# NOTE: THIS IS A SAMPLE WRITEUP – PLEASE REFER TO THE INSTRUCTIONS IN THE ASSIGNMENT DOCUMENT FOR THE METRICS YOU NEED TO ANALYZE AND ADDITIONAL TASKS YOU NEED TO PERFORM

Introduction

WordPress is a leading open source content management system used primarily to host Blogs. The project that was analyzed, using Understand, is the Android app that allows users to manage their websites. The app is hosted on the Google Play Store [[1]](#_bookmark0) and presently has a 4-star rating with over 5,000,000 installs.

The source code for the app is hosted and maintained on GitHub [[2]](#_bookmark0). The source code for each release is maintained as a tag. For this assignment, a total of 11 releases of the app were analyzed. The releases range from 2011 (the first available release) to 2016 (the most recent release when running the analysis). Each year between 2011 and 2016 has at least one release included in the analysis.

# Analysis

The table below shows the metrics that were captured per release. The metrics include complexity, volume, and object-oriented (CK) metrics.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Release** | **Release Date** | **Average LCOM** | **Average DIT** | **Average IFANIN** | **Average CBO** | **Average NOC** | **Average RFC** | **Average NIM** | **Average NIV** | **Average WMC** | **Total Classes** | **Ratio Comment**  **/Code** | **Total Lines of**  **Code** | **Total Files** | **Total Functions** | **Cyclomatic Complexity** |
| 1.4.0 | 2011-04-13 | 12.38095 | 1.207283 | 1.722689 | 1.327731 | 0.028011 | 2.439776 | 2.078431 | 1.123249 | 2.310924 | 360 | 0.24 | 15889 | 55 | 838 | 2.84009547 |
| 1.4.1 | 2011-04-20 | 12.38095 | 1.207283 | 1.722689 | 1.327731 | 0.028011 | 2.439776 | 2.078431 | 1.123249 | 2.310924 | 360 | 0.24 | 15925 | 55 | 838 | 2.85083532 |
| 2.0 | 2011-12-18 | 15.86486 | 1.258258 | 1.615616 | 2.735736 | 0.024024 | 3.276276 | 2.807808 | 1.483483 | 3.156156 | 348 | 0.23 | 18354 | 72 | 1076 | 2.59944238 |
| 2.1 | 2012-05-04 | 15.16316 | 1.271053 | 1.671053 | 2.115789 | 0.034211 | 3.252632 | 2.807895 | 1.444737 | 3.126316 | 407 | 0.21 | 19792 | 80 | 1225 | 2.62367347 |
| 2.5 | 2013-11-08 | 20.64739 | 1.393512 | 1.637518 | 2.500705 | 0.100141 | 5.390691 | 3.895628 | 1.461213 | 4.244006 | 767 | 0.12 | 33991 | 189 | 3080 | 2.00064558 |
| 2.8.1 | 2014-05-08 | 23.65165 | 1.357357 | 1.580581 | 3.432432 | 0.088088 | 5.753754 | 3.946947 | 1.664665 | 4.565566 | 1077 | 0.12 | 50073 | 350 | 4620 | 2.07656316 |
| 3.0 | 2014-08-04 | 24.10536 | 1.353448 | 1.584291 | 3.498084 | 0.078544 | 5.743295 | 3.939655 | 1.639847 | 4.613985 | 1121 | 0.11 | 53767 | 372 | 4862 | 2.10214401 |
| 4.0 | 2015-05-13 | 24.68534 | 1.328473 | 1.585572 | 3.579432 | 0.059094 | 5.651573 | 4.008442 | 1.824252 | 4.643899 | 1410 | 0.11 | 67310 | 470 | 6100 | 2.09677939 |
| 5.0 | 2016-02-09 | 24.80014 | 1.324774 | 1.594726 | 3.641915 | 0.049271 | 5.802221 | 4.145732 | 1.878557 | 4.870923 | 1535 | 0.1 | 79022 | 532 | 7062 | 2.12649763 |
| 6.0 | 2016-10-24 | 23.18931 | 1.322911 | 1.607163 | 3.660603 | 0.077317 | 5.730529 | 4.073906 | 1.820921 | 4.785105 | 1879 | 0.1 | 93383 | 598 | 8493 | 2.10082281 |
| 6.5 | 2017-01-19 | 22.94098 | 1.314754 | 1.607104 | 3.712568 | 0.073224 | 5.620219 | 4.036066 | 1.836066 | 4.761202 | 1954 | 0.1 | 97205 | 611 | 8786 | 2.11461831 |

*Figure 1: Release Metrics*

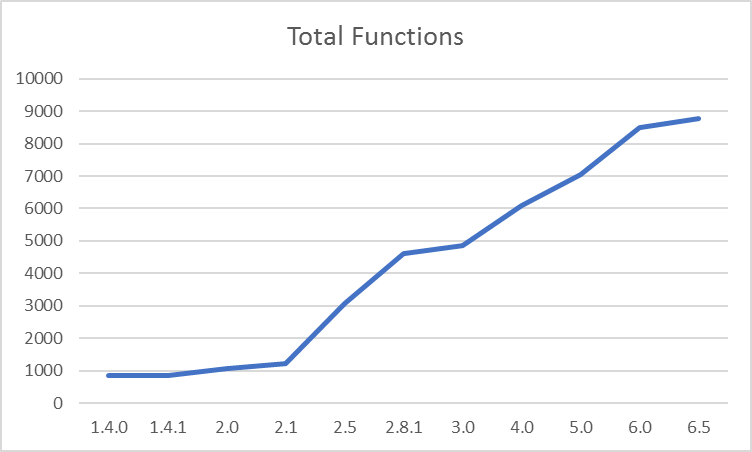
## Complexity Metrics

Interestingly, the *Cyclomatic Complexity* of the project shows a downward trend from the initial release. Usually, during the evolution of a software application, more functionality is added to the application, which most often increases complexity. However, this downward trend indicates that the development team is continually refactoring the project to simplify methods. This not only reduces complexity but also enables better unit testing.

*Note: unit test coverage was not captured in this assignment.*

## Volume Metrics

|  |  |
| --- | --- |
|  |  |
|  |  |



While the *Lines of Code* increase, the *Comment to Code Ratio* shows a downward trend. This indicates that developers are not documenting their code, which could impact maintenance. However, the effectiveness of the *Comment to Code Ratio* metric is debatable as there is a school of thought that code should be self-explanatory and comments should be sparingly used.

The *Total Lines of Code*, *Total Files*, *Total Classes,* and *Total Functions* graphs show an upward trend. This is to be expected for an evolving application as more functionality is added. When all the releases were compared, it was noticed that, between releases 2.1 & 2.8.1, there was a steep rise in the number of classes, functions, and files added to the project. This could be due to the gap between the releases; perhaps if more releases between 2.1 & 2.8.1 were included, it might show a steady increase instead of a steep increase. On a positive note, the cyclomatic complexity has a 0.6 reduction.

The graphs indicate that the project is growing in terms of functionality with each release. With this continued growth, the developers need to keep in mind the project's complexity and the effort required to maintain it.

Object Oriented Metrics

|  |  |
| --- | --- |
| The LCOM graph shows an upward trend; this is an indication of an increase in complexity. An approach to resolve this is to split the classes, with high values, into multiple sub-classes.  The last three releases show a decrease in the LCOM metric. This could possibly be related to the decrease of NIM and NIV in the same releases. | The amount of coupling between classes increases from release 1.4.1 to 2.0. Even though the *Total Classes* graph shows a reduction from  1.4.1 to 2.0, the developers would have increased the number of links between existing classes.  However, it seems that the developers refactored the code at release  2.1 to improve the coupling. An increase in classes could have helped reduce the CBO value (possibly by moving coupled methods to a single class).  From release 2.1, coupling starts increasing again. To improve (reduce  the value), developers must promote modularity in the code. |

|  |  |
| --- | --- |
| The WMC calculated by Understand is the number of local (not inherited) methods [3]. The upward trend of the WMC graph indicates a rise in the number of methods in the class, which indicates that the maintainability of the class increases. Developers can refactor the code to promote reuse to reduce the value. | There is a significant jump from release 2.1 to 2.5. A reason for this could be the increase in NIM and lines of code in the same period. The last three releases show a decrease in the RFC metric, which again corresponds to a drop in the NIM metric. There is an increase in the number of classes for this period as well; this could also have contributed to the reduction of RFC, as related methods would have  moved out to other classes. |

|  |  |
| --- | --- |
| The DIT metric peaks at release 2.5, but then continues a downward trend. Refactoring from release 2.5 would have reduced the depth of inheritance. The lower the value reduces the complexity. However, small values could also indicate less reusability. | The sharp increase from release 2.1 to 2.5 might be related to the increases of total classes in the same period. The greater the value, would indicate more reuse, but it will also increase improper abstraction of the parent class. Developers would need to settle for a tradeoff. |
|  |  |

|  |  |
| --- | --- |
| An upward trend in this graph indicates that more functionality is added to the application. However, the developers need to ensure that the number of methods being added does not negatively impact other metrics such as LCOM,  WMC, etc. | An upward trend in this graph indicates that more functionality is added to the application. |

# References

1. WordPress - Android Apps on Google Play. Available at: <https://play.google.com/store/apps/details?id=org.wordpress.android&hl=en> [Accessed February 02, 2017]
2. GitHub - wordpress-mobile. Available at: <https://github.com/wordpress-mobile/WordPress-Android/> [Accessed February 02, 2017]
3. SciTools – Metrics. Available at: <https://scitools.com/support/metrics_list#CountDeclMethod> [Accessed February 02, 2017]