upgrade Build

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <UpgradeConfiguration
3    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4    xsi:noNamespaceSchemaLocation="upgrade-configuration.xsd">
5    <!-- List of Builds -->
6    <build majorVersion="1" minorVersion="0" releaseVersion="0"
7      buildNumber="151122" buildSequence="0">
8      <!-- List of Steps -->
9      <steps>
10       <Step number="0">
11         <!-- Upgrade Action -->
12         <upgrade class="br.puc.rio.model.RunSQLAction">
13           <sql>CREATE TABLE MY_FIRST_TABLE(id int(4) AUTO_INCREMENT,
                 name
14             varchar(30) NOT NULL,PRIMARY KEY (id));</sql>
15         </upgrade>
16         <!-- Downgrade Action -->
17         <downgrade class="br.puc.rio.model.RunSQLAction">
18           <sql>DROP TABLE MY_FIRST_TABLE;</sql>
19         </downgrade>
20       </Step>
21       <Step number="1">
22         <!-- Upgrade Action -->
23         <upgrade class="br.puc.rio.model.RunSQLAction">
24           <sql>CREATE TABLE MY_SECOND_TABLE(id int(4) AUTO_INCREMENT
                 ,name
25             varchar(30) NOT NULL,PRIMARY KEY (id));</sql>
26         </upgrade>
27         <!-- Downgrade Action -->
28         <downgrade class="br.puc.rio.model.RunSQLAction">
29           <sql>DROP TABLE MY_SECOND_TABLE;</sql>
30         </downgrade>
31       </Step>
32     </steps>
33     <message>UpgradeConfiguration version 1.0.0.151122</message>
```

```
34   </build>
35   <!-- New Build declaration -->
36   <build majorVersion="2" minorVersion="0" releaseVersion="0"
37     buildNumber="161122" buildSequence="0">
38     <!-- List of Steps -->
39     <steps>
40       <Step number="0">
41         <!-- Upgrade Action -->
42         <upgrade class="br.puc.rio.model.RunSQLAction">
43           <sql>CREATE TABLE MY_THIRD_TABLE(id int(4) AUTO_INCREMENT,
                 name
44             varchar(30) NOT NULL,PRIMARY KEY (id));</sql>
45         </upgrade>
46         <!-- Downgrade Action -->
47         <downgrade class="br.puc.rio.model.RunSQLAction">
48           <sql>DROP TABLE MY_THIRD_TABLE;</sql>
49         </downgrade>
50       </Step>
51       <Step number="1">
52         <!-- Upgrade Action -->
53         <upgrade class="br.puc.rio.model.RunSQLAction">
54           <sql>CREATE TABLE FOURTH (id int(4) AUTO_INCREMENT,name
55             varchar(30) NOT NULL,PRIMARY KEY (id));</sql>
56         </upgrade>
57         <!-- Downgrade Action -->
58         <downgrade class="br.puc.rio.model.RunSQLAction">
59           <sql>DROP TABLE FOURTH ;</sql>
60         </downgrade>
61       </Step>
62     </steps>
63     <message>UpgradeConfiguration version 2.0.0.161122</message>
64   </build>
65 </UpgradeConfiguration>
```

6.3.4 Downgrade your Build

In some cases it may be valuable to downgrade your database to a point earlier than the current one. In these cases, choose a previous build and set the downgrade attribute to true. All later builds will be regressed.

The XML code 4 shows an example where the Build 2.0.0.161122 is being downgraded to Build version 1.0.0.151122. It´s also available on GitHub at https://github.com/rcabral/upgrade-configuration/raw/main/source/upgrade-configuration/demo/downgrade-build/upgrade-configuration.xml. Note that in line 7 the downgrade attribute is set to true.

Code 4: Downgrade Build

```
<?xml version="1.0" encoding="UTF-8"?>
<UpgradeConfiguration
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="upgrade-configuration.xsd">
  <!-- List of Builds -->
  <build majorVersion="1" minorVersion="0" releaseVersion="0"
    buildNumber="151122" buildSequence="0" downgrade="true">
    <!-- List of Steps -->
    <steps>
      <Step number="0">
        <!-- Upgrade Action -->
        <upgrade class="br.puc.rio.model.RunSQLAction">
          <sql>CREATE TABLE MY_FIRST_TABLE(id int(4) AUTO_INCREMENT,
              name
            varchar(30) NOT NULL,PRIMARY KEY (id));</sql>
        </upgrade>
        <!-- Downgrade Action -->
        <downgrade class="br.puc.rio.model.RunSQLAction">
          <sql>DROP TABLE MY_FIRST_TABLE;</sql>
        </downgrade>
      </Step>
      <Step number="1">
        <!-- Upgrade Action -->
        <upgrade class="br.puc.rio.model.RunSQLAction">
          <sql>CREATE TABLE MY_SECOND_TABLE(id int(4) AUTO_INCREMENT
              ,name
            varchar(30) NOT NULL,PRIMARY KEY (id));</sql>
        </upgrade>
        <!-- Downgrade Action -->
        <downgrade class="br.puc.rio.model.RunSQLAction">
          <sql>DROP TABLE MY_SECOND_TABLE;</sql>
        </downgrade>
      </Step>
    </steps>
    <message>UpgradeConfiguration version 1.0.0.151122</message>
  </build>
  <!-- New Build declaration -->
  <build majorVersion="2" minorVersion="0" releaseVersion="0"
    buildNumber="161122" buildSequence="0">
    <!-- List of Steps -->
    <steps>
      <Step number="0">
        <!-- Upgrade Action -->
        <upgrade class="br.puc.rio.model.RunSQLAction">
          <sql>CREATE TABLE MY_THIRD_TABLE(id int(4) AUTO_INCREMENT,
              name
            varchar(30) NOT NULL,PRIMARY KEY (id));</sql>
        </upgrade>
        <!-- Downgrade Action -->
        <downgrade class="br.puc.rio.model.RunSQLAction">
```

```
48            <sql>DROP TABLE MY_THIRD_TABLE;</sql>
49          </downgrade>
50        </Step>
51        <Step number="1">
52          <!-- Upgrade Action -->
53          <upgrade class="br.puc.rio.model.RunSQLAction">
54            <sql>CREATE TABLE FOURTH (id int(4) AUTO_INCREMENT,name
55              varchar(30) NOT NULL,PRIMARY KEY (id));</sql>
56          </upgrade>
57          <!-- Downgrade Action -->
58          <downgrade class="br.puc.rio.model.RunSQLAction">
59            <sql>DROP TABLE FOURTH ;</sql>
60          </downgrade>
61        </Step>
62      </steps>
63      <message>UpgradeConfiguration version 2.0.0.161122</message>
64    </build>
65 </UpgradeConfiguration>
```

6.3.5 Execution Log

The execution log is available through the upgrade.log file at the root of your project. An example of the Log can be seen in Figure 6.1.

upgrade.log

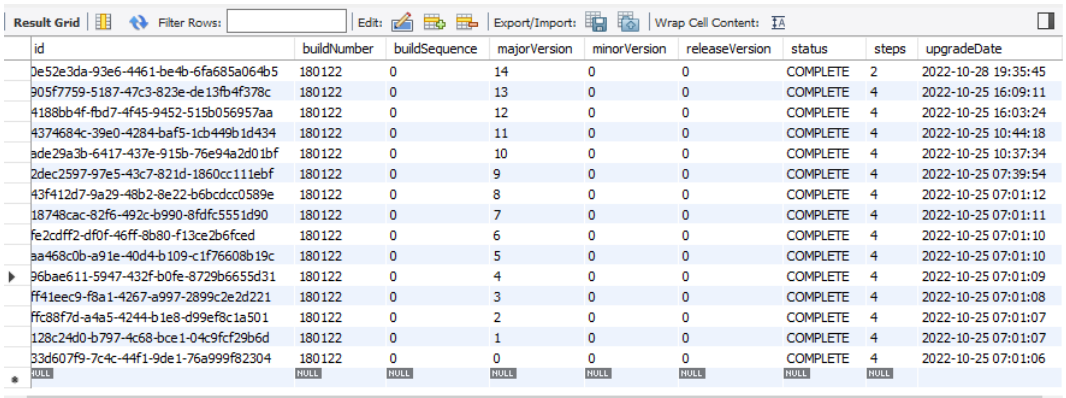
```
2022-11-13 10:44:05,980 [main]  INFO br.puc.rio.controller.UpgradeConfigurationController:39 - Initializing update process
2022-11-13 10:44:11,156 [main]  INFO br.puc.rio.business.Upgrade:82 - Upgrade Message: UpgradeConfiguration version 15.0.0.180122
2022-11-13 10:44:11,267 [main]  INFO br.puc.rio.business.Upgrade:82 - Upgrade Message: UpgradeConfiguration version 16.0.0.180122
2022-11-13 10:44:11,364 [main]  INFO br.puc.rio.business.Upgrade:82 - Upgrade Message: UpgradeConfiguration version 17.0.0.180122
2022-11-13 10:44:11,372 [main]  INFO br.puc.rio.business.Upgrade:82 - Upgrade Message: UpgradeConfiguration version 18.0.0.190522
2022-11-13 10:44:11,432 [main]  INFO br.puc.rio.business.Upgrade:82 - Upgrade Message: UpgradeConfiguration version 19.0.0.230522
2022-11-13 10:44:11,439 [main]  INFO br.puc.rio.business.Upgrade:82 - Upgrade Message: UpgradeConfiguration version 20.0.0.230522
2022-11-13 10:44:11,446 [main]  INFO br.puc.rio.business.Upgrade:82 - Upgrade Message: UpgradeConfiguration version 21.0.0.240822
2022-11-13 10:44:11,454 [main]  INFO br.puc.rio.business.Upgrade:82 - Upgrade Message: UpgradeConfiguration version 22.0.0.2408222
2022-11-13 10:44:11,461 [main]  INFO br.puc.rio.business.Upgrade:82 - Upgrade Message: UpgradeConfiguration version 23.0.0.2408223
2022-11-13 10:44:11,558 [main]  INFO br.puc.rio.business.Upgrade:82 - Upgrade Message: UpgradeConfiguration version 24.0.0.130922
2022-11-13 10:44:11,655 [main]  INFO br.puc.rio.business.Upgrade:82 - Upgrade Message: UpgradeConfiguration version 25.0.0.101022
2022-11-13 10:44:11,657 [main]  INFO br.puc.rio.controller.UpgradeConfigurationController:49 - Update process finished
2022-11-13 10:52:45,126 [main]  INFO br.puc.rio.controller.UpgradeConfigurationController:39 - Initializing update process
2022-11-13 10:52:50,469 [main]  INFO br.puc.rio.business.Downgrade:79 - Downgrade Build: br.puc.rio.model.Build@36920bd6[majorVersion=25,
2022-11-13 10:52:50,504 [main]  INFO br.puc.rio.business.Downgrade:79 - Downgrade Build: br.puc.rio.model.Build@2e86807a[majorVersion=24,
2022-11-13 10:52:50,512 [main]  INFO br.puc.rio.business.Downgrade:79 - Downgrade Build: br.puc.rio.model.Build@39c96e48[majorVersion=23,
2022-11-13 10:52:50,517 [main]  INFO br.puc.rio.business.Downgrade:79 - Downgrade Build: br.puc.rio.model.Build@3a4ab7f7[majorVersion=22,
2022-11-13 10:52:50,522 [main]  INFO br.puc.rio.business.Downgrade:79 - Downgrade Build: br.puc.rio.model.Build@3c6c4689[majorVersion=21,
2022-11-13 10:52:50,527 [main]  INFO br.puc.rio.business.Downgrade:79 - Downgrade Build: br.puc.rio.model.Build@1a565afb[majorVersion=20,
2022-11-13 10:52:50,550 [main]  INFO br.puc.rio.business.Downgrade:79 - Downgrade Build: br.puc.rio.model.Build@3eb3232b[majorVersion=19,
2022-11-13 10:52:50,555 [main]  INFO br.puc.rio.business.Downgrade:79 - Downgrade Build: br.puc.rio.model.Build@663f237a[majorVersion=18,
2022-11-13 10:52:50,586 [main]  INFO br.puc.rio.business.Downgrade:79 - Downgrade Build: br.puc.rio.model.Build@4f2d995e[majorVersion=17,
2022-11-13 10:52:50,619 [main]  INFO br.puc.rio.business.Downgrade:79 - Downgrade Build: br.puc.rio.model.Build@1756f7cc[majorVersion=16,
2022-11-13 10:52:50,652 [main]  INFO br.puc.rio.business.Downgrade:79 - Downgrade Build: br.puc.rio.model.Build@f425231[majorVersion=15,m
2022-11-13 10:52:50,652 [main]  INFO br.puc.rio.controller.UpgradeConfigurationController:49 - Update process finished
2022-11-13 12:11:10,357 [main]  INFO br.puc.rio.business.Upgrade:82 - Upgrade Message: null
2022-11-13 12:11:10,369 [main]  INFO br.puc.rio.business.Upgrade:82 - Upgrade Message: null
2022-11-13 12:11:10,384 [main] ERROR br.puc.rio.business.Upgrade:84 - Upgrade Build execution error:build Step:0
2022-11-13 12:11:10,391 [main]  INFO br.puc.rio.business.Upgrade:82 - Upgrade Message: null
2022-11-13 12:11:10,504 [main]  INFO br.puc.rio.business.Downgrade:79 - Downgrade Build: build
2022-11-13 12:11:10,511 [main]  INFO br.puc.rio.business.Downgrade:79 - Downgrade Build: build
2022-11-13 12:11:10,524 [main]  INFO br.puc.rio.business.Downgrade:79 - Downgrade Build: build
2022-11-13 12:11:10,565 [main]  INFO br.puc.rio.business.Downgrade:79 - Downgrade Build: build
2022-11-13 12:11:10,568 [main]  INFO br.puc.rio.business.Downgrade:79 - Downgrade Build: build
2022-11-13 12:11:10,571 [main]  INFO br.puc.rio.business.Downgrade:79 - Downgrade Build: build
2022-11-13 12:11:10,596 [main] ERROR br.puc.rio.business.Downgrade:81 - Downgrade Build execution error:buildInformation
```

Figure 6.1: Execution Log

6.3.6 Checking synchronized Builds in the Database

To check the Builds applied to your Database, just check the contents of the BuildInformation table in your database. This table is created automatically on the first run of Upgrade Configuration.

An example can be seen in Figure 6.2.



Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	buildNumber	buildSequence	majorVersion	minorVersion	releaseVersion	status	steps	upgradeDate
0e52e3da-93e6-4461-be4b-6fa685a064b5	180122	0	14	0	0	COMPLETE	2	2022-10-28 19:35:45
905f7759-5187-47c3-823e-de13fb4f378c	180122	0	13	0	0	COMPLETE	4	2022-10-25 16:09:11
4188bb4f-fbd7-4f45-9452-515b056957aa	180122	0	12	0	0	COMPLETE	4	2022-10-25 16:03:24
4374684c-39e0-4284-baf5-1cb449b1d434	180122	0	11	0	0	COMPLETE	4	2022-10-25 10:44:18
ade29a3b-6417-437e-915b-76e94a2d01bf	180122	0	10	0	0	COMPLETE	4	2022-10-25 10:37:34
2dec2597-97e5-43c7-821d-1860cc111ebf	180122	0	9	0	0	COMPLETE	4	2022-10-25 07:39:54
43f412d7-9a29-48b2-8e22-b6bcdcc0589e	180122	0	8	0	0	COMPLETE	4	2022-10-25 07:01:12
18748cac-82f6-492c-b990-8fdfc5551d90	180122	0	7	0	0	COMPLETE	4	2022-10-25 07:01:11
fe2cdff2-df0f-46ff-8b80-f13ce2b6fced	180122	0	6	0	0	COMPLETE	4	2022-10-25 07:01:10
aa468c0b-a91e-40d4-b109-c1f76608b19c	180122	0	5	0	0	COMPLETE	4	2022-10-25 07:01:10
96bae611-5947-432f-b0fe-8729b6655d31	180122	0	4	0	0	COMPLETE	4	2022-10-25 07:01:09
ff41eec9-f8a1-4267-a997-2899c2e2d221	180122	0	3	0	0	COMPLETE	4	2022-10-25 07:01:08
ffc88f7d-a4a5-4244-b1e8-d99ef8c1a501	180122	0	2	0	0	COMPLETE	4	2022-10-25 07:01:07
128c24d0-b797-4c68-bce1-04c9fcf29b6d	180122	0	1	0	0	COMPLETE	4	2022-10-25 07:01:07
33d607f9-7c4c-44f1-9de1-76a999f82304	180122	0	0	0	0	COMPLETE	4	2022-10-25 07:01:06
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	

Figure 6.2: Sample of content in BuildInformation table

6.4 Detailing the structure of a Build

A build is composed of a list of Steps. As can be seen in the XML code 1.

6.4.1 Detailing the structure of a Step

A step is composed of the number attribute and an upgrade action and downgrade action. The number attribute informs the step sequence. The upgrade action executes scripts in order to evolve the database. The downgrade action executes scripts in order to regress the database. As can be seen in the XML code 1.

Note that the upgrade action is mandatory for every step, but the downgrade action is optional.

6.4.2 Types of Action

The types of actions available for Upgrade or Downgrade are just implementations of the Action interface.

This subsection presents the RunSQLAction, RunSQLFileAction implementations available for use. And an example of creating a custom action.

6.4.2.1 RunSQLAction action type

To use the RunSQLAction action type, set the class attribute of the upgrade, or downgrade tags with the value "br.puc.rio.model.RunSQLAction", and fill the sql tag, with the sql script to be executed. As can be seen in the XML code 5.

Code 5: RunSQLAction sample

```
<!-- Upgrade Action -->
<upgrade class="br.puc.rio.model.RunSQLAction">
    <sql>CREATE TABLE MY_FIRST_TABLE(id int(4) AUTO_INCREMENT,name
            varchar(30) NOT NULL,PRIMARY KEY (id));</sql>
</upgrade>
```

6.4.2.2 RunSQLFileAction action type

To use the RunSQLFileAction action type, set the class attribute of the upgrade, or downgrade tags with the value "br.puc.rio.model.RunSQLFileAction", and fill the path tag, with the path to the sql file that has the content to be executed. As can be seen in the XML code 6.

Code 6: RunSQLFileAction sample

```
<!-- Upgrade Action -->
<<upgrade class="br.puc.rio.model.RunSQLFileAction">
    <path>demo/upgrade-script.sql</path>
</upgrade>
```

6.4.2.3 Build example with custom action

To use a Custon action type, set the class attribute of the upgrade, or downgrade tags with complete name of your custom class. As can be seen in the XML code 7.

Your Custom Class just need to implements the interface Action, how the sample code 8. Also available on GitHub at https://github.com/rcabral/upgrade-configuration/blob/main/source/upgrade-configuration/src/main/java/br/puc/rio/sample/HelloCustomAction.java.

Code 7: CustomAction sample

```
<!-- Upgrade Action -->
<upgrade class="br.puc.rio.sample.HelloCustomAction"/>
```

Code 8: HelloCustomAction sample

```
package br.puc.rio.sample;

import javax.persistence.EntityManager;

import br.puc.rio.model.Action;

/**
 * A sample Class of a CustomAction implementation.
 */
public class HelloCustomAction implements Action {

  @Override
  public void execute(EntityManager entityManager) throws Exception
      {
    System.out.println("Hello Custom Action! Here you can put your
        arbitrary code.");
  }

}
```

6.5 XSD with upgrade-configuration.xml structure details

The XSD file with details of the upgrade-configuration.xml structure is available on GitHub at https://github.com/rcabral/upgrade-configuration/blob/main/source/upgrade-configuration/upgrade-configuration.xsd.

6.6 Sample Web Project

A Sample Web Project is available on GitHub at https://github.com/rcabral/upgrade-configuration/tree/main/source/sample-web-project, for free use and modification.

7
Bibliography

GAMMA, E. et al. **Design patterns: elements of reusable object-oriented software**. [S.l.]: Pearson Deutschland GmbH, 1995.

JACOCO. **Jacoco Maven Plug-in**. 2022. Disponível em: <https://www.eclemma.org/jacoco/trunk/doc/maven.html>.

JUNIT. **JUnit**. 2022. Disponível em: <https://junit.org/junit4/>.

MARTIN, R. C. Principles of ood. **Online]. Tersedia: butunclebob. com/ArticleS. UncleBob. PrinciplesOfOod.[Diakses pada Juli 2018]**, 1995.

MARTIN, R. C. **Clean code: a handbook of agile software craftsmanship**. [S.l.]: Pearson Education, 2009.

MOCKITO. **Mockito**. 2022. Disponível em: <https://site.mockito.org/>.

PMD. **PMD Source Code Analyzer**. 2022. Disponível em: <https://pmd.github.io/>.

SUREFIRE. **Maven Surefire Plugin**. 2022. Disponível em: <https://maven.apache.org/surefire/maven-surefire-plugin/>.