Assignment #4: Coverage

Andrew Bwogi Anders Eriksson Mona Lindgren Adrian Lökk
Juha-Pekka Puska

February 28, 2018

Contents

0 Project choice

For this assignment, the group chose the project *Karate*, a framework for testing web services written in Java. The project is free and open-source software and can be found on GitHub (`https://github.com/intuit/karate`).

1 Complexity Measurement

1.1 10 Most Complex Methods

The group used the tool *Lizard* (`https://github.com/terryyin/lizard`) to analyze the cyclomatic complexity of the `karate-core` module of the project. The 10 most complex methods and their NLOC and CCN can be seen in table 2.

Table 1: 10 Most Complex Methods

Class::Method	**NLOC**	**CCN**
`Script::matchNestedObject`	205	70
`Script:matchStringOrPattern`	224	64
`Script::matchJsonOrObject`	106	38
`HttpClient<T>::buildRequestInternal`	91	30
`Script::evalXmlEmbeddedExpression`	84	28
`ScriptContext::configure`	100	24
`Script::evalKarateExpression`	80	22
`FeatureServerHandler::writeResponse`	78	18
`Script::matchPrimitive`	46	18
`Script::call`	53	18

Out of these methods, the cyclomatic complexity of the following methods was manually calculated by two of the group members:

- `FeatureServerHandler::writeResponse`
- `HttpClient<T>::buildRequestInternal`
- `ScriptContext::configure`
- `Script::evalKarateExpression`
- `Script::call`

The results were identical with the ones provided by Lizard. For the Lizard summary output, see appendix A.

1.2 Description of Methods

Script::matchNestedObject

This function is used to assert whether a given object or pattern matches a given `ScriptContext`. The function handles nested values in JSON or XML. The definition of "matches" is given in the

parameter `stringMatchType`. This is an essential part of the framework since it allows for easy testing of JSON or XML, the two most common standards for sending data to and from web APIs. The function returns an `AssertionResult` which also contains the reason for the matching failed or passed. The function contains a lot if-statements, mainly to ensure the correct output (the reason it passed/failed).

Due tue a massive amount of JSON, XML, and String parsing, the method is absolutely huge at 205 NLOC and insanely complex at 70 CCN.

Script::matchStringOrPattern

This method does the same thing as the above mentioned `Script::matchNestedObject` except that it takes a string or pattern instead of a general object and it does not deal with nested values.

Due to a ton of string parsing, the function is not only incredibly complex with a CCN of 64. It is also extremely long at 224 NLOC.

Script::matchJsonOrObject

This function could be described as an interface to the other ”match” methods of the `Script` class. It contains large switch statements which leads to a very high CCN. These switch statements are used to determine the type of the object sent to the method as a parameter which is then handled accordingly.

HttpClient<T >::buildRequestInternal

This function builds a http response from a HttpRequestBuilderObject. Again, the complexity if caused by if-statements that test certain parameters of the http reponse object. Most of the if-statements are single-line and very easy to understand. At the end though, there are nested conditionals four statements deep, which is usually a sign that the function should be refactored.

The Lizard tool gives this function a CCM number of 30. This time the function has 21 conditionals and 8 for-statements that mean the CCM given by the formula is 30.

Script::evalXmlEmbeddedExpression

This function evaluates embedded expressions in XML (expressions that are evaluated at run time). Due to a lot of parsing of XML and the embedded expressions, there is a lot of if-statements leading to the high CCN.

ScriptContext::configure

This function is passed a key/value-pair and it is used to configure options for the script. The complexity is caused by if-statements that test different options for the key.

There are 23 conditional statements in the function, and therefore the CCN would be the same as the one give nby the tool. However, the function also has throw and return-statements which do not seem to affect the CCN

Script::evalKarateExpression

This function takes as input a string containing a karate expression, i.e., a high level statement that the user can write to define the api test. These expressions can contain JSON, JS functions, set variables etc. The function parses these expressions and calls appropriate functions to handle the input. Once again, string parsing causes a lot of if-statements that result in a high cyclomatic complexity.

Manual counting gives us 21 conditional statements which means the CCN is the same as the one given by the tool.

FeatureServerHandler::writeResponse

This function reads an HttpRequest object and generates an HttpResponse. It acts as a wrapper around the regular netty HttpResponse object. The complexity comes mostly from short if/else statements that test certain fields in the request object. The function itself is not particularily complicated to understand if one is familiar with http networking terms.

The lizard tool gives this function a CCM number of 18. Counting manually we see that the function has 17 binary decision predicates and therefore according to the formula, the CCN is 18. This is the same as the output given by the tool.

Script::matchPrimitive

This function does the same thing as the earlier mentioned `Script::matchStringPattern` except that it matches primitives instead of strings or patterns.

The function contains a total of 22 conditional statements which means the CCN is the same as the one given by Lizard.

Script::call

The function is used to call javascript functions contained in the Karate-expressions. The function contains switch statement with two branches, each of which contains two nested switch statements themselves. The high complexity of the function comes form the number of switch cases. However, most of the cases are actually empty, and the function is quite short.

Manual counting gives us 17 case-statements (not including the default-cases) which means that for this function we get the same CCN as outputted by the tool.

1.3 Documentation of the Methods

There is barely any documentation of the methods for developer and as such, the only way to get information about the possible outcomes of the functions is to read and understand the code. Considering there are several methods longer than 100 lines, this is a very time consuming and tiring process.

1.4 Regarding Exceptions

Several of the methods include exceptions and try/catch-statements. During the manual counting of the CCN, it seemed that Lizard completely disregarded anything within try or catch blocks. If we consider try/catch-blocks the same as if/else-blocks, the CCN of some of these methods

would increase dramatically. For example, `Script::evalXmlEmbeddedExpressions` contains two try/catch-statements with a total of 13 if/else-statements. The group is unsure whether counting these blocks makes any sense since there is no way to know how far down a try-block the program will progress. At the same time, the same argument could be made regarding if-statements and switch-statements.

2 Coverage Measurement & Improvement

2.1 Tools

We used *openClover* (`http://openclover.org`) to check the coverage of the testing suite, some using the IntelliJ plugin and others running it straight from the command line. The plugin provides an overlay on the code after running a test suite which will highlight lines covered in green and lines not covered in red, making it a very intuitive tool. The documentation was adequate, and the few problems we did encounter were not related to the plugin itself.

2.2 DIY

The following two commands will output a patch showing the difference between the original code and after adding our ad-hoc coverage tool. We used two slightly different implementations, hence the additional git branches.

Adrian: `git diff -patch master..coverage-2-adrian`
Andrew: `git diff -patch master..coverage1`

2.2.1 Methods Measured

- `Script::matchPrimitive`
- `Script::setValueByPath`
- `Script::matchJsonOrObject`
- `JsonUtils::recursePretty`
- `Script::call`
- `ScriptValue::getAsString`
- `JsonUtils::setValueByPath`
- `StepDefs::toMatchType`
- `ScriptContext::configure`
- `ScriptContext::ScriptContext`

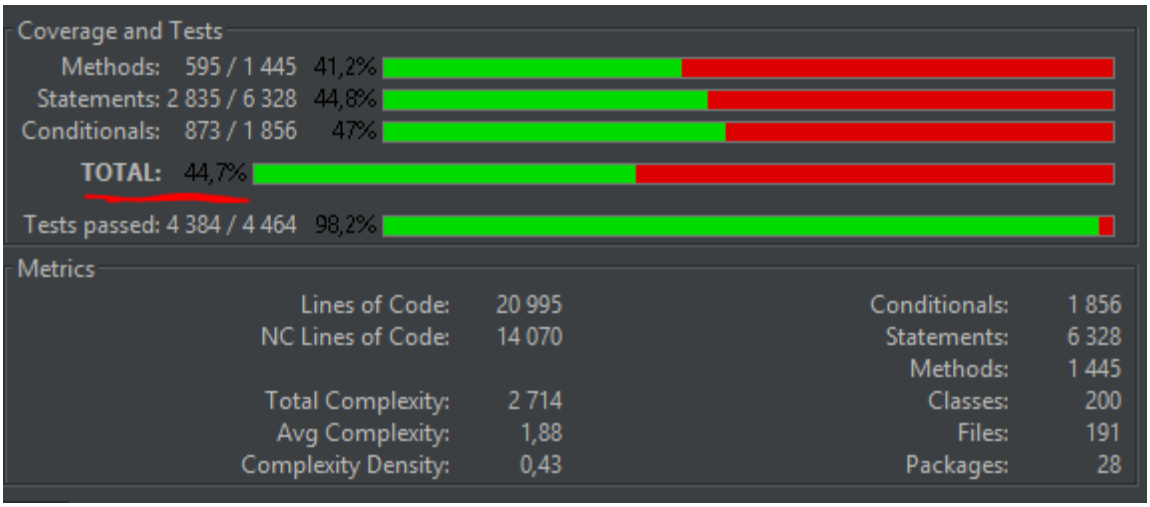
2.2.2 Method

A data structure with fields for class name, function name and the number of branches was created. To use the tool, a CoverageStructure object is created at the beginning of the function, then for each branch, call the `addBranch()` method with a unique id. This will add it to an array and then write that array to a file in the system's temporary directory with the template `/path/to/tmp/coverage-className_functionName`. This file will be read from to instantiate the array anytime a new `CoverageStructure` is created with the same class- and function name, thus saving the reached branches between runs.

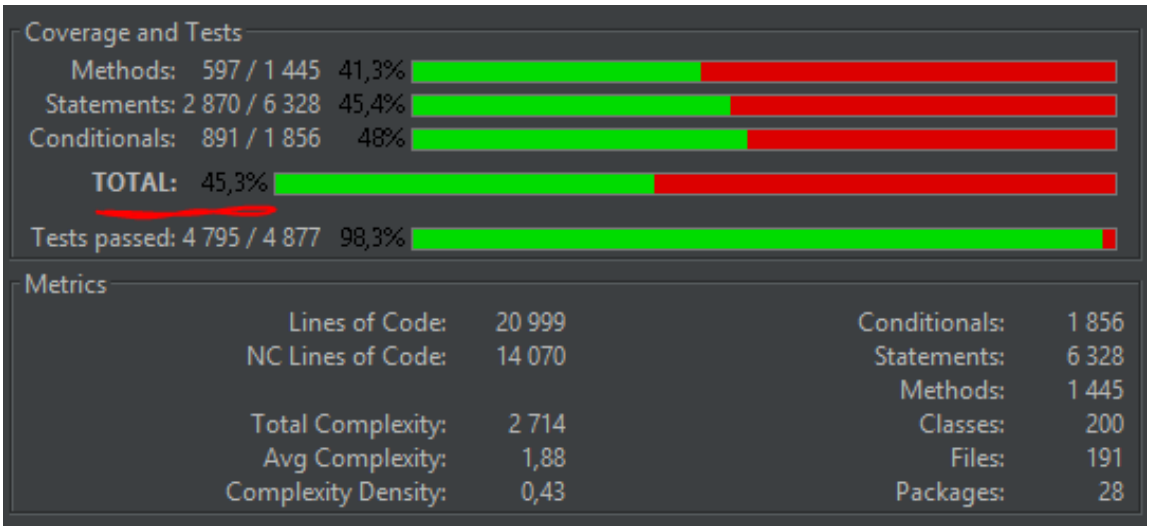
For most of the functions the DIY solution matches the coverage reported by openClover, however some discrepancies can be observed. The handling of switch/case blocks in Java seem to be more sophisticated (as expected) with openClover, being able to make a distinction between a reached case and a fall-through case, something which our rudimentary solution can not.

2.2.3 Evaluation

Report of old coverage



Report of new coverage



Test Cases Added

`git diff -patch master..develop-test`

3 Refactoring

3.1 Script::matchNestedObject

Method Signature

```
private static AssertionResult matchNestedObject(char delimiter,
    String path, MatchType matchtype, Object actRoot, Object
    actParent, Object actObject, Object expObject, scriptContext
    context)
```

Description

The method has a long if/else if/else-statement that checks what the type of the expected object is. Depending on its type, further checks are made inside the if-statements. The secondary checks are also carried out by if-statements, which result in deep nested conditional statements and consequently a high cyclomatic complexity. This method can be refactored to contain only the first type check and calls to private methods that perform the secondary checks.

Suggested Refactoring

```
public static AssertionResult matchNestedObject(char delimiter,
    String path, MatchType matchtype, Object actRoot, Object
```

```
    actParent, Object actObject, Object expObject, scriptContext
    context) {
        if( expObject == null )
                return matchNull(...)
        else if ( expObject instanceOf Map )
                return matchNestedMap( ... )
        ... etc ...

        else if (isPrimitive(expObject.getClass()))
                return matchPrimitive( ... )
                else  // this should never happen
                throw new RuntimeException( ... );
}
```

Private methods for the secondary type check must be added. The code in them can basically be copied from the original method:

```
private static AssertionResult matchNestedString(char delimiter,
    String path, MatchType matchtype, Object actRoot, Object
    actParent, Object actObject, Object expObject, scriptContext
    context){
        ScriptValue actValue = new ScriptValue(actObject)
        return matchStringOrPattern( ... );
}
private static AssertionResult matchNestedMap(...) {...}
        ... etc ...
```

3.2 Script::matchStringOrPattern

Method Signature

```
public static AssertionResult matchStringOrPattern(char delimiter,
    String path, MatchType stringMatchType, Object actRoot, Object
    actParent, ScriptValue actValue, String expected, ScriptContext
    context)
```

Description

This method has code duplication. It shares almost identical code with `matchNestedObject` in its first if-statement. A general method for checking null-values can be used in both `matchNestedObject` and `matchStringOrPattern`.

Suggested Refactoring

```
public static AssertionResult matchStringOrPattern(char delimiter,
    String path, MatchType stringMatchType, Object actRoot, Object
    actParent, ScriptValue actValue, String expected, ScriptContext
    context) {
        if ( expected == null )
```

```
                return matchNull( ... )
        else if ( isMacro(expected) )
                matchMacro( ... )
        else if ( actValue.isStringOrStream() )
                matchStringOrStream( ... )
        else {
                ... etc ...
        }
        if (stringMatchType == MatchType.NOT_EQUALS) {
                return matchFailed( ... )
        }
        return AssertionResult.PASS
}
```

3.3 Script::matchJsonOrObject

Method Signature

```
public static AssertionResult matchJsonOrObject(MatchType matchType,
    ScriptValue actual, String path, String expression,
   ScriptContext context)
```

Description

The behavior of this method was very hard to analyze since it contain several long switch statements with fall-through blocks. The fall-through blocks are mixed with cases that breaks the enclosing switch-statement, but also case-statements that returns from the method. One way to refactor this method would be to:

- Change the switch statements to if/else if/else-statements.
- Put breaking or returning cases in separate else-if-statements.
- get rid of the fall-through cases and handle all of them in the else-statement.
- Identify blocks of code that is guaranteed to return and substitute them with a call to a private method.

Suggested Refactoring

```
public static AssertionResult matchJsonOrObject( ... ) {
        Type t = actual.getType()
        if( t == Type.LIST )
                actualDoc = actual.getAsJsonDocument()
        else if ( t == Type.XML )
                actualDoc = XmlUtils.toJsonDoc( ... )
        else if ( t == Type.INPUT_STREAM )
                return matchInputStream( ... )
        else if ( t == Type.PRIMITIVE )
```

```
                return matchPrimitive( ... )
        else if ( t == Type.NULL )
                return matchJsonOrObjectNull( ... )
        else
                throw new RunTimeException( ... )
```

The two remaining switch-cases can be refactored in about the same way.

3.4 Script::matchPrimitive

Method Signature

```
private static Assertionresult matchPrimitive(MatchType matchType,
    String path, Object actObject, Object expObject)
```

Description

There is code-duplication in the beginning of `matchPrimitive`. If it is important that only one of `actObject` and `expObject` is null, then XOR can be used instead of the if/else if statement.

Suggested Refactoring

```
matchPrimitive( MatchType matchType, String path, Object actObject,
    Object expObject ){
        if( (actObject == null) ^ (expObject == null) ){
                return matchPrimitiveNull(matchType, path, actObject
                    , expObject)
        }
}

private static AssertionResult matchPrimitiveNull( MatchType
    matchType, String path, Object actObject, Object expObject ) {
        if ( matchType == MatchType.NOT_EQUALS )
                return AssertionResult.PASS
        else
                return matchFailed( ... )
}
```

3.5 HttpClient::buildRequestInternal

Method Signature

```
private T buildRequestInternal(HttpRequestBuilder request,
    ScriptContext context)
```

Description

The code under the `if(methodRequiresBody)`-statement can be replaced with a call to a separate method.

Suggested Refactoring

```
private T buildRequestInternal(HttpRequestBuilder request,
    ScriptContext context) {
...
        if(methodRequiredBody) {
                return buildMethodBody(request, context);
        }
}

private T buildMethodBody(HttpRequestBuilder request, ScriptContext
    context) {
        String mediaType = request.getContentType()
        ... etc ...
        return getEntityInternal(body, mediaType)
}
```

4 Refactoring

url : https://github.com/andrewbwogi/karate-DIY-coverage.git in the branch *refactor*.
Script::matchNestedObject is refactored so that cnn is decreased from 70 to 9.
HttpClient<T>::buildRequestInternal is refactored so that cnn is decreased from 30 to 19(exactly 35%)

Table 2: 10 Most Complex Methods after refactoring

Class::Method	NLOC	CCN
Script::matchStringOrPattern	224	64
Script::matchJsonOrObject	106	38
Script::matchNestedObjectMap	86	29
Script::evalXmlEmbeddedExpressions	84	28
Script::matchNestedObjectList	75	25
ScriptContext::configure	100	24
Script::evalKarateExpression	80	22
HttpClient<T>::buildRequestInternal	53	19
Script::matchPrimitive	46	18
Script::call	53	18

5 Time & Effort

For each team member, the following amount of time was spent doing:

Plenary Discussions/Meetings

In our first meeting we all spent **3 hours** deciding on a project and splitting the work.

Discussions Within Parts of the Group

This is very hard for the group to know since it differs so much for everyone. We approximate that each member spent at least **an hour** in discussions.

Reading Documentation

Approximately **1 hour** each.

Installation & Configuration

Each member spent about **2 hours** building and configuring the project. This includes installation of Lizard and coverage measurement tools, as well as understanding how these work.

Analyzing Code

Each member spent approximately **7 hours** reading and understanding code. This includes analyzing output. The lack of documentation led to this taking a proportionally very long time.

Writing Documentation

Mona, Juha-Pekka, and Anders spent around **6 hours** each writing the project report. **Andrew and Adrian** spent **two hours** adding to the project report.

Writing Code

Andrew & Adrian spent **6 hours** each creating the manual branch coverage. Additional **two hours** were spent by every group member writing tests to improve test coverage. This includes finding suitable uncovered blocks of code and understanding the methods containing them. Mona spent 2 hours to refactor two of the most complex methods.

Running Code

The tests were quickly executed, only a negligible amount of time was spent on running the code.

6 Overall Experience

Besides learning to calculate cyclomatic complexity and creating our own measurement tool, the process of dissecting the codebase of a large, open-source project has been a rewarding experience. Although difficult at first, the test writing went relatively smoothly considering the magnitude of the code.

Branch coverage turned out to be a tedious work. The principle can be understood in five minutes but to apply it in practice is a time consuming enterprise. Writing tests for our project also turned out to be tedious as we were dealing with a web service testing framework that demanded a lot of parsing. On the positive side, the detailed inspection of the flow of a program can serve as a reminder of the value of refactoring.

Appendices

A Lizard

The Lizard tool was run on the `karate-core` module of Karate with the following command:

```
lizard -s cyclomatic_complexity
```

Adding `-s cyclomatic_complexity` sorts the output according to cyclomatic complexity.

A.1 Summary Output

```
================================================
  NLOC    CCN   token  PARAM  length  location
------------------------------------------------
205      70   1509      8     211 Script::matchNestedObject@1166-1376
   @./src/main/java/com/intuit/karate/Script.java
224      64   1588      8     231 Script::matchStringOrPattern@710
   -940@./src/main/java/com/intuit/karate/Script.java
106      38    741      5     108 Script::matchJsonOrObject@998-1105@
   ./src/main/java/com/intuit/karate/Script.java
91       30    681      2      93 HttpClient<T>::
   buildRequestInternal@113-205@./src/main/java/com/intuit/karate/
   http/HttpClient.java
84       28    663      3      84 Script::
   evalXmlEmbeddedExpressions@430-513@./src/main/java/com/intuit/
   karate/Script.java
100      24    791      2     104 ScriptContext::configure@174-277@./
   src/main/java/com/intuit/karate/ScriptContext.java
80       22    580      4      86 Script::evalKarateExpression@201-286
   @./src/main/java/com/intuit/karate/Script.java
46       18    404      4      49 Script::matchPrimitive@1400-1448@./
   src/main/java/com/intuit/karate/Script.java
53       18    313      4      54 Script::call@1531-1584@./src/main/
   java/com/intuit/karate/Script.java
63       17    401      4      64 JsonUtils::setValueByPath@240-303@./
   src/main/java/com/intuit/karate/JsonUtils.java
58       16    452      6      58 Script::setValueByPath@1472-1529@./
   src/main/java/com/intuit/karate/Script.java
```