ICS 691E: Software Quality Assurance (Fall 2022)
Assignment 2

Introduction

ZXing ("Zebra Crossing") is an open-source, multi-format 1D/2D barcode image processing library implemented in Java, with ports to other languages [1]. The project that was analyzed, using Understand [2], is the Barcode Scanner application that appears on the Google Play store. It was originally authored by Google developers in 2009 and was one of the first applications in the store. It was downloaded over 100 million times, has close to 650k reviews, and a 3.9-star rating. An application with the same name briefly appeared on the Google Play store but turned out to be installing malicious code on users' devices. The ZXing doppelganger was quickly removed but the mix-up may be what led to the less-than-stellar review rating.

The source code for the app is hosted and maintained on GitHub [2]. The source code for each release is maintained as a tag. In total we analyzed 6 releases of the app. The releases range from 2014 (when the project was moved to GitHub) to 2022 (the most recent release when running the analysis). Most of the releases were minor version releases and the last three releases analyzed each spanned about three years between releases. The first two releases (2.3.0 and 3.0.0) analyzed occurred only one month apart but were included to show the shift between major versions 2.0 and 3.0.

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 Cyclomatic Complexity	Average Comment /Code	Average DIT	Average NIM	Average NIV	Average Local Methods	Average CBO	Average LCOM	Average NL	Average LOC	Average NSC	Average Number of functions	Total Classes	Total Lines of Code	Total Files	Total Functions
2.3.0	2014-01-18	2.57	0.28	1.69	5.91	2.39	7.64	6.88	18.23	57.00	40.25	20.51	5.91	587	64,884	476	2,815
3.0.0	2014-02-28	2.51	0.29	1.72	6.10	2.45	7.90	6.94	18.96	56.93	40.14	20.40	4.49	555	75,695	643	2,886
3.2.0	2015-02-15	2.58	0.29	1.72	6.17	2.46	8.01	7.02	19.11	57.26	40.50	20.37	6.10	564	65,399	471	2,871
3.3.0	2016-09-16	2.55	0.30	1.72	6.08	2.49	7.91	7.00	18.81	57.26	40.35	20.19	6.00	582	66,692	485	2,910
3.4.0	2019-05-17	2.56	0.31	1.74	6.12	2.43	7.98	7.13	18.68	57.78	40.60	20.30	5.87	561	67,682	499	2,931
3.5.0	2022-05-01	2.55	0.33	1.72	6.46	2.53	8.44	7.17	19.23	40.77	58.02	20.41	6.27	589	72,676	513	3,215

Figure 1: Release metrics as reported by the Understand tool.

Complexity Metrics

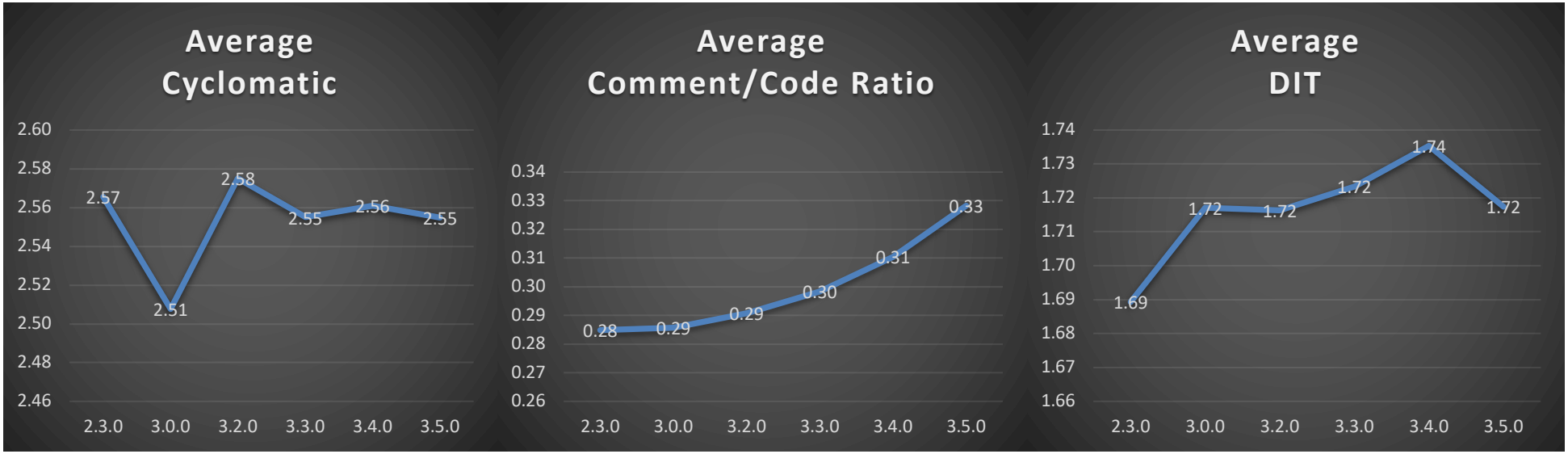

Figure 2: Cyclomatic Complexity, Comment to Code Ratio, and Max Inheritance Tree averages (left to right) as reported by the Understand tool. The code and functionality increased but the complexity remained similar.

The *Cyclomatic Complexity* of the project shows a sharp decline from versions 2.3.0 and 3.0.0, followed by a climb from the initial major release to the second minor release (3.0.0 to 3.2.0). After this, the complexity trends downward. The decrease between major versions is probably due to refactoring the code as seen between versions 3.2.0 to 3.5.0. However, after the new major version is released, the sudden climb might be due to adding features or fixing unforeseen bugs.

In the second graph, the *Ratio of Comment to Code* increases continuously across all versions. Good comments make the code easier to understand and easier to maintain. Too few comments can lead to confusions while too many comments may indicate poor structure or naming conventions. The consistency for ZXing's ratio likely shows the code is well commented from version 2.3.0 and got even better by the most recent release.

The *Max Inheritance Tree* (DIT) shows the maximum length from a class to its root. The deeper the tree, the greater the design complexity. The graph above shows that the complexity increased slightly between major versions, but the developers appeared to try and keep the code relatively simple.
In summary, the complexity metrics indicate the ZXing codebase is not overly complex and is probably clean and well documented.

Object Oriented Metrics

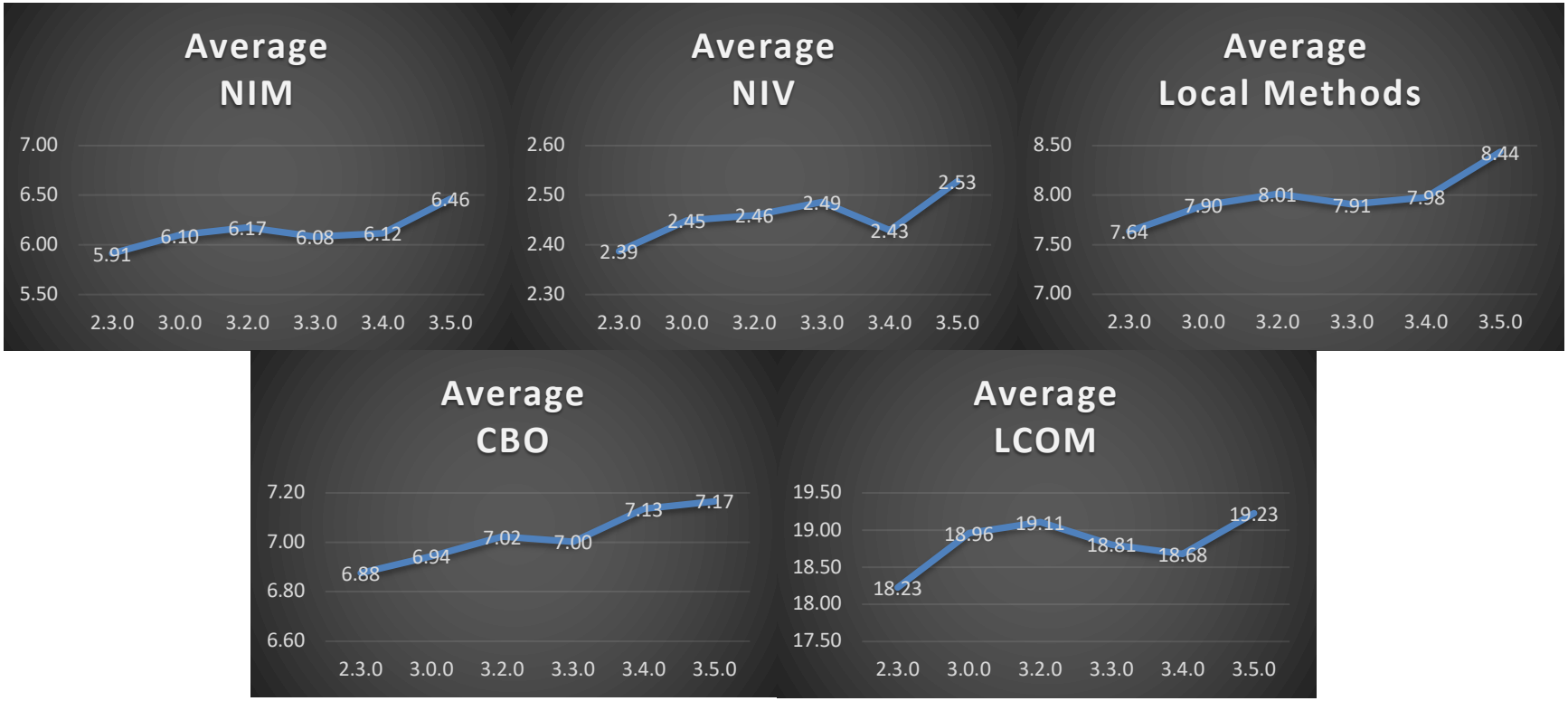

Figure 3: Average number of Instance Methods, Average number of Instance Variables, Average number of Local Methods, Average Class Coupling, and Average Lack of Cohesion Methods (left-to-right) as reported by the Understand tool. The graphs show a small increase in complexity over the last 8 years (versions 2.3.0 to 3.5.0).

The second set of graphs all focus on metrics tied to Object-oriented programming. The graphs appear to be correlated as each has a similar sinusoidal-like curve, trending upwards apart from a shallow drop around release versions 3.3.0 and 3.4.0. The top row of graphs includes the averages between versions for *Number of Instance Methods* (NIM), *Number of Instance Variables* (NIV), and *Average Number of Local Methods* and the second row of graphs show *Class Coupling* (CBO) and *Lack of Cohesion in Methods* (LCOM).

These metrics are often used to help detect anti-patterns and design smells such as Blog or Spaghetti Code. Looking at the CBO alone, it seems the application is gaining in complexity because increased coupling could mean classes are less modular, or more difficult to reuse in other applications. However, the CBO only did not increase significantly, and the application did not appear to suffer from lack of cohesion. If the LCOM increases dramatically, it may indicate the need to split classes into sub-classes. In doing so, the average number of local methods will likely decrease as well.

Volume Metrics

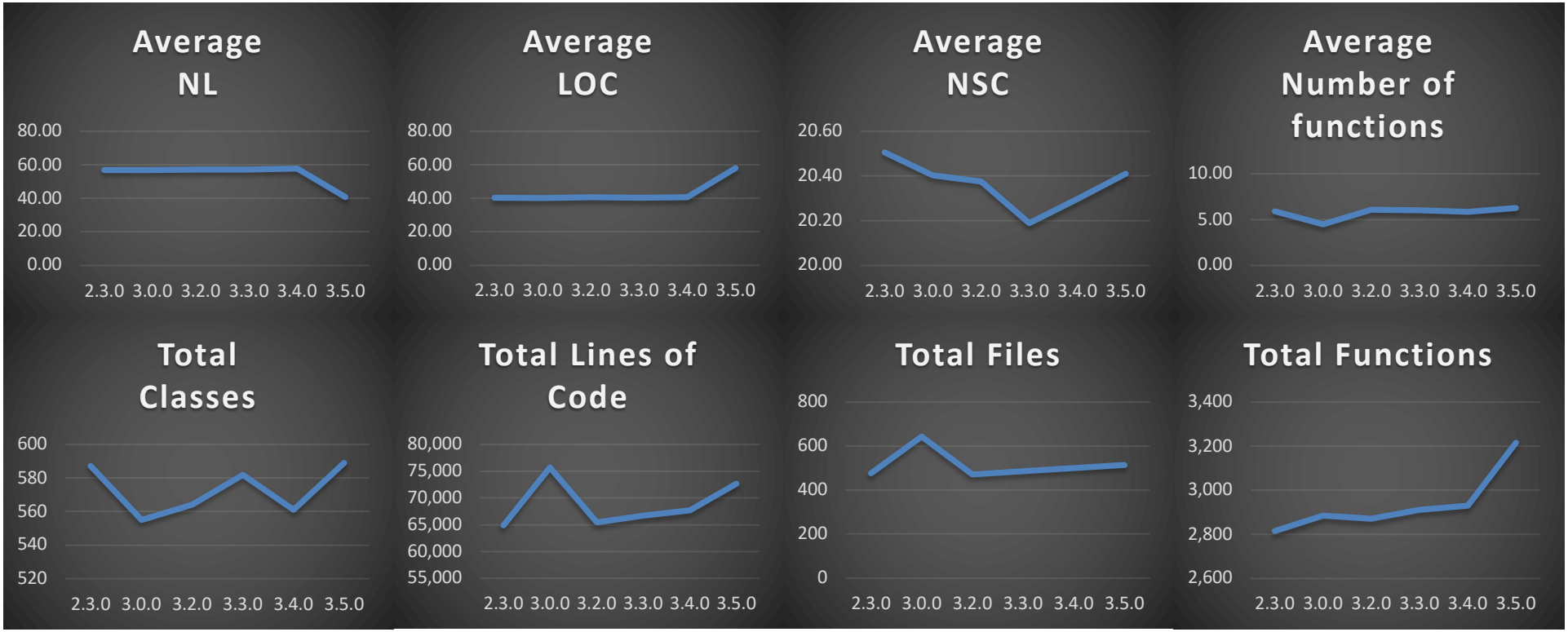

Figure 4: The eight graphs above each have minimal movement which implies the ZXing code is quite stable apart from adding new functionality or methods to keep up with modern architecture requirements.

The average *Line Count* remained consistent over the years and then suddenly dropped slightly between the 2019 (3.4.0) and 2022 (3.5.0) releases. However, the *Average Lines of Code* found in the most recent version increased. This is likely due to removing dead code, code comments, and blank lines, i.e., cleaning up the codebase. The *Number of Semicolons* decreased in the until the 3.3.0 release and then began to climb again. The next metric, *Average Number of Functions* also shows a tiny uptick toward the final release but is difficult to characterize its overall impact due to is relatively tiny variability footprint. The chart for *Total Classes* shows that when the product first arrived on Github in 2014 with the 2.3.0 release, it had close to 600 classes. Between that release and the 3.0.0 release, we can see a sharp decline in the number of classes and again between the 3.3.0 and 3.4.0 releases. It then begins to climb again. Too many classes can lead to more complexity, especially if they are dependent on one another or have high coupling. The *Total Lines of Code* and *Total Functions* both increased at a higher rate than the *Total Files* which suggests the developers wanted to keep the files to a smaller number while still introducing more functionality, which inevitably leads to more lines of code. It seems there was a larger increase in functionality from version 2.3.0 to 3.5.0 but using the Mann-Whitney U-Test (Figure 5), it appears the changes were not statistically significant from one release to the next.

P-Values (Mann-Whitney U-Test)

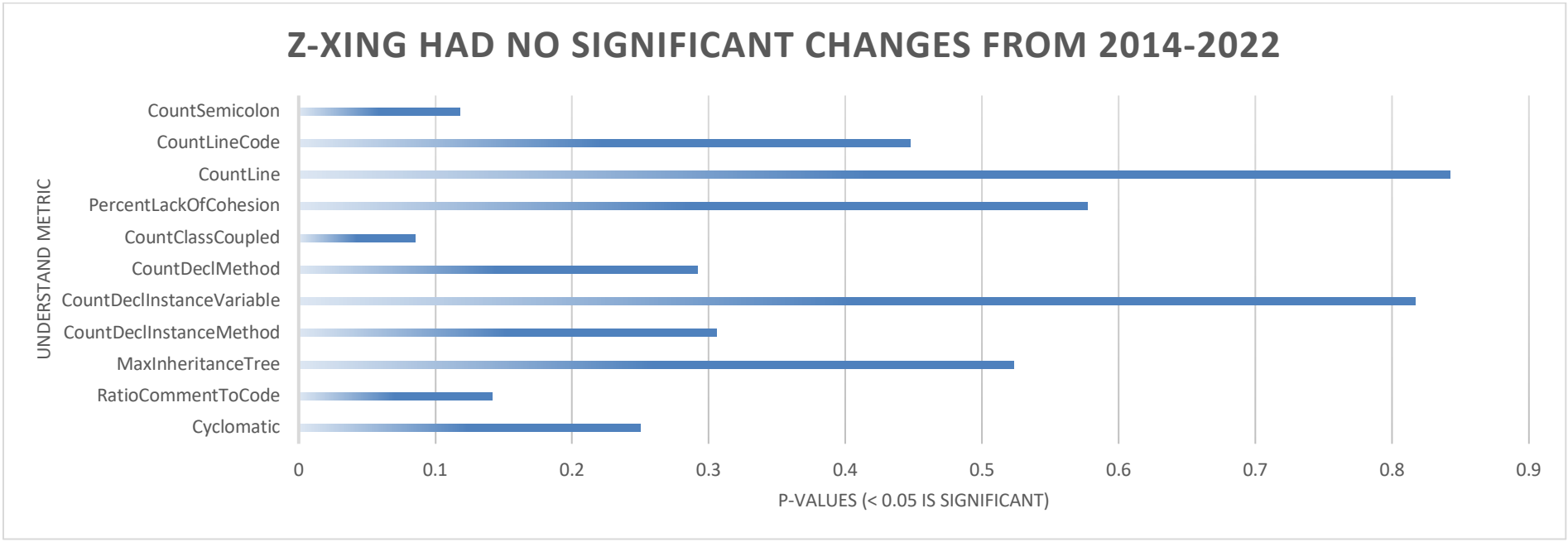

Figure 5: The graph above shows P-Value from a series of Mann-Whitney U-Tests comparing the first available and latest release versions. The largest deviation from the mean occurred with the CBO metric, indicating the code may have become more or less "coupled" or complex. In other words, we cannot reject the null hypothesis that the two versions of software are from the same (or similar) distribution. The software did not change much in 8 years.

References

[1] Barcode Scanner - Android Apps on Google Play. Available at: https://play.google.com/store/apps/details?id=com.google.zxing.client.android [Accessed Oct 30, 2022]

[2] GitHub - ZXing Project. Available at: https://github.com/zxing [Accessed Oct 29, 2022]

[3] SciTools – Metrics. Available at: https://support.scitools.com/support/solutions/articles/70000582223-what-metrics-does-understand-have- [Accessed February 02, 2017]