Analysis of software testing tools:

How is DesigniteJava, InFusion and Understand helpful in software testing?

Erica Filgueras and Luigi Yerma

I. Introduction

Software testing is used to check if the actual results match the expected results. It doesn't always mean finding bugs in your code. It helps to secure software, improves product Quality and is cost effective. It executes a software component to test various interests such as object oriented metrics, anti-patterns or code smells. It can be either done manually or using tools such as InFusion, DesigniteJava, and Understand.

This paper is aimed towards analysis using various software testing tools through feature comparison as well as analysis of each tools' advantages and drawbacks. This is to understand the effectiveness, strength, and significance of each tool for software testing

II. Methodology

An in-depth exploration through analysis of software functionality, definition, as well as, each software's advantages and drawbacks were used to understand why the three softwares: DesigniteJava, InFusion, and Understand were helpful in software testing.

The method of data collection was obtained by using the three softwares: DesigniteJava, InFusion and Understand. This data was obtained by running 3 projects with 2 versions each through each of the software indicated in the previous statement. By analyzing existing data outputted by each of the software mentioned, we are able to understand this question: *How is Understand, DesigniteJava and Infusion helpful in software testing?*

III. Results

A. Understand

Understand is an IDE that helps users understand their code. It creates a database of relationships in the process of analyzing code. Understand gives the user the overarching architecture of the project. For example, it has the ability to generate graphs such as UML diagrams, analyze metrics for software quality and analyze dependencies to evaluate software architecture.

1. Features

   a. Understand IDE had a user-friendly GUI. The user had the ability to export specified metrics such as:

   - Weighted Method per Class
   - Depth of inheritance tree
   - Number of Children
   - Coupling between objects
   - Lack of Cohesion Methods

The user was able to export data from the project onto a CSV file where you could further analyze the code architecture, through data analysis.

Visualization of the code design your through dynamic diagram presentation was also one of its notable features.

- The most helpful functionality of Understand was mostly found in the Information browser. The information browser showed great detail of the source code architecture, such as:
  - Packages
  - Public, Private and Protected Classes
  - The interfaces implemented by Classes
  - Classes that extend and/or were extended by other classes

2. Advantages and Disadvantages

Advantages:

- The software had a user friendly GUI. Although Understand required a bit of exploration, it was intuitive. The location of the graphs were not hard to find and extracting the metrics were easily done.

Drawbacks:

- There was no summary of the dissection of the source code. To obtain the manual calculation of the metrics indicated above, one had to count methods, classes and variables manually. This was quite lengthy, as one also had to manually analyze the data.

Define abbreviations and acronyms the first time they are used in the text, even after they have been defined in the abstract. Abbreviations such as IEEE, SI, MKS, CGS, sc, dc, and rms do not have to be defined. Do not use abbreviations in the title or heads unless they are unavoidable.

3. Multi project output

Gantt project

Version	WMC	DIT	CBO	LCOM	RFC	NOC
2.8.3	4.385	1.653	3.796	17.677	9.862	0.475
2.8.10	4.385	1.654	3.77	17.987	9.954	0.481

Apache-Tomcat

Version	WMC	DIT	CBO	LCOM	RFC	NOC
4.1.30	2.792	1.642	3.056	29.748	12.788	0.323
4.1.36	2.327	1.562	4.011	30.354	16.148	0.476

Spring-Boot

Version	WMC	DIT	CBO	LCOM	RFC	NOC
1.1.x	4.412	1.239	1.693	16.921	4.347	0.204
2.2.x	3.676	1.186	1.959	13.665	4.611	0.176

B. Infusion

InFusion is a testing software that analyzes code smells such as Architecture level smells, Class level smells, Method level smells as well as Variable level smells. It allows the user to examine the maintainability of the software.

1. Features

- InFusion had a user friendly GUI.
- Software spits out a summary of code smells on the application itself.
- The enumerated code smells were also presented in addition to the summary.

2. Advantages and Drawbacks

Advantages

- User friendly GUI
- Summary of code as well as enumeration of details under each category.

Drawbacks

- No automatic outputs delivered to an output folder. In order to analyze code for software testing, one had to enter the data manually in a separate file.
- The levels of code smells such as Class level smells and Architecture level smells weren't defined explicitly in the software. As a user, one had to manually look up where these smells belonged.
- This software was OS sensitive. Meaning it couldn't run on a non Windows OS. In order to do so, one would have to run it in a virtual environment.
- The software had a limitation on the size of the source code. Source code that was over 100,000 lines couldn't be analyzed.
- Runs only on Windows(VM ware needed)

3. Output

Project names: Gantt project v 2.8.3 and v 2.8.10, Apache Tomcat version 4.1.30 and 4.1.36, Spring Boot version 1.1.x and 2.2.x

Class Level smells

a. God Class

The latest version 2.8.10 has 6 God Classes and the other version analyzed, 2.8.3 has 5 God classes. For apache Version 3.1.30 had 8 God classes while version 3.1.36 had 16. Lastly, Spring Boot outputted that The god class increased from 20 to 30 as version 2.2.x was deployed. Complexity, Encapsulation,Coupling and Cohesion are tied to the God Class. God Classes are the Class level smells as well as architectural smells. These are classes that have too much
responsibility. Since God classes violate the single-responsibility principle they should be split into smaller classes that have their own responsibility. This allows for easier maintainability of the code.

b. Schizophrenic Class

A schizophrenic class is a class that has disjoint sets of public methods that are used by disjoint sets of client classes. The Schizophrenic classes form V2.8.3 started at 11 then but then went up for V2.8.10. The Schizophrenic Class in spring boot has increased from 25 to 40 from versions 1.1.x to 2.2.x. For Apache the god classes The schizophrenic result stayed the same for both version 3.1.36 and version 3.1.30.All of the categories have this type of code smell. It has gotten worse if we look at it starting with the initial version, this also because the code base is getting bigger.

c. The presence of Data Classes

The Data classes fluctuate from 33 starting with V2.8.3. The value 32 with the latest version. In spring boot, the presence of data classes has increased from 20 from version 1.1.x to 25 in version 2.2.x. In apache it maintained at a level of for both version 3.1.30 and version 3.1.36. Data classes are only found in encapsulation and cohesion smells.A data class refers to a class that only contains methods that are getters and setters. These classes don't really have additional functionality and can't operate on their own independently.Data classes prevents lower coupling between the classes. Lower coupling helps to easily maintain classes.

Method level smells

a. Blob Operation

The Blob operation has decreased from 8 to 7, if we start at version 2.8.3 to 2.8.10. In Apache the blob increased from 10 in version 3.1.30 to to 11 in 3.1.36. Lastly in springboot, the number of blobs increased from 25 in version 1.1.x . This however, increased to 30 when version 2.2.x was deployed. Blob Operations are similar to God classes in a way. They grow larger and larger overtime, becoming more expensive to maintain. Complexity and coupling are the onlyones associated with this code smell.

b. External Duplication

The external Duplication for all 4 versions has maintained at 6. External Duplication is associated with Encapsulation and Coupling.

c. Feature Envy

Feature envy has maintained at a level of 5 and is present in Complexity,Encapsulation, Coupling and Cohesion. For Apache, Version 3.1.30 had a feature envy of 5 but dropped to 4 on version 3.1.36. Springboot version 1.1.x started 10 and increased to 25 for version 2.2.x. Feature envy is a code smell describing when an object accesses fields of another object. Feature envy breaks encapsulation.This is also a coupling code smell, it couples two objects together inappropriately. Since feature envy increases coupling, which increases the likelihood of introducing bugs.

Variable level smells:

a. Data Clumps

Data Clumps have increased from 7 to 11, starting from version 2.8.3 to 2.8.10. For springboot, version 1.1.x there 20 and it increased by 25 when 2.2.x was deployed. For Apache, Data Clumps in version 3.1.30 is 10 and increased to 20 in version 3.1.36. Data Clumps is the name given to a group of variables passed around together in a lot of theparts of the program. Because it has increased, the developers may have not been aware of this phenomenon or did not put their code through testing before releasing.

Gantt project v 2.8.3 and 2.8.10

Version	Complexity deficit	Encapsulation deficit	Coupling deficit	Inheritance deficit	Cohesion deficit	Quality deficit index
2.8.3	6.6	10.5	10.6	1.5	5.2	6.9
2.8.10	7	10.3	11.1	1.9	5	7.1

Version	Complexity design flaws	Encapsulation design flaws	Coupling design flaws	Inheritance design flaws	Cohesion design flaws	Total design flaws
2.8.3	45	67	77	14	54	257
2.8.10	51	72	78	16	55	272

Apache Tomcat

Version	Complexity deficit	Encapsulation deficit	Coupling deficit	Inheritance deficit	Cohesion deficit	Quality Deficit Index	Loc
4.1.30	16.5	12.2	15.2	2	5.3	10.3	237667
4.1.36	10.5	7	19.4	7.7	4	13.9	309294

Version	Complexity design flaws	Encapsulation design flaws	Coupling design flaws	Inheritance design flaws	Cohesion design flaws	Total design flaws
4.1.30	426	290	417	41	163	481
4.1.36	592	418	606	118	228	680

Spring-Boot

Version	Complexity deficit	Encapsulation deficit	Coupling deficit	Inheritance deficit	Cohesion deficit	Quality Deficit Index	Loc
1.1.x	10.5	7	19.4	7.7	4	9.7	121735
2.2.x	6.7	6.7	14.6	5.8	3.5	7.5	479737

Version	Complexity deficit	Encapsulation deficit	Coupling deficit	Inheritance deficit	Cohesion deficit	Quality Deficit Index	Loc
1.1.x	10.5	7	19.4	7.7	4	9.7	121735
2.2.x	6.7	6.7	14.6	5.8	3.5	7.5	479737

C. DesigniteJava

Designite Java is a quality assessment tool that allows users to detect code smells similar to InFusion. It gives the summary and details of various architecture level smells, design smells as well as implementation smells. DesigniteJava also computes commonly used Object oriented metrics.

Because of this, DesigniteJava reduces technical debt and improves maintainability of the software.

1. Features

    - There was no GUI, one had to run the software on the terminal or command prompt.
    - It does the analysis for the user and outputs it in a CSV file in the specified output folder.
    - DesigniteJava gives the user a summary and It splits up the code smells into levels.
        It was able to output additional code smells. Including but not limited to the following:
        - Imperative abstraction
        - Multifaceted abstraction
        - Unnecessary abstraction
        - Unutilized abstraction

2. Advantages and Drawbacks

Advantages

- Gives a summary of the analysis.
- Specifies the levels of smells such as architecture, design, and implementation smells. This is helpful especially if the developer is looking for a quick reference to maintain code.
- Spits out the output to an output folder that the user provides.
- Runs on all OSs.

Drawbacks

- There was no GUI. One had to run the software on the terminal or command prompt. Although, the user also had the option of adding DesigniteJava as a plug in for an IDE they are already using, such as IntelliJ.
- This software wasn't user friendly when it came to running it on the terminal. One had to have a bit of understanding on how the terminal works. This included basic understanding on terminal commands.

3. Multi project output

Project name: gantt project v 2.8.3

<u>--Analysis summary--</u>

Total LOC analyzed: 67683

- Number of packages: 87
- Number of classes: 1033
- Number of methods: 9477

-Total architecture smell instances detected-

- Cyclic dependency: 132
- God component: 6
- Ambiguous interface: 0
- Feature concentration: 17
- Unstable dependency: 28
- Scattered functionality: 0
- Dense structure: 1

-Total design smell instances detected-

- Imperative abstraction: 0
- Multifaceted abstraction: 2
- Unnecessary abstraction: 11
- Unutilized abstraction: 156
- Feature envy: 28
- Deficient encapsulation: 84
- Unexploited encapsulation: 1
- Broken modularization: 5
- Cyclically-dependent modularization: 62
- Hub-like modularization: 2
- Insufficient modularization: 67
- Broken hierarchy: 90
- Cyclic hierarchy: 2
- Deep hierarchy: 0
- Missing hierarchy: 1
- Multipath hierarchy: 2
- Rebellious hierarchy: 5
- Wide hierarchy: 2

-Total implementation smell instances detected-

- Abstract function call from constructor: 2
- Complex conditional: 62
- Complex method: 96
- Empty catch clause: 20
- Long identifier: 37
- Long method: 17
- Long parameter list: 109
- Long statement: 824
- Magic number: 1806
- Missing default: 36

II.Project name: gantt project v 2.8.10

<u>--Analysis summary--</u>

Total LOC analyzed: 64528

- Number of packages: 86
- Number of classes: 1006
- Number of methods: 9091

-Total architecture smell instances detected-

- Cyclic dependency: 132

- God component: 6
- Ambiguous interface: 0
- Feature concentration: 19
- Unstable dependency: 25
- Scattered functionality: 0
- Dense structure: 1

-Total design smell instances detected-

- Imperative abstraction: 0
- Multifaceted abstraction: 2
- Unnecessary abstraction: 11
- Unutilized abstraction: 154
- Feature envy: 29
- Deficient encapsulation: 78
- Unexploited encapsulation: 1

III. Project name: Spring-Boot v 1.1.x

--Analysis summary--

Total Lines Of Code Analyzed: 11,650

- Number of Packages: 23
- Number of classes: 170
- Number of Methods: 1252

-Total design smell instances detected-

- Imperative abstraction: 0
- Unnecessary Abstraction 0
- Feature Envy: 0
- Unexploited encapsulation: 1
- Cyclically-dependent modularization: 1
- Insufficient Modularization: 9
- Cyclic hierarchy: 2
- Missing hierarchy: 1
- Rebellious hierarchy: 0
- Multifaceted abstraction: 0
- Unutilized abstraction: 44
- Deficient encapsulation: 10
- Broken Modularization: 0
- Hub-like modularization: 0
- Broken hierarchy: 17
- Deep hierarchy:Multipath hierarchy: 0
- Wide hierarchy : 0

- Total Implementation smell instances detected-

- Abstract function call from constructor: 0
- Complex method: 10
- Long Identifiers: 2
- Long parameter list: 2
- Magic number: 95
- Complex conditional: 11
- Empty catch clause: 33
- Long Method: 0
- Long statement 92
- Missing default: 1

IV. Project name: Spring-Boot v 2.2.x

--Analysis summary--

Total Lines Of Code analyzed: 532,531

- Lines Of Code Analyze: 35,406
- Number of Packages: 71
- Number of classes: 579
- Number of Methods: 4124

-Total architecture smell instances detected-

- Cyclic dependency: 0
- Ambiguous interface: 1
- Unstable dependency: 2
- Dense structure: 0
- God component: 2
- Feature concentration: 14
- Scattered functionality: 0
- Total design smell instances detected
- Imperative abstraction: 0
- Unnecessary Abstraction 7
- Feature Envy: 1
- Unexploited encapsulation: 1
- Cyclically-dependent modularization: 8
- Insufficient Modularization: 19
- Cyclic hierarchy: 5
- Missing hierarchy: 3
- Rebellious hierarchy: 0
- Multifaceted abstraction: 0
- Unutilized abstraction: 140
- Deficient encapsulation: 24
- Broken Modularization: 0
- Hub-like modularization: 0
- Broken hierarchy: 46
- Deep hierarchy: 0
- Multipath hierarchy: 0
- Wide hierarchy : 0

-Total Implementation smell instances detected-

- Abstract function call from constructor: 0
- Complex method: 13
- Long Identifiers: 34
- Long parameter list: 38
- Magic number: 106

- Complex conditional: 24
- Empty catch clause: 59
- Long Method: 0
- Long statement: 297
- Missing default: 3
- Broken modularization: 4
- Cyclically-dependent modularization: 55 Hub-like modularization: 2
- Insufficient modularization: 63
- Broken hierarchy: 90
- Cyclic hierarchy: 2
- Deep hierarchy: 0
- Missing hierarchy: 1
- Multipath hierarchy: 2
- Rebellious hierarchy: 4
- Wide hierarchy: 2

-Total implementation smell instances detected-

- Abstract function call from constructor: 2
- Complex conditional: 52
- Complex method: 86
- Empty catch clause: 16
- Long identifier: 34
- Long method: 14
- Long parameter list: 101
- Long statement: 749
- Magic number: 1717
- Missing default: 34

IV. CONCLUSION

Although Understand, InFusion and DesigniteJava were different, they were all essential in software testing because they allowed the user to test different parts of the code .Understand and Infusion had user friendly GUIs while DesiginiteJava had none. That being said, using Infusion and Understand was easier to navigate because of their user friendly GUIs, DesigniteJava had a bit of a learning curve.

It is also notable to mention that these three software used had different functionalities. Understand measured code architecture quality, InFusion as well as DesiginiteJava assessed software quality through the examination of code smells. The three software: Understand, InFusion, and DesiginiteJava all had something similar, they all help developers increase the maintainability of their code.