

System Re-engineering
Assignment 3:
Re-Engineering AssertJ

Name: Shashidar Ette

User-id: se146

NOTE:

1. The code modified or updated during the course work are details ***Code Change*** boxes
2. Outputs generated during analysis are available under [*dataFiles*]
3. Supplementary files along with the report are available under [*notesAndReports*] folder

Table of Contents

Overview	4
Initial Exploration	4
History	4
Technical background	4
System perspective	5
Metrics and Analysis	6
Using Bash Scripts	7
Using the .class files	8
JUnit – Code Coverage.....	10
Repository log mining	11
Trace Analysis.....	11
The Big Picture	12
Visualisation - Code City.....	12
File Comparision for code duplication	14
Class Diagram.....	17
Code churn from release to release.....	19
Reengineering	21
Re-engineered Sections	23
Conclusion.....	24
References	25
1. Costigliola, J.....	25
2. AssertJ: API changes review	25

Table of Figures

Figure 1 Comparison between main source and JUnit source w.r.t number of classes/packages	6
Figure 2 Project Dependencies in Eclipse	6
Figure 3 Lines, LOC, Comments, % of Comments	7
Figure 4 Candidate classes for Re-engineering from code documentation perspective	8
Figure 5 Candidate methods (CC > 10) ordered by cyclometric complexity	9
Figure 6 Candidate classes for Re-engineering based on number of methods	10
Figure 7 Code coverage by JUnit tests	10
Figure 8 Candidate classes for re-engineering based on code coverage	10
Figure 9 Repository mining - code churn	11
Figure 10 Trace Analysis outputs	12
Figure 11 AbstractAssert superclass	18
Figure 12 core.internal.Arrays - composed within arrays assertion classes	19
Figure 13 GOD classes	19
Figure 14 API changes viewer[2]	20
Figure 15 Code churn from Release 2.8.0 to Release 3.9.0	20

Overview

This document details the analysis of AssertJ core from system re-engineering perspective. The sections within the document begin with an introduction to the system and then dwelling into details of it with the help of static and dynamic analysis techniques covered as part of System Re-engineering module.

Initial Exploration

As part of the initial exploration phase, I have primarily applied “Skim Documentation” pattern to understand the motivation behind the project and documentation available in terms of project overview, environment setup.

As part of “Chat with the Maintainers” pattern, I have also contacted with Joel Costigliola (joel.costigliola@gmail.com) for his views on the current state of the system and if he personally has any set of improvements or extended to be to the library. Incidentally, I got hold of the [Interview with Joel Costigliola, creator of AssertJ](#) which sheds light on some of key information. In addition, there an active Google+ forum (Joined it) with discussions dated till last week <https://groups.google.com/forum/?fromgroups=#!forum/assertj> but mostly GitHub is used as discussion by contributors.

History

- The motivation behind the library came from the limitation of Junit and assert instructions available with it.
- Initially Joel was contributing to [Fest Assert](#) created by [Alex Ruiz](#), but after some time upon request of Joel and Alex’s consent AssertJ got created as a fork for Fest Assert 2.x.
 - The main reasons to create a fork of Fest Assert were:
 - FEST 2.0 provided limited set of assertions than FEST 1.x
 - It was not open enough to users and contributors.
- Joel mentioned to keep AssertJ a more “community-driven” project.

Technical background

- “AssertJ core is a Java library that provides a fluent interface for writing assertions. Its main goal is to improve test code readability and make maintenance of tests easier.” [1]
- AssertJ helps developers to create unit tests for complex objects including lists and strings.
- The main project website is here: <http://joel-costigliola.github.io/assertj/assertj-core.html>
- As a developer, the system can be fairly understood at high level via [ReadMe](#)
- Quick start guide is available at: <http://joel-costigliola.github.io/assertj/assertj-core-quick-start.html>
- Github project is available here: <https://github.com/joel-costigliola/assertj-core>
 - Joel mentioned in the interview that he would like to move the git hub repository under AssertJ organization.
- AssertJ Core 3.9.0 is the latest released on 2nd Jan 2018. However it has different versions to support different versions of Java.
 - Assert 3.x requires Java 8 or higher

- Assert 2.x requires Java 7 or higher
 - Assert 1.x required Java 6 or higher
- Support for [Android](#)
- Supports several modules listed below (with Start guides are available for each):
 - Guava
 - Joda Time
 - Database
 - Neo4J
 - Swing
- Support for several dependency/deployment tools
 - Maven
 - Gradle
 - Ivy
 - Groovy
 - Scala as well
- Mockito-core is used as mock library for creation of proxy objects during test execution
- It provides [Assertion Generator](#) : it allows developers to create assertion for their own classes by “either through a simple [command line tool](#), a [Maven plugin](#) or a third-party [Gradle plugin](#)”.

System perspective

- Although the library is meant to be used for unit testing and write JUnit test cases. It a feature rich library.
- It has clearly documented guidelines for
 - Code of conduct
 - Contributor
 - Issue template
 - Pull request
- Active contributor list with contributor guidelines are available
<http://joel-costigliola.github.io/assertj/assertj-core.html#contributing>
<https://github.com/joel-costigliola/assertj-core/blob/master/CONTRIBUTING.md>
- The Code & Issues tracker is available on [Git Hub](#).
 As on 24th December, it had:
 - 10 active pull requests
 - Regarding Issues - 59 open items - mostly defects, some of them are new feature, improvements, documentation items

Later I have applied “Read Code in One Hour” pattern to go through the source code at a high level.

Important insights are:

- The code is documented extensively.
- It is structured reasonably as a set of packages with most of the code available under **core.api** package. There are a total of 21 packages.
- **core.api** package has abstract classes which act as a base for other concrete classes. Although the by names I sense some of the classes are related or might have duplicate code such as *Assertions.java* and *Java6Assersions.java* or *Assertions.java* and *AssertionsForClassTypes.java*

- **core.api** package is the biggest of the packages with 190+ java classes. Next section will further detail of analysis done to find some additional insights.
- JUnit test case set available is extensive with 2645 test classes under 119 packages with a total of 10K+ unit tests.
- In terms of ratio, for one class, there is an equivalent of ~4.6 test classes which is a good indicator.

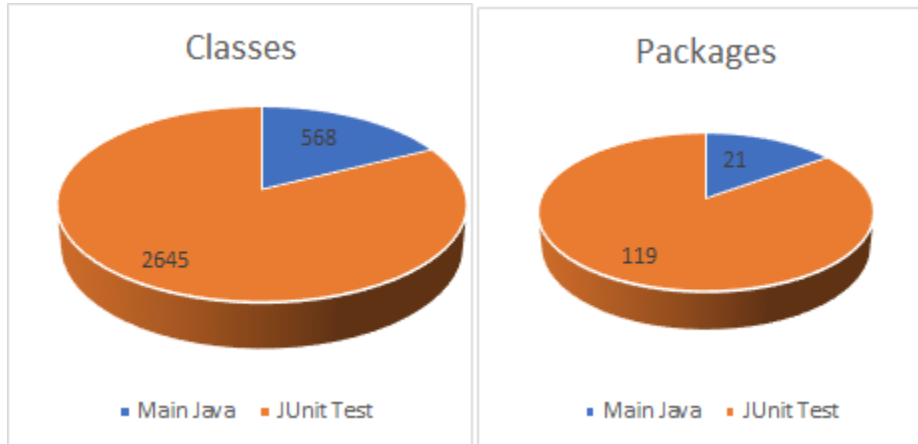


Figure 1 Comparison between main source and JUnit source w.r.t number of classes/packages

Metrics and Analysis

To take the re-engineering to next level, I have followed “Mock Install”, “Study Exceptional Entities” and “Speculate about the design” patterns which led to some of the interesting insights.

As part of “Mock Install” step:

- AssertJ Core is a maven based project comes with required dependencies to build the project.
- From the pom.xml, the version of the *assertj-core* is 3.9.0 whereas *assertj-parent-pom* is 2.1.6
- Details of the dependencies are as below:

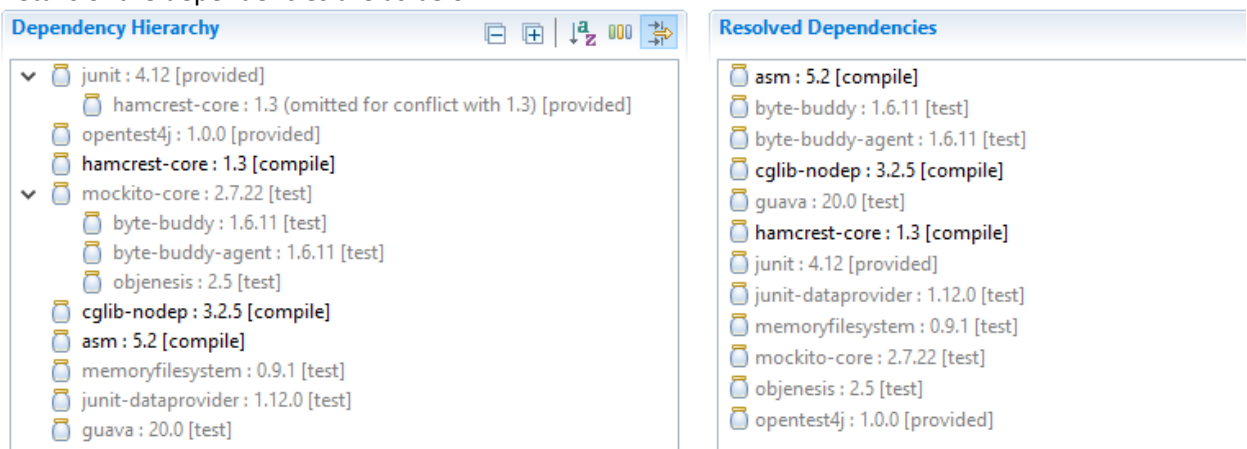


Figure 2 Project Dependencies in Eclipse

- The source code and project was imported in eclipse easily.
- It was successfully built without any build errors.
- Later I have executed JUnit test cases, there were total of 10861 test cases of which 91 were skipped with no failures and one of the tests in *Closeables_closeQuietly_Test* showed a warning.

As part of “Study Exceptional Entities” step:

- I have done static analysis of the source code in 2 ways i.e. Using Bash scripts and Analysing the .class files.

Using Bash Scripts

- Main motivation was to compute metrics such LOC, Comments and Vocabulary details of the code.
- I have used the scripts available under [*analysisCode\src\main\scripts*] folder.

Code Change: I have found that *countComments.sh* has an issue does not consider all the comment lines. Especially for asserj-core system the comments are of the format

```
/*
 *
 * .....
 * */
```

So I have committed a change in *countComments.sh*.

- Generated outputs are saved into .csv files via bash shell and are available under [*dataFiles\main_code_stats*] and [*dataFiles\test_code_stats*] folder
- Each of the folder has *lines.csv*, *comments.csv* and *vocab.csv* file generated using *countLines.sh*, *countComments.sh* and *countVocab.sh* script files respectively.
- For main java source code, I have combined the outputs from *lines.csv* and *comments.csv* into the *lines_comments.xlsx*.

Important insights:

- There are total of 568 classes in main source code.
- Percentage of comments against LOC

As mentioned in previous section, the code is documented extensively. In addition to other metrics. I have computed “% of comments” as below:

% of comments = number of lines in comments / total number of lines.

Java File Name	Lines	Updated comments	Updated LOC	% of comments	
Total Code	97396	58419	38977	59.98	59.9809
./java/org/asserj/core/api/AtomicReferenceArrayAssert.java	2842	2174	668	76.5	76.49543
./java/org/asserj/core/api/Assertions.java	2773	1903	870	68.63	68.62604
./java/org/asserj/core/api/AbstractObjectArrayAssert.java	2749	2065	684	75.12	75.11822
./java/org/asserj/core/api/WithAssertions.java	2683	1891	792	70.48	70.48081
./java/org/asserj/core/api/AbstractIterableAssert.java	2648	1688	960	63.75	63.74622
./java/org/asserj/core/api/AbstractDateAssert.java	2606	2093	513	80.31	80.31466
./java/org/asserj/core/api/Java6Assertions.java	2225	1558	667	70.02	70.02247
./java/org/asserj/core/api/AssertionsForClassTypes.java	1693	1142	551	67.45	67.45422
./java/org/asserj/core/api/BDDAssertions.java	1302	817	485	62.75	62.74962
./java/org/asserj/core/api/ObjectEnumerableAssert.java	1260	1148	112	91.11	91.11111
./java/org/asserj/core/api/AbstractMapAssert.java	1195	813	382	68.03	68.03347
./java/org/asserj/core/api/AbstractPathAssert.java	1173	999	174	85.17	85.16624
./java/org/asserj/core/api/Assumptions.java	1151	607	544	52.74	52.73675
./java/org/asserj/core/api/AbstractCharSequenceAssert.java	1148	868	280	75.61	75.60976
./java/org/asserj/core/internal/Iterables.java	1142	442	700	38.7	38.70403
./java/org/asserj/core/api/WithAssumptions.java	1022	607	415	59.39	59.39335
./java/org/asserj/core/api/Java6BDDAssertions.java	1016	653	363	64.27	64.27165
./java/org/asserj/core/api/AbstractDoubleArrayAssert.java	903	688	215	76.19	76.19048
./java/org/asserj/core/api/AbstractFloatArrayAssert.java	901	686	215	76.14	76.13762
./java/org/asserj/core/internal/Strings.java	886	362	524	40.86	40.85779
./java/org/asserj/core/internal/Objects.java	865	341	524	39.42	39.42197
./java/org/asserj/core/internal/Dates.java	830	429	401	51.69	51.68675
./java/org/asserj/core/api/AbstractZonedDateTimeAssert.java	829	585	244	70.57	70.56695
./java/org/asserj/core/api/AbstractByteArrayAssert.java	802	592	210	73.82	73.81546
./java/org/asserj/core/api/Java6AbstractBDDSoftAssertions.java	798	465	333	58.27	58.27068

Figure 3 Lines, LOC, Comments, % of Comments

For a system as a whole, code documentation amounts to ~60% which is an interesting insight.

Lines	Updated comments	Updated LOC	% of comments	
97396	58419	38977	59.98	59.9809

- Of the total code, **core.api** package contains

189 classes, ~65% of total lines, 73% of total code documentation and ~50% of actual LOC.

Java File Name	Lines	Updated comments	Updated LOC	% of comments
core.api.package	62624	42859	19765	68.43862

- From system re-engineering perspective, the classes which have => 250 LOC but less than <= 25% documentation. There are about 6 classes which might need attention.

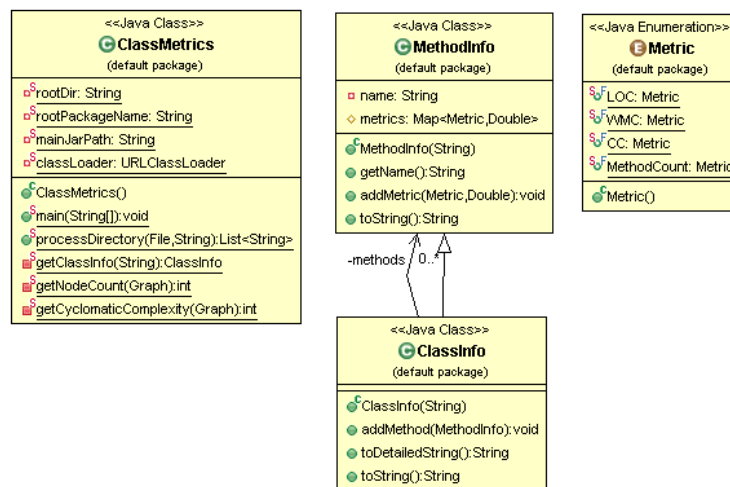
Java File Name	Lines	% of commen
./java/org/assertj/core/internal/Paths.java	312	4.81
./java/org/assertj/core/internal/Arrays.java	748	5.75
./java/org/assertj/core/api/ListAssert.java	310	8.06
./java/org/assertj/core/api/IterableAssert.java	295	9.15
./java/org/assertj/core/presentation/StandardRepresentation.java	517	12.96
./java/org/assertj/core/internal/DeepDifference.java	712	21.77

Figure 4 Candidate classes for Re-engineering from code documentation perspective

Using the .class files

- For additional metrics such as Cyclometric Complexity, Weighed Method count. I have used the ClassMetrics.java available under [analysisCode\src\main\java] folder.

Code Change: To make the analysis generic, I have extended the ClassMetrics.java to process the class files from a root directory. In addition to the extension, introduced new classes Metric.java, MethodInfo.java and ClassInfo.java.



- In addition to LOC and CC, computed method count and weighed method count of a class (sum of CC for all methods in a call).
- The solution generates 2 outputs in CSV format
 - class_metrics.csv – contains class level metrics
 - class_method_metrics.csv – contains class's method metrics
- Important insights:
 - Cyclometric Complexity of all the methods is maximum 36 which implies the methods with CC above 10 might need refactoring with
 - CC – 11-20 - More complex, moderate risk

- CC – 21-50 - Complex, high-risk program

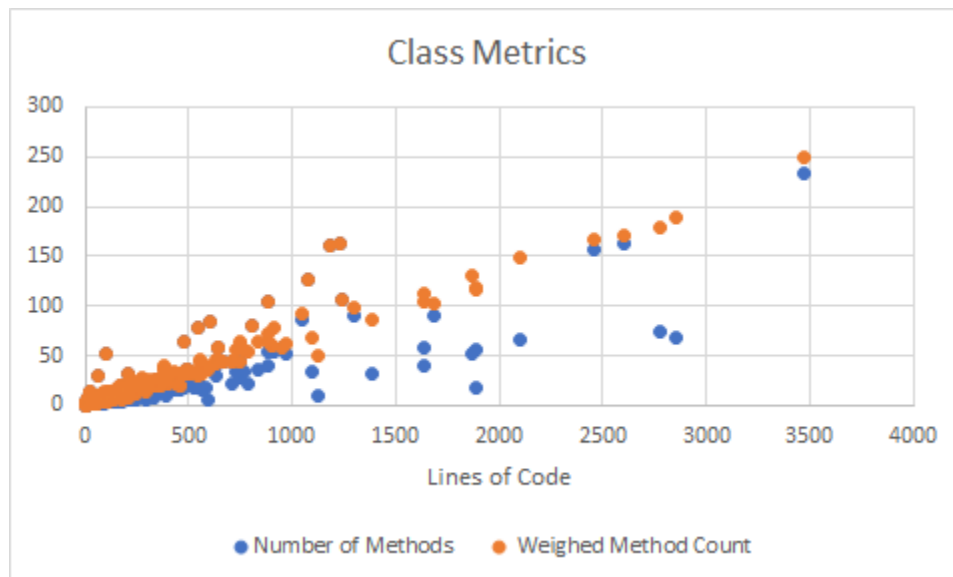
Class Method Name	LOC (from CFG)	Cyclometric Complexity
org/assertj/core/internal/DeepDifference.determineDifferences	621	36
org/assertj/core/presentation/StandardRepresentation.toStringOf	345	33
org/assertj/core/util/diff/DiffUtils.parseUnifiedDiff	420	25
org/assertj/core/util/DateUtil.formatTimeDifference	336	24
org/assertj/core/internal/Comparables.assertIsBetween	191	16
org/assertj/core/util/diff/myers/MyersDiff.buildPath	283	14
org/assertj/core/util/diff/myers/MyersDiff.buildRevision	210	14
org/assertj/core/internal/DeepDifference.deepHashCode	214	12
org/assertj/core/internal/DeepDifference.initStack	181	12
org/assertj/core/api/AbstractSoftAssertions.getFirstStackTraceElementFromTest	104	12
org/assertj/core/util/URLs.loadContents	122	11
org/assertj/core/presentation/StandardRepresentation.format	173	10
org/assertj/core/util/diff/Delta.equals	98	10
org/assertj/core/util/diff/DiffUtils.processDeltas	350	9
org/assertj/core/internal/Strings.assertContainsSequence	192	9

Figure 5 Candidate methods (CC > 10) ordered by cyclometric complexity

- Maximum LOC (from CFG) of all the methods is 621 nodes which implies that methods of above LOC could be candidates for refactoring or re-engineering.

- Classes:

Below chart is generated from ClassMetrics.xlsx (based on class_metrics.csv)



From the above chart, its evident to find some of the candidates for re-engineering.

- about ~30 classes have more than 50 methods
- about ~20 classes have above 1000 LOC

Class Name	LOC (from Methods CFG)	Number of Methods
org/assertj/core/api/AbstractIterableAssert	3468	233
org/assertj/core/api/AbstractObjectArrayAssert	2604	162
org/assertj/core/api/Assertions	1232	162
org/assertj/core/api/WithAssertions	1185	161
org/assertj/core/api/AtomicReferenceArrayAssert	2453	157
org/assertj/core/api/Java6Assertions	1072	127
org/assertj/core/api/AbstractMapAssert	1240	106
org/assertj/core/api/AbstractDateAssert	1684	91
org/assertj/core/api/AbstractAssert	1297	91
org/assertj/core/api/Assumptions	1047	86
org/assertj/core/internal/Iterables	2776	74
org/assertj/core/internal/Arrays	2858	69
org/assertj/core/internal/Strings	2101	66
org/assertj/core/internal/Dates	1632	58
org/assertj/core/internal/Objects	1885	56
org/assertj/core/presentation/StandardRepresentation	1864	53

Figure 6 Candidate classes for Re-engineering based on number of methods

JUnit – Code Coverage

- In the previous section, although it was conveyed that test case bed is extensive. But I wanted to capture the code coverage by JUnit test cases.
- To do this analysis, I have used EcJemma plugin to calculate the code coverage. The results are available under [dataFiles\JUnitCodeCoverage] folder.
- Overall coverage is 84% and Main source code coverage is 91%

assertj-core

Element	Missed Instructions	Cov.
src/test/resources		n/a
src/test/java	<div><div></div></div>	82%
src/main/java	<div><div></div></div>	91%
Total	49,892 of 325,524	84%

Figure 7 Code coverage by JUnit tests

- If we look at the **core.api** package coverage, order by number of methods missed w.r.t coverage below list of classes will be candidates for re-engineering.

org.assertj.core.api

Element	Missed Instructions	Cov.	Missed	Lines	Missed	Methods	Missed	Classes
Java6Assertions	<div><div></div></div>	2%	137	140	124	127	0	1
WithAssumptions	<div><div></div></div>	0%	78	78	78	78	1	1
WithAssertions	<div><div></div></div>	51%	88	178	76	161	0	1
Java6BDDAssertions	<div><div></div></div>	0%	64	64	64	64	1	1
AssertionsForClassTypes	<div><div></div></div>	59%	65	116	53	104	0	1
BDDAssertions	<div><div></div></div>	57%	37	85	37	85	0	1
AbstractMapAssert	<div><div></div></div>	73%	32	98	32	67	0	1
AbstractIterableAssert	<div><div></div></div>	88%	43	299	28	140	0	1
AbstractListAssert	<div><div></div></div>	53%	29	57	25	43	0	1
Assertions	<div><div></div></div>	88%	24	179	17	162	0	1
Java6AbstractStandardSoftAssertions	<div><div></div></div>	72%	16	58	16	58	0	1
AbstractBDDSoftAssertions	<div><div></div></div>	53%	10	22	10	22	0	1

Figure 8 Candidate classes for re-engineering based on code coverage

Repository log mining

Another aspect of an evolving system is the code churn. As part of “Learn from the Past” pattern, I have done repository mining of assert-core git repo. I have considered latest 1000 commits for analysis. The outputs generated are available under [dataFiles\RepositoryMiningOutputs] folder.

Code Change: Updated *rep-mining-churn.sh* as needed for assertj-core repo to check out revision 9d45e93.

Insights: Below charts generated from diffs generated as part of code churn, clearly show either additions or deletion of code are mostly done in classes part of **core.api** package with few of them in *util* and *internal* packages.

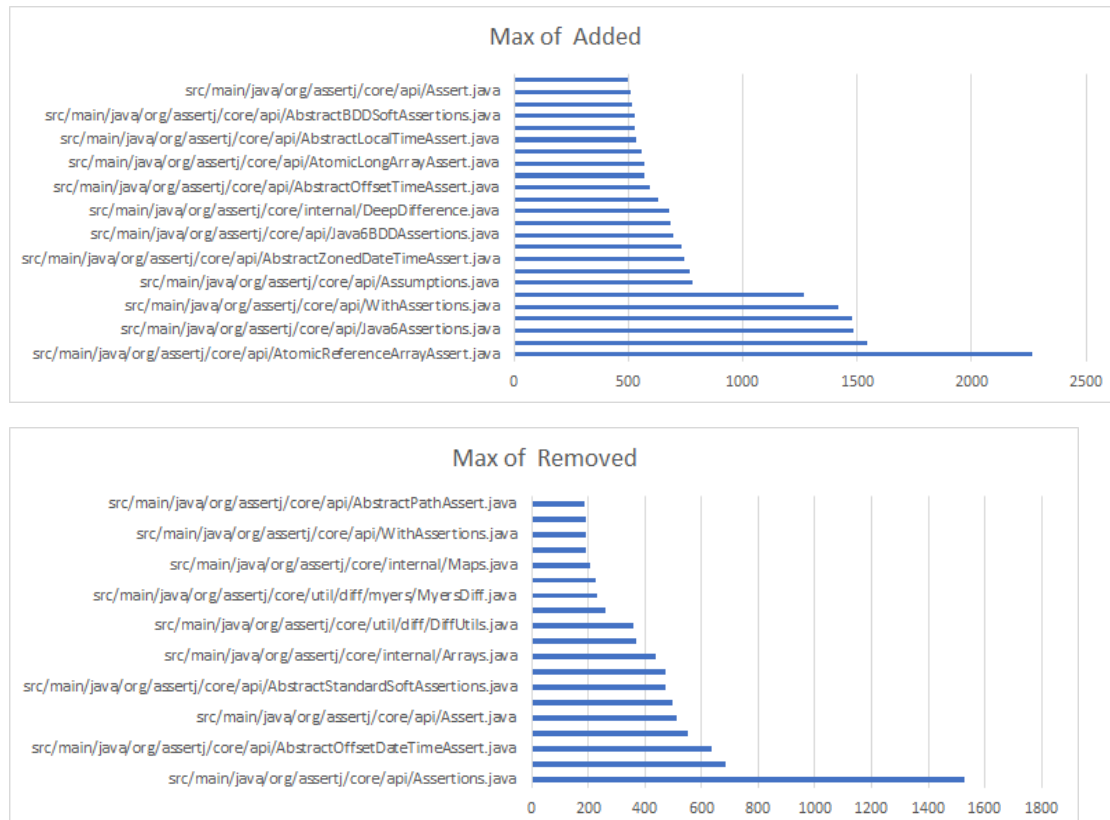


Figure 9 Repository mining - code churn

Trace Analysis

Considering asserj-core is a library of assertions which is less intensive than an UI library such as jfreechart. I have used AspectJ weaving to analyse the traces for tests executed for **core.api** package.

Code Change:

- Aspect file **AssertJAspect.aj** added under [analysisCode\src\main\java\aspects]. This is referred from coursework 2 of System Re-engineering. Updated for AssertJ core library.
- Added **CoreApiTestSuite** and **CoreApiTestClass** under [org.assertj.core.api] test class package. These classes are added to generate a JUnit executable jar. The generated jar *CoreApiTest.jar* is used along with Aspect.jar for trace recoding.

Insights:

Initially the trace was generated with a filter as “org.assertj.core”, the output file was ~330MB which was difficult to analyse. To narrow down the trace, filter was changed to “org.assertj.core.api” which

generated ~3MB output (available under dataFiles\TraceAnalysis). Based on the TraceAnalysis i.e. Number of occurrences of a class, the top 15 classes include:

- WritableAssertionInfo
- Assertions
- AssertionsForClassTypes
- ListAssert
- Abstract*Assert classes.

Class Name	Count of ClassName	Max of StackDepth
org.assertj.core.api.SoftProxies\$CollectErrorsOrCreateExtractedProxy	13054	15
org.assertj.core.api.WritableAssertionInfo	1013	10
org.assertj.core.api.Assertions	711	5
org.assertj.core.api.AssertionsForClassTypes	452	6
org.assertj.core.api.ListAssert\$listFromStream	435	8
org.assertj.core.api.AbstractAssert	362	9
org.assertj.core.api.Assertions_sync_assertThat_with_BDD_and_Soft_variants_Test	330	4
org.assertj.core.api.AbstractObjectAssert	286	7
org.assertj.core.api.AbstractIterableAssert	172	8
org.assertj.core.api.AssertionsForInterfaceTypes	164	5
org.assertj.core.api.ListAssert	121	4
org.assertj.core.api.WithAssertions	114	4
org.assertj.core.api.BaseAssertionsTest	102	9
org.assertj.core.api.AbstractSoftAssertions	98	11
org.assertj.core.api.WithAssertions_delegation_Test	94	8
org.assertj.core.api.AbstractTemporalAssert	77	4
org.assertj.core.api.SoftAssertionsTest	72	9
org.assertj.core.api.Assertions_assertThatExceptionOfType_Test	71	6
org.assertj.core.api.SoftProxies	64	12

Figure 10 Trace Analysis outputs

With regards to Stack Depth, considering the trace is filtered for **core.api** the maximum call depth of 15 is a good indicator and not an alarming one.

The Big Picture

As part of this section, I have applied “Learn from the Past” “Speculate about Design”, “Compare Code Mechanically”, “Visualize Code as Dot plots” patterns with help of different visualization and analysis techniques to understand the system in detail in turn find out candidates for re-engineering.

It covers insights from below items:

- Code City visualizing software
- File comparison to find out the duplication
- Release to Release comparison analysis
- Class diagram

Visualisation - Code City

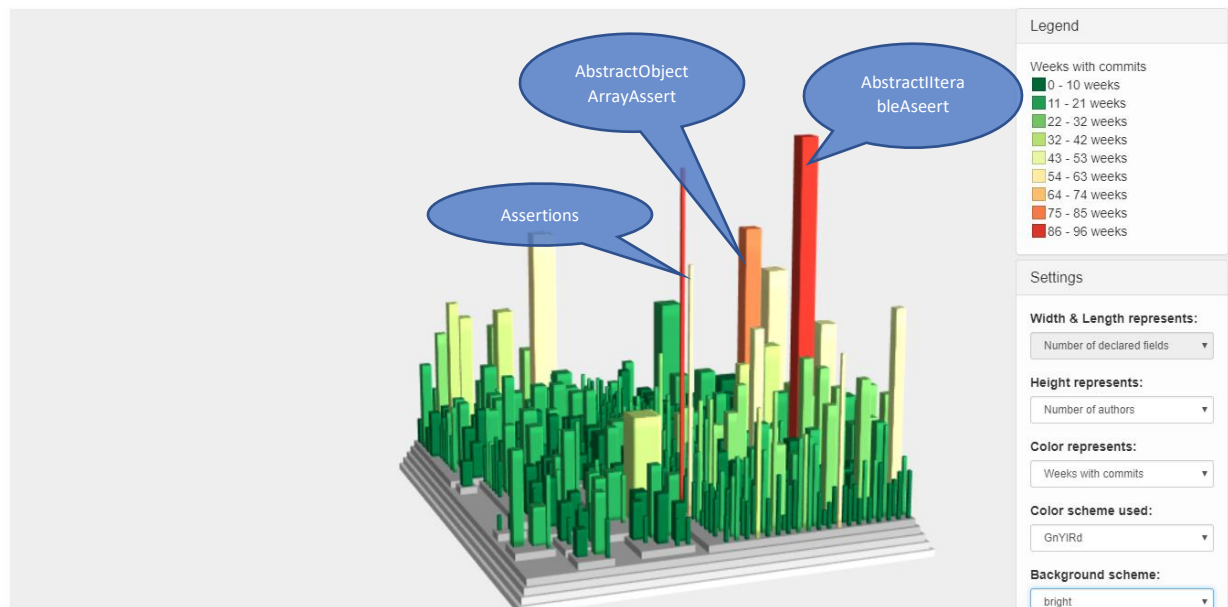
To understand more about the system and to apply “Learn from the Past”, I have taken the code from original asserj-core repository @ 9d45e93. Used the Code City visualization software from Eclipse to generate additional insights based on the repository log from git.

Below is the visualization of main source code java with color representing the number of weeks with commits and height representing number of authors. Notably below entities stand-out:

- AbstractIterableAssert.java – 88 weeks with commits & 32 authors
- Assertions.java – 96 weeks with commits & 30 authors

- AbstractObjectArrayAssert.java - 79 weeks with commits & 23 authors
Other entities are relative to them.

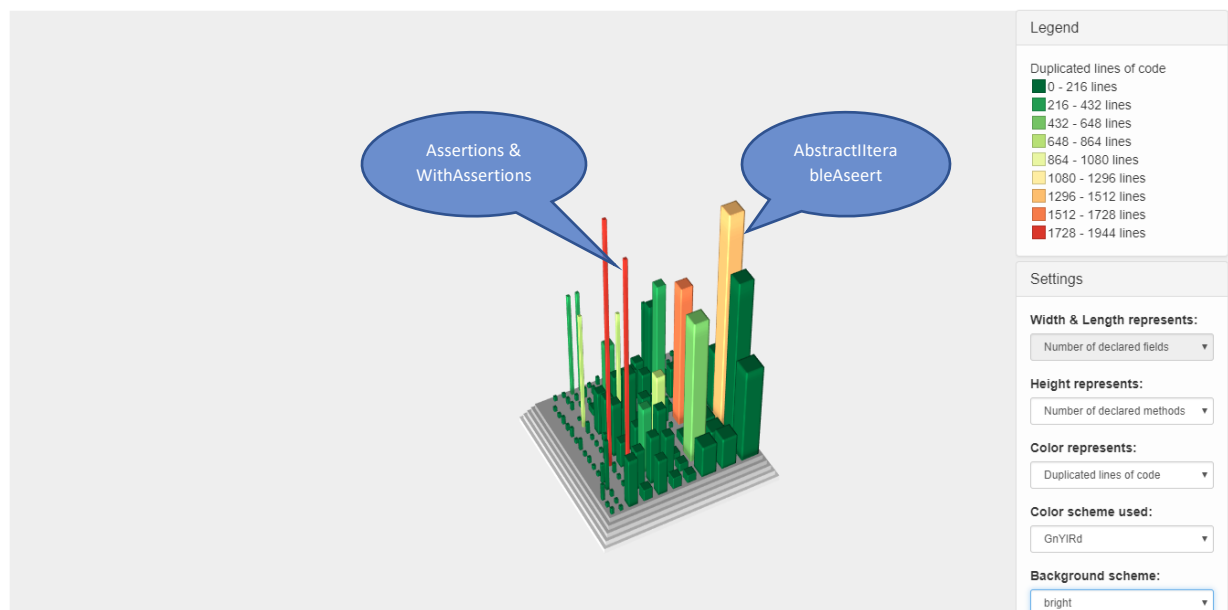
SoWaCa Visualizing software



Below is the visualization of only core.api package with color showing duplicated lines of code and height showing number of declared methods. Interestingly the same entities standout similar to above visualization.

SoWaCa Visualizing software

Press F11 to exit full screen



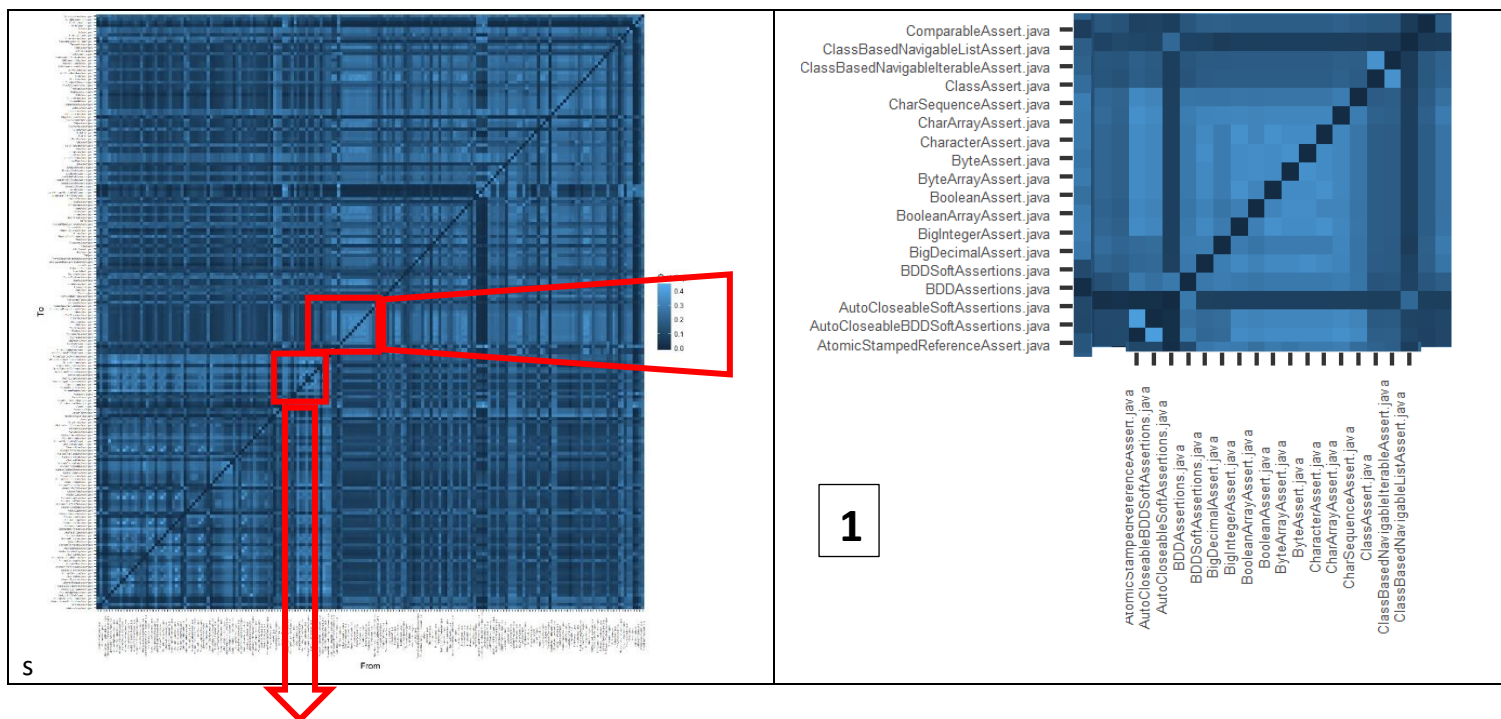
File Comparision for code duplication

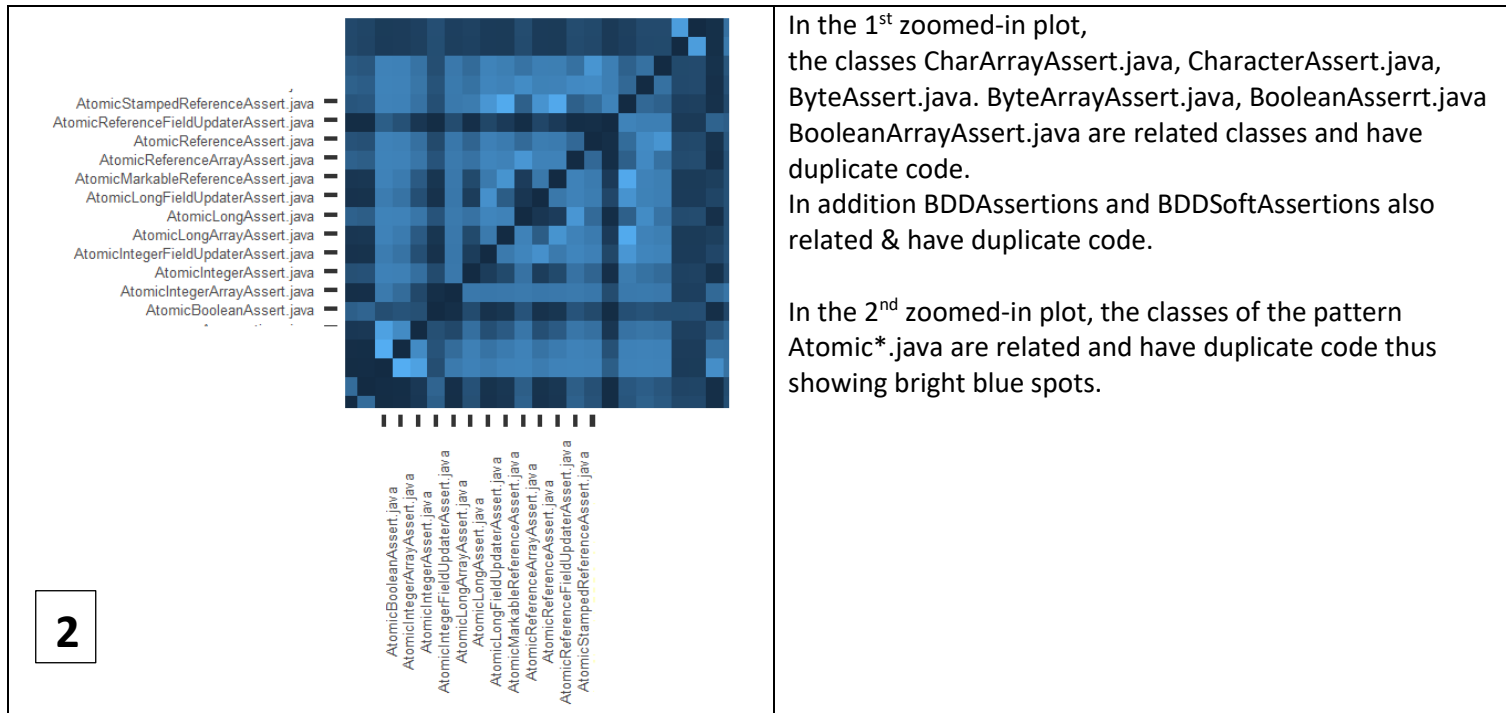
As part “Visualize Code as Dot plots”, I have used the file comparison code available under *analysisCode* and R-Studio to generate the dot plots using *fileComparisonScripts.R* script.

To make the comparison structured, I have considered the code base at package level instead of whole code together.

- Package Core.API

From the below dot plot it is evident that there is duplication of code from bright blue spots. One of the duplication is the documentation of the code which is repeated in a uniform structure. But there are couple of areas which stand-out in the dot plot. These areas are zoomed out with areas 1 and 2 below:

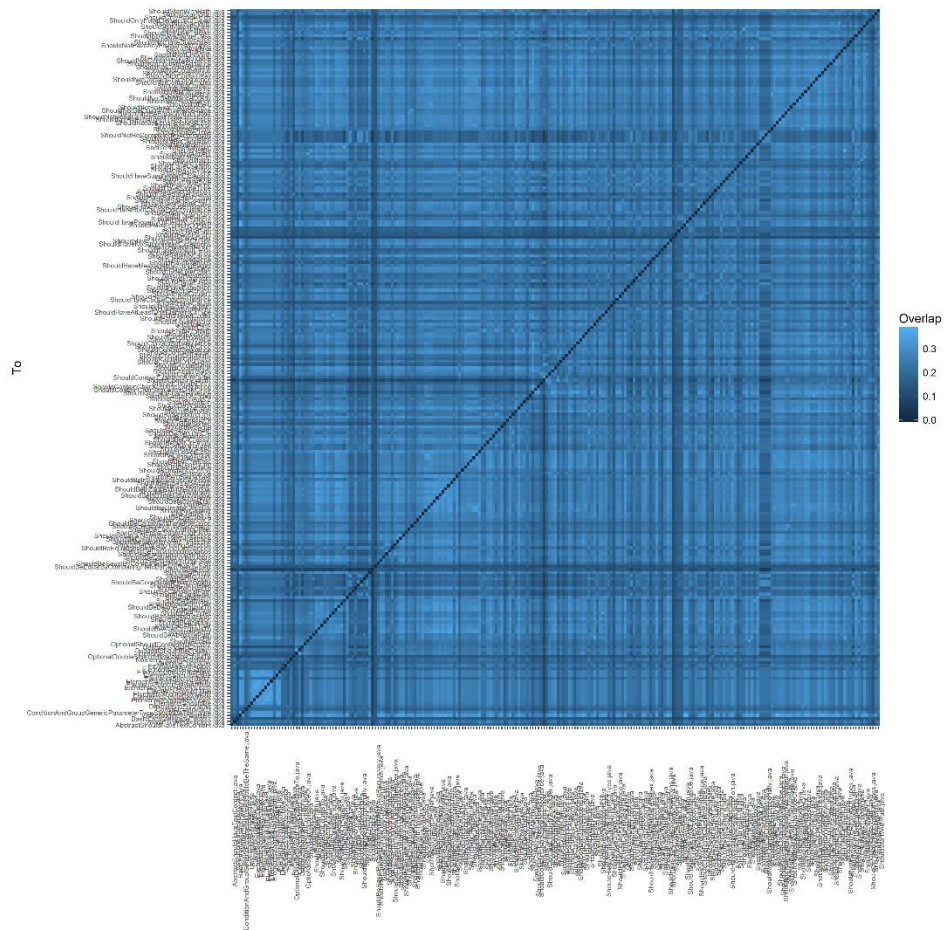




- Package Error
- The dot plot for this package shows lot of duplication. Since most of the classes in this package have similar classes names and functions within each of the classes follow uniform naming convention.

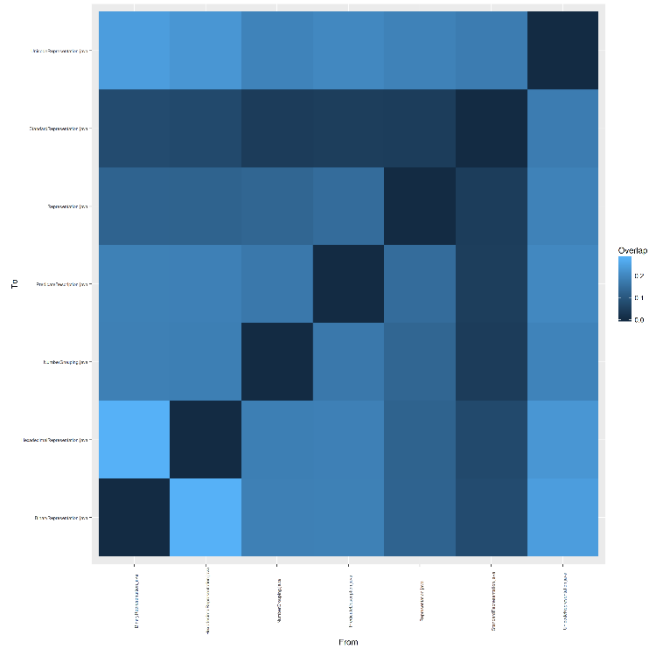
There are group of classes which are particularly related are:

- Element*.java
- ShouldBe*.java
- ShouldContain*.java
- ShouldHave*.java etc.

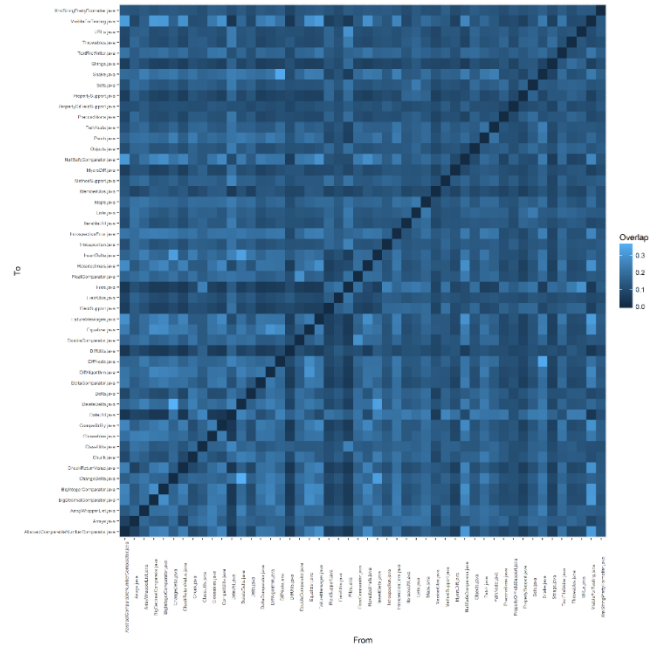


For other significant packages, the duplication arises due to classes inherited from same base class whereas package util has fairly distinct code since each of the classes have separate responsibility.

Package Representation:



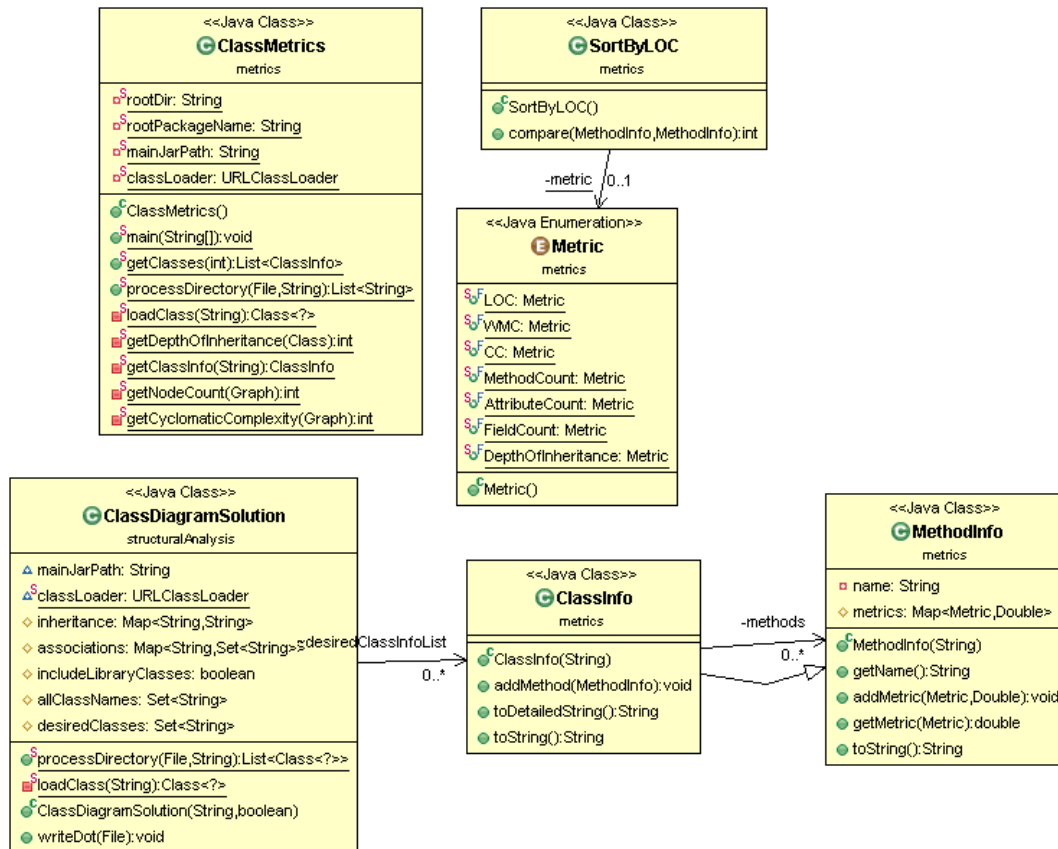
Package Util



Class Diagram

For Class Diagram solution, I have extended the Class Metric solution.

Code Change: In addition to the existing metrics, I have added *AttributeCount*, *FieldCount* and computed metric for each class *DepthOfInheritance*. *DepthOfInheritance* is computed as number of levels of inheritance until the Superclass is java.object class.



In terms of class generation in the dot file, important notations are:

- Number of methods in a class is represented by width of the box shape
- LOC in a class is represented by height of the box shape
- *DepthOfInheritance* is represented by text font size of the class, so higher the depth larger the font size

Based on the updated solution, the output generated is available as `EnhancedClassMetrics.csv` under `[dataFiles]` folder. The dot file and pdf generated using dot utility are available under `[dataFiles\classDiagramSolution]` folder.

As per “Speculate about Design” and “Study the Exceptional Entities” patterns, from the class diagram generated

- The classes look highly related showing cohesion
- `AbstractAssert` stands out as an important class as being the superclass of most of the Assertion classes.

In the second image, `core.api.internal.Arrays` shown as part of Arrays assertion classes. In the third image, the 4 classes below due to their LOC and number of methods stand out in the class diagram as GOD classes.

- `org.assertj.core.api.AbstractObjectArrayAssert`
- `org.assertj.core.api.AtomicReferenceArrayAssert`
- `org.assertj.core.api.AbstractIterableAssert`
- `org.assertj.core.internal.Iterables`

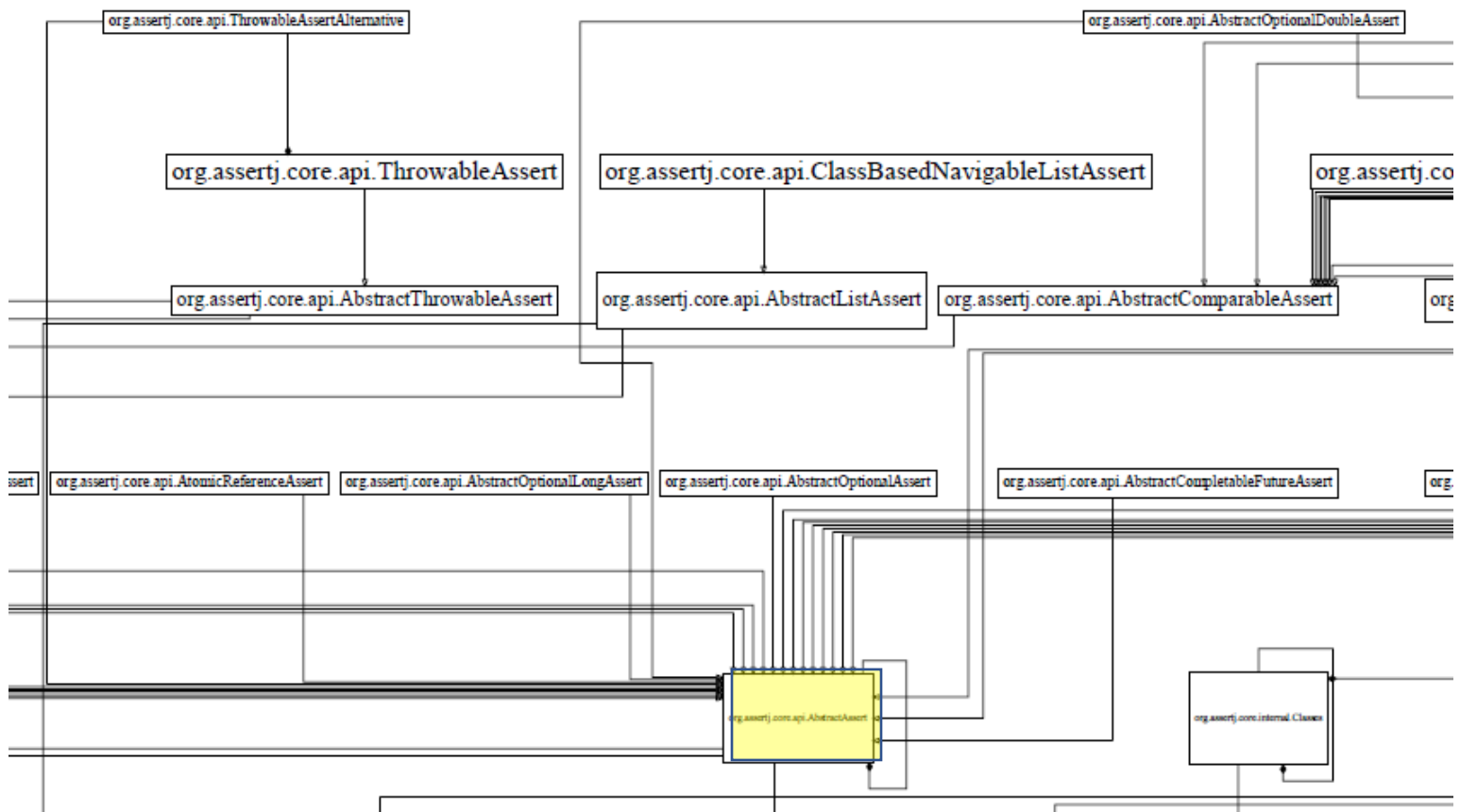


Figure 11 AbstractAssert superclass

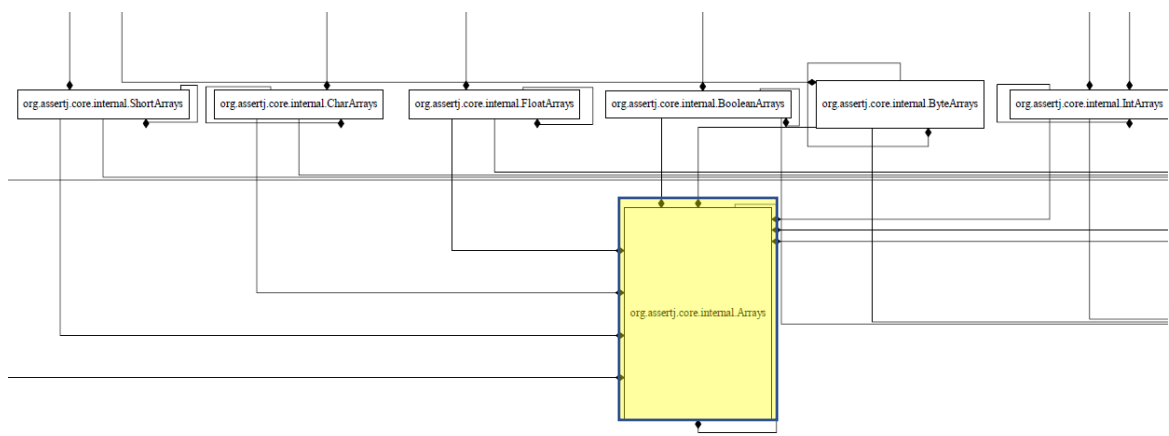


Figure 12 core.internal.Arrays - composed within arrays assertion classes

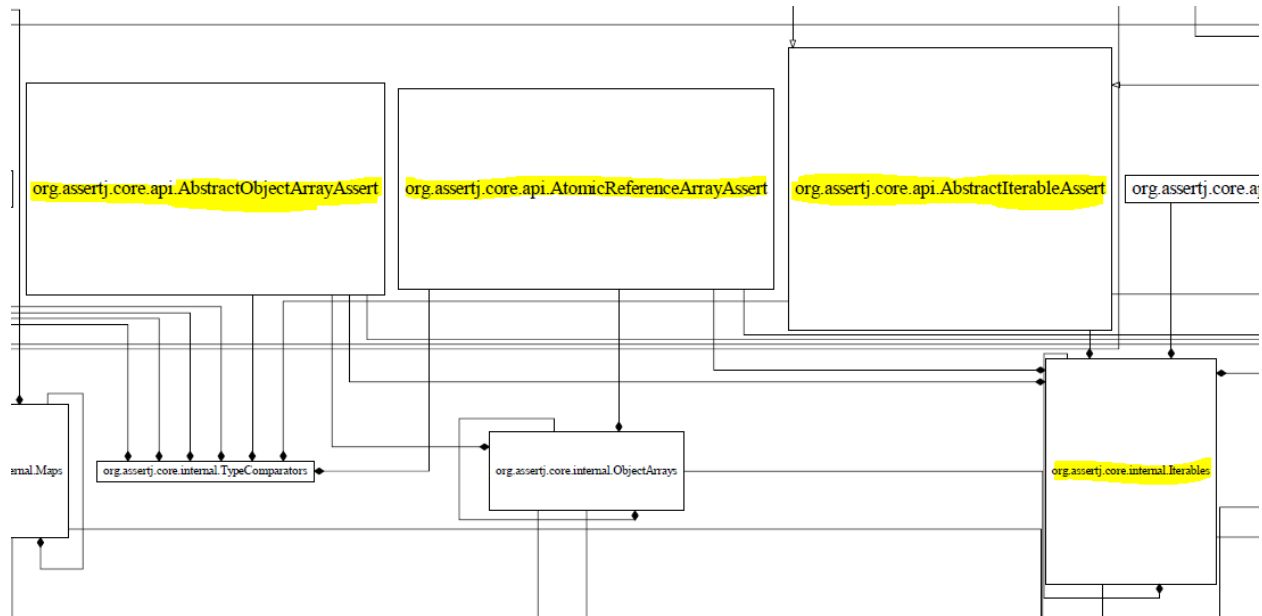


Figure 13 GOD classes

Code churn from release to release

AssertJ core as a library had many releases since its inception. There has been total of 37 releases from release 1.0.0 in March 2013 to the latest release 3.9.0 dated 2nd Jan 2018. One of the important aspect is the code churn from release to release.

Incidentally I found an interesting [link](#) on git hub of the system, which discussed API tracking from release to release. More information can be found at this link: <https://ab-laboratory.pro/java/tracker/timeline/assertj-core/>

Image [2] in the next page shows the code churn from release 3.1.0 till 3.9.0. The image shown the code churn in terms of methods added and removed. However I was interested in code churn from release 2.8.0 (supports Java 7) to release 3.9.0 (supports Java 8). I have considered the code from this tag: <https://github.com/joel-costigliola/assertj-core/releases/tag/assertj-core-2.8.0> and generated class metrics and class method metrics as detailed in [Using the .class files](#) section.

Noticibly between Release 2.8.0 and Release 3.9.0 –

- There were 82 new classes added and about 1300 new methods
- About 20 K lines of new source code added (approx. 60% being code documentation)
- Majority changes happened in classes in **core.api** package which in-line with earlier analysis.

API changes review

Version	Date	Backward Compatibility		Added Methods	Removed Methods
		BC	SC		
3.9.0	2018-01-02	98.3%	96.9%	426 new	41 removed
3.8.0	2017-05-21	100%		32 new	0
3.7.0	2017-05-07	77.2%	75.6%	691 new	29 removed
3.6.2	2017-01-21	100%		0	0
3.6.1	2016-11-27	99.97%		1 new	1 removed
3.6.0	2016-11-21	98.8%	96.6%	259 new	40 removed
3.5.2	2016-07-17	100%		0	0
3.5.1	2016-07-03	100%		0	0
3.5.0	2016-07-03	88.3%	87.3%	409 new	14 removed
3.4.1	2016-04-10	100%		0	0
3.4.0	2016-03-30	99.8%	98.6%	402 new	0
3.3.0	2016-01-10	98.3%	97.1%	248 new	16 removed
3.2.0	2015-09-21	95.2%		289 new	108 removed
3.1.0	2015-06-25	97%	85.4%	393 new	62 removed

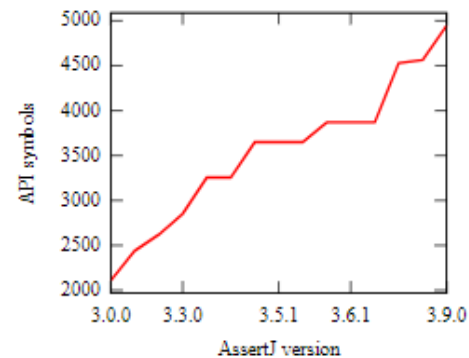


Figure 14 API changes viewer [2]

	Package	Classes	Total Methods	Total Lines	LOC	Comments	% of Comments
Release 2.8.0	20	516	5174	75828	44421	31407	58.58
Release 3.9.0	21	598	6443	97396	58419	38977	59.98
	NEW 82			21568			

Class in Release 2.8.0	LOC	Class in Release 3.9.0	LOC	% of Change
org/assertj/core/api/AbstractIterableAssert	2883	org/assertj/core/api/AbstractIterableAssert	3468	20.29
org/assertj/core/internal/Arrays	2690	org/assertj/core/internal/Arrays	2858	6.25
org/assertj/core/internal/Iterables	2379	org/assertj/core/internal/Iterables	2776	16.69
org/assertj/core/api/AbstractObjectArrayAssert	2116	org/assertj/core/api/AbstractObjectArrayAssert	2604	23.06
org/assertj/core/internal/Strings	2101	org/assertj/core/internal/Strings	2101	0.00
org/assertj/core/api/AtomicReferenceArrayAssert	2096	org/assertj/core/api/AtomicReferenceArrayAssert	2453	17.03
org/assertj/core/internal/Objects	1852	org/assertj/core/internal/Objects	1885	1.78
org/assertj/core/api/AbstractDateAssert	1684	org/assertj/core/api/AbstractDateAssert	1684	0.00
org/assertj/core/internal/Dates	1632	org/assertj/core/internal/Dates	1632	0.00
org/assertj/core/presentation/StandardRepresentation	1452	org/assertj/core/presentation/StandardRepresentation	1864	28.37
org/assertj/core/api/AbstractMapAssert	1220	org/assertj/core/api/AbstractMapAssert	1240	1.64
org/assertj/core/api/AbstractAssert	1159	org/assertj/core/api/Java6Assertions	1072	-7.51
org/assertj/core/api/Assertions	1114	org/assertj/core/api/Assertions	1232	10.59

Figure 15 Code churn from Release 2.8.0 to Release 3.9.0

Reengineering

This section reviews the AssertJ core system with respect to re-engineering opportunities and listing the candidates of it. Based on the analyses done in previous section, highlights are:

- AssertJ Core although started by Joel Costigliola, its community-driven project.
- The project has a strong coding and contributor guidelines. With a total of 119 authors, 37 releases (since 2013) and 2000 commits the assertion library is feature rich with a detailed documentation.
- The main source code and test code are structured into several packages.

The list of re-engineering opportunities is as below:

- GOD classes

Due to extensive code documentation, some of the classes appear larger in size. The actual LOC is at most 960. The classes which have methods [more than 50](#) are direct for re-engineering.

Class Name	LOC (from Methods CFG)	Number of Method
org/assertj/core/api/AbstractIterableAssert	3468	233
org/assertj/core/api/AbstractObjectArrayAssert	2604	162
org/assertj/core/api/Assertions	1232	162
org/assertj/core/api/WithAssertions	1185	161
org/assertj/core/api/AtomicReferenceArrayAssert	2453	157
org/assertj/core/api/Java6Assertions	1072	127
org/assertj/core/api/AbstractMapAssert	1240	106
org/assertj/core/api/AbstractDateAssert	1684	91
org/assertj/core/api/AbstractAssert	1297	91
org/assertj/core/api/Assumptions	1047	86
org/assertj/core/internal/Iterables	2776	74
org/assertj/core/internal/Arrays	2858	69
org/assertj/core/internal/Strings	2101	66
org/assertj/core/internal/Dates	1632	58
org/assertj/core/internal/Objects	1885	56
org/assertj/core/presentation/StandardRepresentation	1864	53

- Duplicate Code








As part of visual analysis, it's evident that there is a duplication of code between several related classes. We can employ automatic comparison or use "Compare Code Mechanically" to refactor out duplicate code.

- Move behaviour close to data

During the static analysis of the code and skim through found instances where a code or a function could be moved into respective class.

- **core.api** package is the largest with 180+ classes within it. The package can be further broken down in more structured way. The proposed changes are as follows:

- SoftAssertions is a crucial feature of AssertJ core which allows to capture output of all the assertions of testing an object instead of returning after first assert failed. So the below classes and related can be considered to be part of a sub-package **core.api.soft**

- >  SoftAssertionClassAssert.java
 - >  SoftAssertionError.java
 - >  SoftAssertionIterableAssert.java
 - >  SoftAssertionListAssert.java
 - >  SoftAssertionMapAssert.java
 - >  SoftAssertionPredicateAssert.java
 - >  SoftAssertions.java

- There are assertion classes compatible with Andriod. These classes can also be considered to be part of a sub-package **core.api.andriod**

For example: Arrays.java lacks code documentation for most of the functions.

Java File Name	Lines	Updated comments	Updated LOC	% of comments	
./java/org/assertj/core/internal/Paths.java	312	15	297	4.81	4.807692
./java/org/assertj/core/internal/Arrays.java	748	43	705	5.75	5.748663
./java/org/assertj/core/internal/Uri.java	183	12	171	6.56	6.557377
./java/org/assertj/core/api/ListAssert.java	310	25	285	8.06	8.064516
./java/org/assertj/core/internal/Urls.java	141	12	129	8.51	8.510638
./java/org/assertj/core/api/IterableAssert.java	295	27	268	9.15	9.152542
./java/org/assertj/core/api/ErrorMessageCollector.java	122	13	109	10.66	10.65574
./java/org/assertj/core/internal/CommonValidations.java	147	16	131	10.88	10.88435
./java/org/assertj/core/error/uri/ShouldHaveParameter.java	105	12	93	11.43	11.42857
./java/org/assertj/core/internal/ErrorMessage.java	147	17	130	11.56	11.56463
./java/org/assertj/core/util/Introspection/PropertyOrFieldSupport.java	98	12	86	12.24	12.2449
./java/org/assertj/core/presentation/StandardRepresentation.java	517	67	450	12.96	12.95938
./java/org/assertj/core/api/SoftProxies.java	83	12	71	14.46	14.45783

- Unit test case set is extensive, but there are classes which have code coverage less than 65%. These can be considered for re-engineering to increase JUnit test base.

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
WithAssertions		51%		n/a	76	161	88	178	76	161
AbstractListAssert		53%		n/a	25	43	29	57	25	43
AbstractBDDSoftAssertions		53%		n/a	10	22	10	22	10	22
BDDAssertions		57%		n/a	37	85	37	85	37	85
AssertionsForClassTypes		59%		n/a	53	104	65	116	53	104
AbstractInputStreamAssert		61%		n/a	1	3	2	7	1	3
AbstractOptionalDoubleAssert		64%		50%	5	12	1	21	0	7

- Architectural enhancements
During the static analysis of the code and skim through found instances where a group of classes structured hierarchically. Additional analysis could find out instances where the overall structure of the API can be improved.

Re-engineered Sections

From the identified candidates for re-engineering, the process of re-engineering is done in an iterative way. As per “Tests! Your Life Insurance” pattern, every change done to the system while re-engineering should be followed by extending the test base and make sure there are no failures.

Since Assertj-core has an extensive unit test base. It’s easier to refactor and test the modified code. However, the code base is large, it’s easier to start with smaller changes than get into bigger ones. i.e. to follow “Refactor to Understand” pattern.

As part of the engineering process, I have used Extract Class, Extract Method, Encapsulate Field, Move method refactor techniques.

Some of the re-engineering completed as part of the course-work are detailed below.

- Split GOD classes and remove duplication of code: I have considered below classes for Split and remove duplicate code.
core\api\Assertions.java
core\api\AssertionsForClassTypes.java
core\api\AssertionsForClassTypes.java

Have 900+ lines of common code. All the code is set of utility functions which are duplicated in each class, hence it is been moved to new AssertionBase class. This splits the GOD classes and reduces code duplication.

Code Change:

a596425 :

Re-engineering - AssertionsBase introduced

- Split GOD classes - Duplication of code in Assertions.java,
AssertionsForClassTypes.java, Java6Assertions.java
8e5b87a

Re-engineering - AssertionsBase update

remove duplication of code from:

core\api\Assertions.java

core\api\AssertionsForClassTypes.java

core\api\AssertionsForClassTypes.java

2. Added a contract for ArrayAssert so that all sub-classes implement required operations

Code Change: 7dd971b

Re-engineering - design update - updated the contract of an ArrayAssert class

3. Moved SoftAssertions related assertion classes to a new package core.api.soft

Code Change: 715b3eb

4. Moved Atomic* related assertion classes to a new package core.api.atomic

Code Change: d3a0fc5

5. Moved Date and Time-related assertion classes to a new package core.api.time

Code Change: 763753f

6. Moved Android related assertion classes to a new package core.api.android

Code Change: ad67d0c

7. Moved method close to data for couple of instances

Code Change: 01b00fd

Re-engineering - Move code close to data

From [DescriptionFormatter.java](#) to [Description.java](#)

From [StandardRepresentation.java](#) to [PredicateDescription.java](#)

8. Re-engineering - Moved method closed to data - 12a2821
- Moved Date parse related functions from AbstractDateAssert to DateUtil class
9. Re-engineering - Code documentation & Move to a package - d24c46d

Conclusion

Analysis of AssertJ core from system re-engineering perspective has given very valuable insights. Each of the object-oriented re-engineering pattern needed application of different analysis tool/techniques such as AspectJ, Bash Scripts, R Studio & R scripts, Code City, Ecl Emma, Excel, ASM and more importantly Java programming language.

Although some of the insights found during analysis felt simple to considered for re-engineering. During actual refactoring of the concerned areas seemed difficult so candidates considered earlier had to left to be as it. Overall the analysis findings enabled to re-engineer the system and in turn improve it.

References

1. Costigliola, J.
Costigliola, J. (2018). AssertJ / Fluent assertions for java. [online] Joel-costigliola.github.io. Available at: <http://joel-costigliola.github.io/assertj/assertj-core.html> [Accessed 23 Dec. 2017].
2. AssertJ: API changes review
Abi-laboratory.pro. (2018). AssertJ: API changes review . [online] Available at: <https://abi-laboratory.pro/java/tracker/timeline/assertj-core/> [Accessed 3 Jan. 2018].