Bahir Dar University Institute of Technology

Requirements Engineering Individual Assignment

JAVA REQUIREMENTS TRACER

(tool)

Documented by: Adonias Haile 1200818

Submitted to : Tadel M.

Contents

1. background/History:

JavaRequirementsTracer was developed by Ronald Koster while working at several software development projects by Logica from 2006 to 2009. Logica donated JavaRequirementsTracer to the open source community in September 2009. Part of the idea of tracing requirements in this way came from Wen Hu and Frank Reerink.

2. Main features and capabilities of the tool:

The list below is the main features of the tool:

- It is needed to quantify the amount of requirements we implemented for an application.
- It quantifies or improves the amount or quality of requirements covered in our software tests.
- It is needed to produce a traceability matrix from requirements to the code.
- It is needed to produce a traceability matrix from the code to the requirements.
- It simply annotates your code with requirements labels.
- It generates a report on your project using the annotations showing all of the info required.
- It can be run through:
    1. A Maven plugin (recommended)
    2. As a standalone Java program
    3. Or through a Unit Test
- Generally, it is the tool for monitoring traceability between requirements and Java code.

3. Limitations of the tool: Plugin searches for annotations basing on classes, not java files. It means, that it uses classes which

have annotations compiled in them. It means, that project which requirements are tracked has to have at least jrt-annotations included/on its classpath. Moreover, it is the same project which has annotations (tracked) and which builds

report (tracking). As it was discussed, it is not very good to include tracking artifacts within tracked code.

4. Steps on how to operate practically basic functionalities mentioned as main features with the tool.

you should follow the following steps to get the correct functionality of the tool:

1. Annotate your code:

a. See JavaDoc of javarequirementstracer.Requirements.

b. Example: See the main and test sources in the jrt-reporter and jrt-system-test subprojects.

2. Creating a report:

a. Create the traceability params and labels files. For an example see

*traceability_params.properties and *traceability_labels.properties in src/main/config and src/test/config of jrt-reporter subproject.

NB. In case of a multi-module project you must create separate params and labels files per module (aka. subproject, subsystem).

b1. Recommended: use maven plugin (goal trace, see below "When Using Maven").

b2.Alternative 1:Run "java javarequirementstracer.JavaRequirementsTracer" (see below"When Not Using Maven").See,also:Run:"java javarequirementstracer.JavaRequirementsTracer --help".

3. Multi-module project: creating overview reports.To create an overview report you must run the ReportAggregator after all module traceability reports have been generated. It aggregates those reports into an overview. You will need to create separate overviews for both main and system test code reports.Recommended approach:

a. Create the traceability_overview_params_main.properties file for the main code overview report and put it in your module last in your dependency chain.

Typically this is your system test code project. It is a best practice to put your system test code in such a separate system test module. Example project structure:

+-MyProject // packaging pom

+---Module1 // packaging jar

+---module2 // packaging jar, depends on Subsystem1

+---System-Test // packaging jar, depends on Subsystem1 and Subsystem2,contains

// system test code only.

For an example see src/test/config/traceability_overview_params_main.properties in

sources of the jrt-system-test subproject.

b. In case you have system test code put your it in your system test module.

Create the traceability_overview_params_system_test.properties file for the system test

code overview report and put it in src/test/config of your system test module.

see src/test/config/traceability_overview_params_system_test.properties I sources of the jrt-system-test subproject.

c1. Recommended: use maven plugin (goal aggreggate, see below "When Using Maven").

c2. Alternative: Run "java javarequirementstracer.ReportAggregator" (see below "When Not Using Maven").

See also: Run "java javarequirementstracer.ReportAggregator --help".

When Using Maven:

1. JavaRequirementsTracer should be in the central Maven repository.

If not add the following to your pom.xml file(s) or your settings.xml:

```
<repositories>
    <repository>
        <id>javarequirementstracer-repo</id>
        <url>http://reqtracer.sourceforge.net/m2-repo</url>
    </repository>
</repositories>
```

See:http://maven.apache.org/guides/introduction/introduction-to-repositories.html

2. Add the following dependency to your pom.xml file(s):

```
<dependencies>
    <dependency>
      <groupId>net.sourceforge.reqtracer</groupId>
      <artifactId>jrt-annotations</artifactId>
      <version>a.b.c</version>
    </dependency>
```

</dependencies>

3. Add the following plugin to your root pom.xml file (aka. superpom):

```
      <build>
            <plugins>
                  <plugin>

<groupId>net.sourceforge.reqtracer.mojo</groupId>
                        <artifactId>jrt-maven-plugin</artifactId>

                              <version>a.b.c</version>

                           <executions>

                                       <execution>
                                        <goals>

      <goal>trace</goal>

      <goal>aggregate</goal>
                                        </goals>
                                 </execution>
                        </executions>
                        <configuration>
                              <!-- Optional params for
goal trace.
                                    NB. Build number is
optional but recommended, eg. from

org.codehaus.mojo.buildnumber-maven-plugin,
                                          or through mvn's -D
option.
                              -->

      <buildNumber>${build.number}</buildNumber>

      <traceParamsFilenames>${jrt.trace.params.files}</t
raceParamsFilenames>
                              -->
```

```
                              <!-- Optional params for
goal aggregate. (only for multi-module projects)

     <aggregateParamsFilenames>...</aggregateParamsFile
names>
                              -->
                         </configuration>
                    </plugin>
               </plugins>
     </build>
```

NB.1.

For a single module project you usually omit the <traceParamsFilenames> element and use the default file locations (see "java javarequirementstracer.JavaRequirementsTracer --help").

For a multi-module project you usually define the property ${jrt.trace.params.files} in your root pom as follows:

```
                    <properties>

     <jrt.trace.params.files>src/main/config/traceabili
ty_params.properties</jrt.trace.params.files>
                    </properties>
```

And in your system-test module pom as follows:

```
                    <jrt.trace.params.files>

     src/test/config/module1_traceability_params.proper
ties,

     src/test/config/module2_traceability_params.proper
ties
                    </jrt.trace.params.files>
```

NB.2.

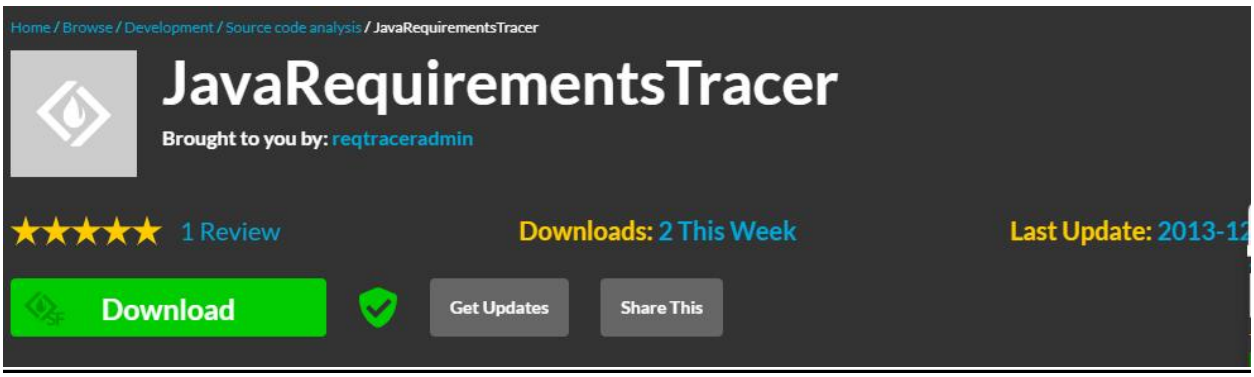
```
          For a multi-module project you usually omit
the <aggregateParamsFilenames> element and use the
          default file locations (see "java
javarequirementstracer.ReportAggregator --help")
          unless want to specify alternate file
locations. Example:

                <properties>
                     <aggregateParamsFilenames>

     ${overview_params_dir}/traceability_overview_param
s_main.properties,

     ${overview_params_dir}/traceability_overview_param
s_system_test.properties
                     </aggregateParamsFilenames>
                </properties>

But there are some cases when you must not use the
Maven.the following are cases you should not use maven:

 1. Download jrt-annotations-a.b.c.jar and put it in
your application compile and  runtime classpath. In
case of a multi-module project do this for each
subproject.
2. Download jrt-reporter-a.b.c-jar-with-
dependencies.jar and put it in your application test
classpath.next run:
"javajavarequirementstracer.JavaRequirementsTracer".
```

Incase of multi-module project :

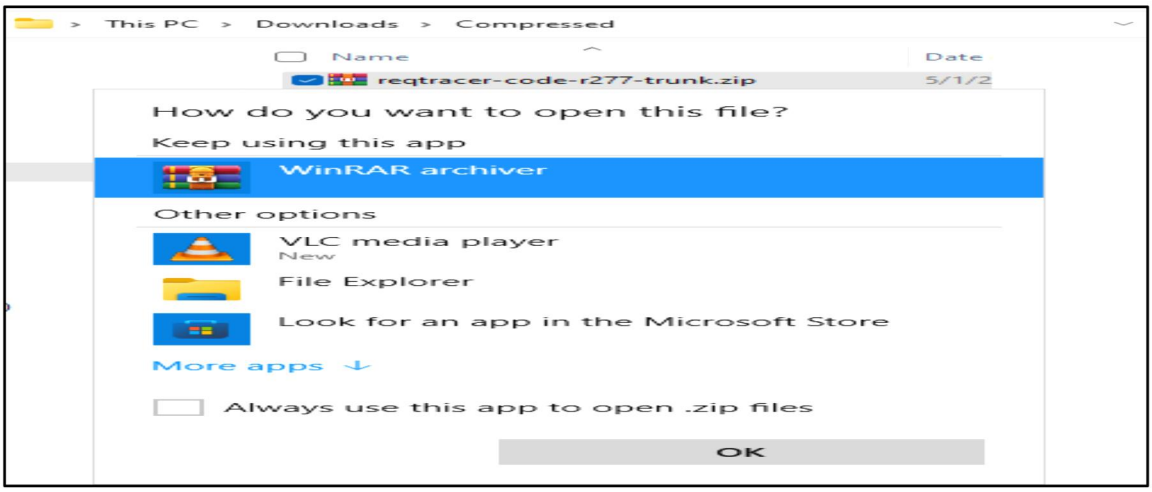
```
 a.do this for each subproject
 b .run “java requirementstracer.reportaggregator” for
both main and system testcode.
```

5. Sample screenshots from the practical implementation of the tool.

Download the File



After you download the file you have to extract it before start using it. You can use WinRAR archiver as I use below:



then follow the how? listed in the above html code to do the required tasks

.**Sample screenshot:**

Traceabilities for Reporter Main Code

Generated with JavaRequirementsTracer 1.7.0_258.

Summary

Timestamp: Mon Nov 08 11:02:19 CET 2010
Build Number: 258

CodeCoverage	100.00%		= traceableTypeCount/allTypesCount = 4/4
RequirementsCoverage	100.00%		= (foundLabelCount - unknownLabelCount)/requiredLabelCount = 4/4

Checksums:
requiredLabelCount = 4 =? missingLabelCount + foundLabelCount - unknownLabelCount = 0 + 4 - 0 = 4 ...OK
allTypesCount = 4 =? traceableTypeCount + untraceableTypeCount = 4 + 0 = 4 ...OK

Completeness estimates:

1. Best choice: Completeness = SytemTestCodeRequirementsCoverage * PercentageSuccessfulSystemTests
2. Next best choice (no System Test code): Completeness = MainCodeRequirementsCoverage * PercentageSuccessfulManualSystemTests

Settings

rootPackageName: javarequirementstracer
includePackageNames: [javarequirementstracer]
excludePackageNames: [javarequirementstracer.exclude, javarequirementstracer.mojo]
excludeTypeNames: [javarequirementstracer.Dummy1, javarequirementstracer.Dummy1Impl, javarequirementstracer.Dummy1ImplExt, javarequirementstracer.Dummy2, javarequirementstracer.Dummy3, javarequirementstracer.Dummy3Sub, javarequirementstracer.Dummy3SubImpl, javarequirementstracer.Dummy4, javarequirementstracer.Dummy5, javarequirementstracer.Dummy6, javarequirementstracer.DummyExcl, javarequirementstracer.IDummyExcl]
includeTestCode: false

Missing, Unknown or Untraceable

Missing Requirements (count = 0)
-

Unknown Requirements (count = 0)
-

Untraceable Types (count = 0)
-

Requirement → Code Elements

Requirement (count = 4)	Code Elements
UC-Generate-Report	...JavaRequirementsTracerBean
UC-Overview-Report	...ReportAggregator
UC-Parseable-Report	...AttributeId
UC-Standalone-Report	...JavaRequirementsTracer

Code Element → Requirements

Code Element (count = 4)	Requirements
...AttributeId	UC-Parseable-Report
...JavaRequirementsTracer	UC-Standalone-Report
...JavaRequirementsTracerBean	UC-Generate-Report
...ReportAggregator	UC-Overview-Report

Requirements Descriptions

Label (count = 4)	Description
UC-Generate-Report	Generate traceabilities report.
UC-Overview-Report	Generate an overview report for multi-module application.
UC-Parseable-Report	Report must be in parseable XML format.
UC-Standalone-Report	Generate report using a standalone Java application.

6. Conclusion: In the author's opinion, Java Requirements Tracer is very good tool to base on for further development. But you should follow the specified steps. Using this tool you can increase the efficiency of you product, you will have isolated component testing, easier impact analysis, regulation compliance documentation, and better team communication.

References:

These are websites I use as a reference while doing assignment:

- **https://confluence.man.poznan.pl/community**
- **http://archiv.ub.uni-heidelberg**
- **https://www.ibm.com › com.ibm.java.vm.80.doc›docs**