**Initial Exploration**

In the first contact with a system we will try to understand what is the system, what does the system do, how the system works and how is it structured.

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.

AssertJ core provides assertions for JDK standard types and can be used with JUnit or TestNG.

AssertJ is a Maven-buildable.

We will investigate the system to gain a broad understanding of its **function**, **key components** and **architectural features**. We will use some re-engineering patterns and we will explain how we apply these patterns and what we learn about the system using these patterns.

* **Read all the Code in One Hour**

The intent is to assess **the state of the system** “first impression of the **quality** of the source code” and describes the **functionality** **of the system**.

We will use a generic checklist to review the kind of code we are dealing with.

|  |  |
| --- | --- |
| Are we able to understand the code easily? | Yes |
| Is the code written following the coding standards/guidelines? | Yes |
| Do classes have a copyright header? | Yes |
| Do the classes do what the name of the class claims that they’ll do? | Yes |
| Are classes too big? Do classes have too many methods? | Yes |
| Are methods too long? | Yes |
| Do methods have too many parameters? | No |
| Do classes and methods commented well? | Yes |
| Does the code compile without errors and run without exceptions? | Yes |
| Does the code compile without warnings? | No |
| Can we unit test / debug the code easily? | Yes |
| Do the methods do what the name of the method claims that they’ll do? | Yes |
| Can we get an understanding of the desired behaviour just by doing quick scans through unit and acceptance tests? | Yes |
| Is this code introducing any new dependencies between classes/components/modules? | Yes |
| Does the system includes many test units and test cases? | Yes |
| Does the system includes interfaces and abstract classes and methods? | Yes |

We can see that the system has a **Readable Code**, **Meaningful Identifiers** and **names**, Followed the **Style Guide**, tested using **test cases**, Described by **Comments**.

On the other hand, we can see that the system has some **Big Classes**, **Long Methods**, **Dependencies** between components and **compile with warning**.

From the previous checklist, we can:

* Filter out what seems important:
  + Split up God Classes
  + Eliminate Navigation code
* Learn the developers vocabulary
* Functional tests and unit tests convey important information about the functionality of a software system. They can help to verify whether the system is functioning as expected, which is very imported during reengineering

We can notice that the reengineering seems feasible, and while the system main function is to assert, the most important classes and packages are where the system should assert and catch the exceptions and errors:

“org.assertj.core.api”

“org.assertj.core.data”  
“org.assertj.core.error”

“org.assertj.core.api.exception”

We need to investigate the test cases further using the tools of dynamic analysis.

* **Skim the Documentation**

The intent is to Assess the relevance of the documentation “identify those parts of the documentation that might be helpful”

The list below summarizes those aspects of the system that seem interesting for our reengineering project. We will match this list against the documentation.

meanwhile we will make a crude assessment of how up to date the documentation seems:

|  |  |
| --- | --- |
| Installation and build | In [Starting Guide](http://joel-costigliola.github.io/assertj/assertj-core-quick-start.html) page, we can find how we can install the system using Maven. |
| System Features | In [Features](http://joel-costigliola.github.io/assertj/assertj-core-features-highlight.html) - [Tutorial](http://joel-costigliola.github.io/assertj/assertj-core-converting-junit-assertions-to-assertj.html) pages, we can find how to use the system and what features the system have. |
| Design Documentation | Not Exist. |
| Structure Documentation | Table of contents Not Exist. |
| Up to Date | Version Number:  AssertJ fluent assertions **3.8.0** API “Up to Date” |
| System Functionality | In [Overview](http://joel-costigliola.github.io/assertj/assertj-core.html) and other pages we can find some Command Descriptions |
| Important Terms | In [Index](http://joel-costigliola.github.io/assertj/index.html) page, we can find most important terms |

We can see that the documentation will be **useful**, user manual “Tutorial” and installation guide is **clear** and **up-to date**, system functionality **described** using **commands descriptions** and we have the **most important terms**.

* **Do a Mock Installation**

The intent is to Check whether we have the necessary artefacts available by installing the system and recompiling the code.

We **Installed**, **Built** and **Recompiled** the system using Maven, no build or compile errors and exceptions, but warnings are exist.

The system is using JRE System Library [JavaSE-1.8] and Referenced Libraries. No need for a database and network toolkits.

* **Speculate about Design**

Produce hypothetical class / sequence diagrams. Try to link to source code. Key business objects? Design patterns & Architectures?

* **Study the Exceptional Entities**

Look for obvious anomalies. Big classes / methods? Standout metrics? Use to diagnose potential problems to focus on.

* Skim over any issues on SourceForge and Github.
* Skim over recent commit logs.

What does it do well, and what does it do poorly?

* Chat with Maintainers

Using this pattern, we will learn about the historical and political context of AssertJ project through discussions with the people maintaining the system. “Since we can’t talk to the maintainers, we will use GitHub features to address this pattern”

We need to answer the questions below:

1. What was the easiest bug you had to fix during the last month? And what was the most difficult one? How long did it take you to fix each of them? Why was it so easy or so difficult to fix that particular bug?

“the answers will provide us with some concrete examples of maintenance problems we might use in later, more high-level investigation”