DESCRIBING LOG4UNIT LIBRARY & ADDRESSING ITS IMPLEMENTATION ISSUES

Jatin R. Ambasan *

Dhaval A. Parikh **

ABSTRACT

This paper describes the Log4Unit Library and addresses issues experienced during its implementation. Log4Unit provides the capability of logging test results along with testing java classes. It provides the capability to log messages from within the testcase methods. It supplies extension to JUnit's TestCase class (LoggedTestCase). It has inbuilt logging functionality, based on Log4J framework, to document initial test settings, test case execution and results. Attractive feature of LoggedTestCase class is that it redirects the log messages to console if Log4J library is not present in the classpath at runtime. This makes it easy for users who do not want to download Log4J and put it in their classpath. LoggedTestCase instantiates a Log4J logger instance and implements utility logging methods such as info, debug, etc. Developer just needs to call the method in order to log messages according to required log levels. GUI of Log4Unit uses a customized Swing-based test runner that shows logged statements and test summary information in a dialog box. This work was prompted due to lack of published results concerning the implementation of Log4Unit. This paper portrays our experiences and lists benefits & limitations of Log4Unit & several recommendations are outlined for future work on it.

Keywords: *Document test results, JUnit, Log4Unit, logging, Test protocols*.

* Computer Science Engineering Government Engineering College, Gujarat Technological University, Gujarat.

** Associate Professor, Computer Engineering Dept., Government Engineering College, Gujarat Technological University, Gujarat.

I. INTRODUCTION

Log4Unit is an OpenFuture Project [1]. As described by Wolfgang Reissenberger [2], Log4Unit is a JUnit extension combining JUnit [3] with Log4J [4] as shown in figure 1. The purpose behind building such an extension is the necessity to create test protocols for JUnit. JUnit is asymmetrical in the sense that it focuses on the documentation of test failures and errors. Successful execution of a test case is not further documented. To keep track of testing process and monitor the performance of test cases, successful execution of a test case should be further documented. So the basic intention behind building Log4Unit is the necessity to create test protocols in order to document the initial test settings, the test case execution and its results. Along with the testing framework a logging component is required to document the results of test case executions. Wolfgang Reissenberger has selected Log4J as the current de facto standard for extending the Testing framework of JUnit to provide the logging facility. The rest of the paper is organized as follows. In section 2, the paper explicates the background knowledge required to understand the Log4Unit structure. Section 3 demonstrates how to make use of it in a java program (for code and detailed steps please contact corresponding author of the paper). In Section 4, emphasis is given to the limitations & in section 5 various recommendations are outlined for future work on it. In the end section 6 illustrates the conclusions of this paper and section 7 consists of acknowledgments.

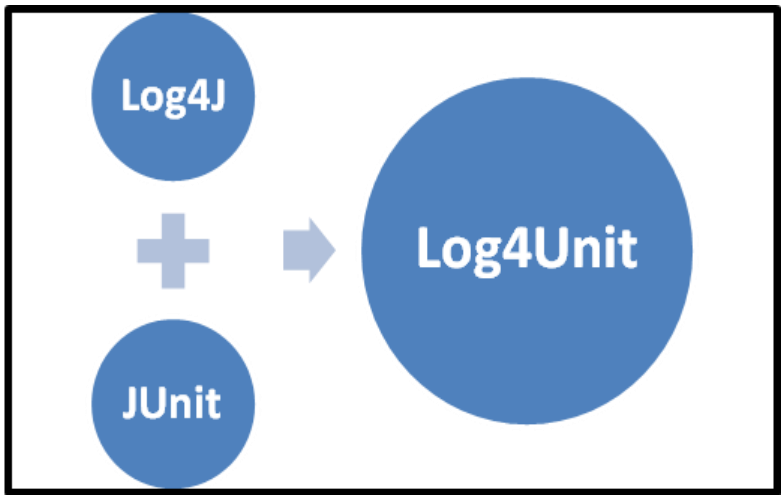

Figure 1: Log4Unit as a combination of Log4J and JUnit

II. BACKGROUNG AND RELATED WORK

According to J. B. Rainsberger, Log4Unit is a ready-made solution for logging messages from within the test cases [5]. It provides us with an extension of JUnit's TestCase class. This class is named as the LoggedTestCase class. This class facilitates its user with the Log4J based

logging functionality. The TestCase class of JUnit framework does not contain any methods for logging the results or any statements. Developers have to use special logging libraries like Log4J and instantiate its Logger object to log statements & results while unit testing. Hence, the subclass junit.log4j.LoggedTestCase is being introduced. This class provides various methods for logging as per required log levels.

Different log levels supported by junit.log4j.LoggedTestCase class are:

- debug( ) → org.apache.log4j.Category.debug( )
- error( ) →org.apache.log4j.Category.error( )
- info( ) → org.apache.log4j.Category.info( )
- warn( ) → org.apache.log4j.Category.warn( )
- fatal( ) → org.apache.log4j.Category.fatal( )

Above logging methods are provided in the LoggedTestCase class for various log levels identified by Log4J (figure 2). Note that the log method itself is not provided; instead each log level method is used directly. It can be said that the design pattern known as Adapter [6] is used to design this class. It uses a concept where Object Adapter relies on object composition as shown in Figure 3. Setup and Teardown methods are also provided.

In JUnit, preparation for testing is called setup. It's independent of the assertions, so the same scaffolding can apply to several individual tests. The setup is run before each test. When a test ends, the scaffolding might require some cleanup actions. In JUnit this is called teardown. Setup and teardown are called a test's fixture. [7].

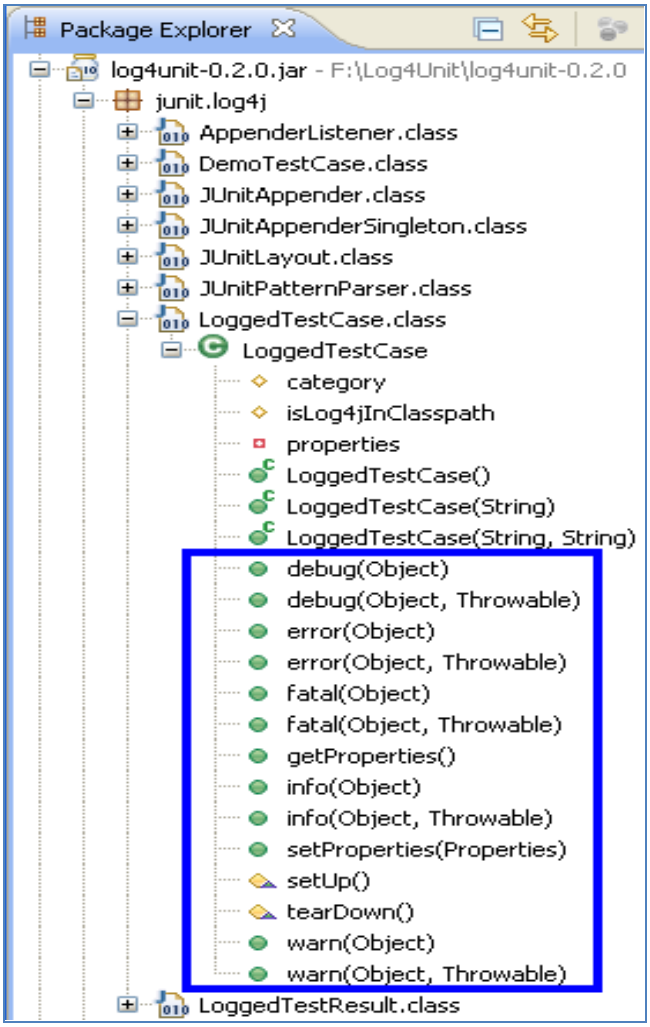

Figure 2: Logging Methods provided by the class LoggedTestCase

Figure 4 shows partial structure diagram of Log4Unit. It shows that the Object Adapter design pattern is implemented to design the LoggedTestCase class. Do note that the LoggedTestCase class is the heart of the Log4Unit, so the authors of this paper have focused on its partial class diagram in this text rather than showing full class diagram.

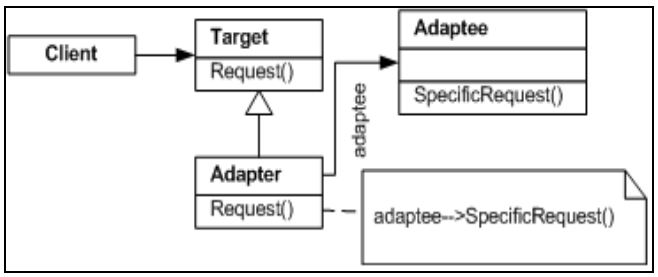

Figure 3: Object Adapter design pattern diagram from [6]

Drawing a full class diagram of the Log4Unit library in itself is a massive task. Thus only the partial class diagram is included, depicting the structure of required classes for current paper understanding.

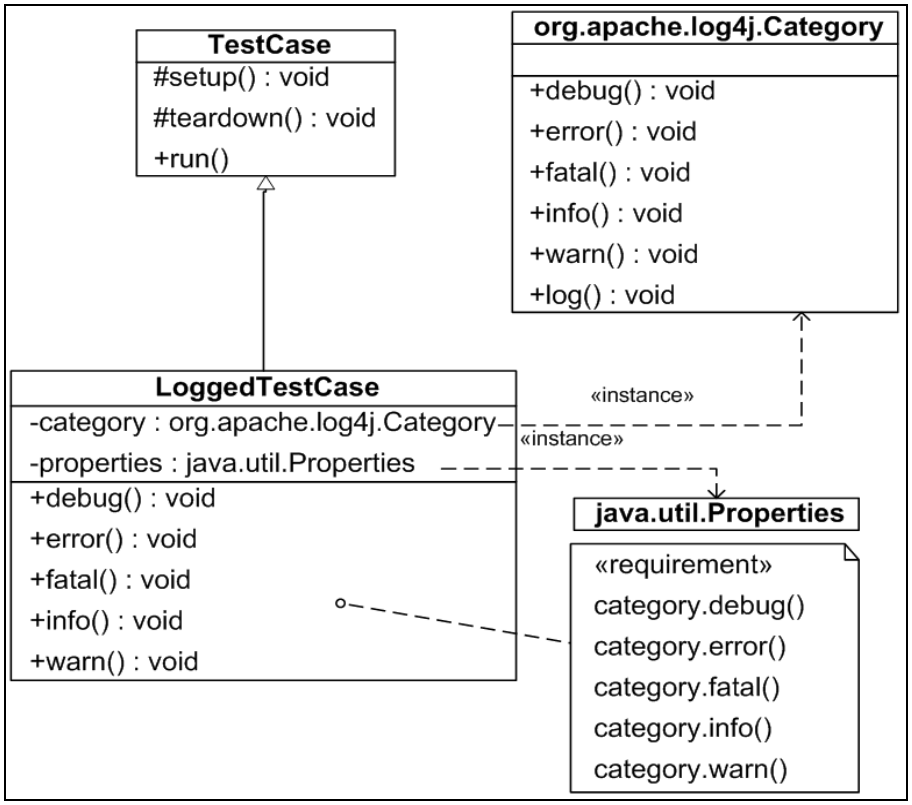

Figure 4: Log4Unit Structure Diagram (Partial). Implementation of the Object Adapter design pattern for designing the LoggedTestCase class

Following aspects of Log4Unit make it an attractive choice for developers who intend to use a testing framework having built-in logging capability:

The class junit.framework.TestCase does not contain any methods for logging. Hence, the subclass junit.log4j.LoggedTestCase is introduced providing methods for logging. In order to configure the logging library, place a log4j.properties file in your classpath. If Log4J library is not present on the classpath then the test cases by default log the messages to System.out object (i.e. the log messages are shown on the console).

By employing the Object Adapter Pattern, Log4Unit instantiates a Log4J logger instance and calls this Logger object's methods inside its own methods having same name as those logger methods. For example, to provide the developer to log messages having log level as "warn" LoggedTestCase implements a method with the following signature: public void warn(Object message)

To understand both the above points consider the following implementation (Table 1) of "warn" method in the LoggedTestCase class under the package junit.log4j:

Table 1: "warn()" method implementation in the LoggedTestCase class:

public void warn(Object message) {
if (isLog4jInClasspath)
{ this.category.warn(message); }
else
{ System.out.println("[warn]: " + message);
}
}

"isLog4jInClasspath" is an instance variable declared in the LoggedTestCase class. It is set to false if Log4J cannot be successfully instantiated. Log messages default to System.out if Log4J is not found in the classpath at runtime.

Another pleasant feature of Log4Unit is that it comes with a customized Swing-based test runner class (junit.logswingui.TestRunner) that shows log statements and test summary information in a dialog box. This dialog box pops up i.e. it becomes visible with the push of a button titled as "Protocol".

Not only the TestCase class but many other components have been modified in the library. For example, a new Log4J appender (JUnitAppender) is introduced to provide the coupling to the JUnit test runner. Since appenders are instantiated by the configurator of Log4J a singleton is needed to populate the logging events to the test runner (JUnitAppenderSingleton).

The org.apache.log4j.PatternLayout is extended and the new JUnitLayout class is designed to provide customized messages like the stack traces of Throwables in the appenders.

III. INTEGRATION OF LOG4UNIT IN A SAMPLE JAVA PROGRAM

Log4Unit can be used to integrate tests with the Log4J logger. Log4Unit library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License [8] as published by the Free Software Foundation. The latest version as of this writing is v0.2.0. Download the .zip or .gzip file and unpack it into a new directory. To use Log4Unit library in any projects two other dependencies need to be downloaded, namely JUnit [3] version 3.8.1 and Log4J [4] version 1.2.8.

To use Log4Unit follow the guidelines given below:

- Extend the TestCases from LoggedTestCase class (class name with package name): junit.log4j.LoggedTestCase.
- Write a Log4J configuration file, or place the directory containing Log4Unit's provided log4j.properties in the class path (probably the source directory where the Log4Unit is unzipped).
- Add log4j-1.2.8.jar and log4unit-0.2.0.jar to usual test class path.
- Class named junit.logswingui.TestRunner can be used as a GUI test runner to have access to the test summary in following manner:

  java junit.logswingui.TestRunner [-noloading] [TestCase]

If everything goes correct then a dialog box similar to the figure 5 may be seen. It represents the GUI feature of Log4Unit i.e. its customized Swing-based test runner with its test protocol feature. This dialog box comes with a new Protocol button.

On pressing the "Protocol" button a screen similar to following Figure 6 can be seen. It is the Test protocol screen which shows the test summary. The test summary can also be saved by pressing save button. This is the primary goal behind creating Log4Unit i.e. the creation of test protocols.

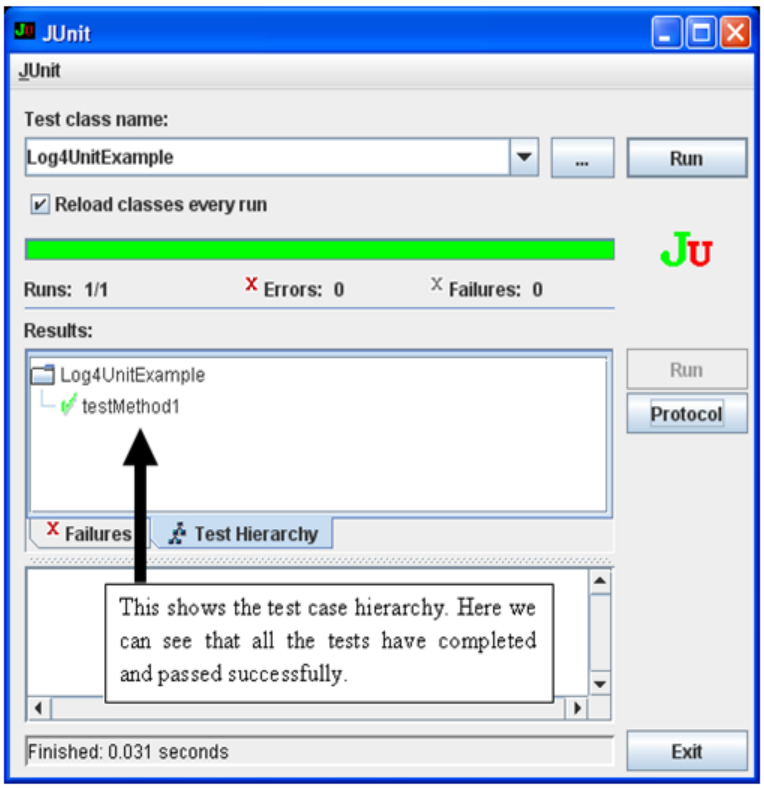

Figure 5: Running Log4UnitExample from command line using Log4Unit's customized Swing-based test runner

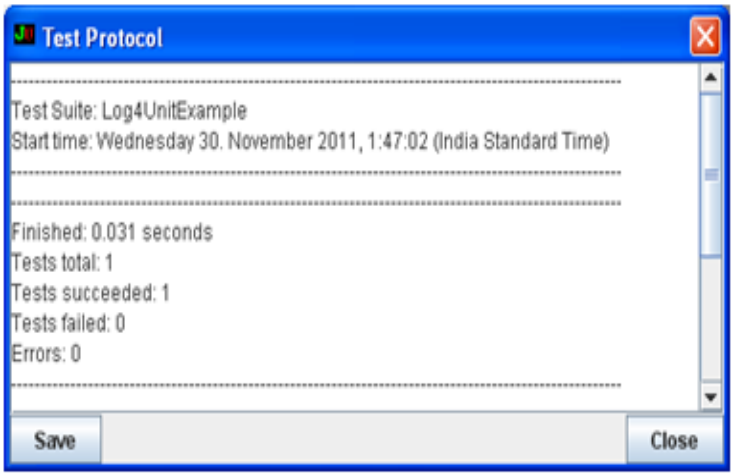

Figure 6: Test Protocol showing test summary

IV. LIMITATIONS OF LOG4UNIT

A. Log4Unit requires user TestCases to be extended from LoggedTestCase. Many users may feel that they are forced into extending a class which has a less importance in their software product. If users want to extend their base test class from other class to provide some other capability (along with the capability to Log the Test case results) then they cannot do so since they would have already extended the LoggedTestCase class. Multiple inheritance is not allowed in Java.

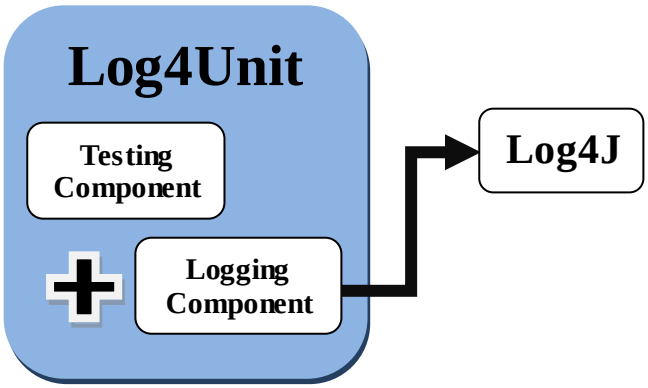

Figure 7: Log4Unit

B. Log4Unit only supports Log4J library for logging. Although Log4J is a very flexible and powerful logging framework, sometimes users may need to use other logging implementations like logback, java logging package (java.util.logging package) or commons logging as per the requirements.

C. Log4Unit was designed to work with JUnit version 3.8.1, and so isn't compatible with the new syntax of JUnit 4. Since in JUnit 4 programmers do not need to extend the TestCase class.

D. Log4Unit is tightly coupled with its underlying logging framework (the defacto standard – Log4J) as shown in figure 7.

V. RECOMMENDATIONS FOR FUTURE WORK ON LOG4UNIT

A. Log4Unit is an open source and small library, so developers can customize it to support the logging implementation of their choice i.e. If any programmer wants to use any other logging framework, for example take Jakarta Commons Logging [9], than they can change the Log4Unit source and create their own library providing customized logging backend. Programmers can abstract the logging implementation using a bridge such as Jakarta Commons Logging.

B. If developers plan to write thousands of tests, it is recommend that they extend a base TestCase of their own from LoggedTestCase and extend all your other tests from that base TestCase. This way if Log4Unit is removed or replaced, they only have to change the base TestCase and not each and every TestCase.

C. Another characteristic of Log4Unit that should be looked into is that, it becomes tightly coupled with the underlying logging framework (the default Log4J or the customized one, whichever the user implements). To overcome this inflexible situation programmers can use a logging façade like SLF4J (Simple Logging Façade for Java) [10]. SLF4J will serve as a simple facade or abstraction for various logging frameworks, e.g. java.util.logging, Log4J or Logback, allowing the end user to plug in the desired logging framework at deployment time. This can provide a loose coupling of Log4Unit with the underlying logging framework.

D. Scarcity of documentation on Log4Unit makes it difficult for programmers and research students to understand its structure and take a look at the Software engineering concepts used in its structure design. So a lot of work needs to be done for creating the class diagrams and other documentation on this topic so that it can be used for further research.

VI. CONCLUSION

This paper presented the Log4Unit library whose goal is to create test protocols for JUnit. It is observed that this library is asymmetrical in the sense that it focuses on the documentation of test failures and errors. It is also tightly coupled to the underlying logging framework. This library is pretty much useful in small projects that need to startup immediately with a Testing mechanism as well as a logging framework. It reduces the efforts of programmer for maintaining a separate logging framework. The library has already been used by some developers and has proved useful to small scale projects. Since the Library is open source it

can also be customized to support the business requirements of individual organizations. Programmers can abstract its logging implementation by using a bridge or a façade like SLF4J so as to decouple it from the logging library, thus making it independent of underlying logging framework.

VII. ACKNOWLEDGMENT

Authors of this paper appreciate the original author of the Log4Unit library, Wolfgang Reissenberger, for providing his invaluable time to review the ideas and also answer the queries promptly. Although this library is open source, this research work began after obtaining prior consent of its original author. Authors of this paper would also thank their colleagues, friends, classmates, teachers and other guides for providing their immense support and helpful comments to improve this paper.

VIII. REFERENCES

1. OpenFuture Project. http://www.openfuture.de
2. Log4Unit. Available at: http://www.openfuture.de/Log4Unit/
3. JUnit. Available at: www.junit.org
4. Log4J. Available at: http://logging.apache.org/log4j
5. Rainsberger, J. B. *JUnit Recipes – Practical Methods for Programmer Testing*. Manning Publications Co, 2005.
6. Gamma, Erich; Richard Helm; Ralph Johnson; and John Vlissides. *Design Patterns*. Addison-Wesley, 1995.
7. Panagiotis, Louridas. *JUnit: Unit Testing and Coding in Tandem*. IEEE Software. July / August 2005.
8. GNU Lesser General Public License:
9. http://www.gnu.org/copyleft/lesser.html
10. Jakarta Commons Logging. Available at:
11. http://jakarta.apache.org/commons/logging.html
12. Simple Logging Facade for Java (SLF4J). Available at: http://www.slf4j.org/