Using Object Oriented Software Metrics for Mobile Application Development

GREGOR JOŠT, JERNEJ HUBER AND MARJAN HERIČKO, University of Maribor

Developing and maintaining software for multiple platforms can be challenging. So, scheduling and budget planning, cost estimation, software debugging, software performance optimization etc. is required. In traditional software, this can be achieved using software metrics. The objective of our article was to examine whether the traditional software metrics are appropriate for measuring the mobile applications' source code. To achieve this goal, a small-scale application was developed across three different platforms (Android, iOS and Windows Phone). The code was then evaluated, using the traditional software metrics. After applying the metrics and analysing the code, we obtained comparable results, regardless of the platform. If we aggregate the results, we can argue that traditional software metrics can be used for mobile applications' source code as well. However, further analysis is required, in light of a more complex mobile application.

Key Words and Phrases: Software product metrics, object-oriented metrics, mobile development, multi-platform mobile development

1. INTRODUCTION

The future of computing is moving towards mobile devices. It is suggested that by the end of 2013, people will purchase 1.2 billion mobile devices. As such, they will replace PC's as the most common method for accessing the Internet [Gordon 2013]. Currently, the two most popular platforms are Google's Android and Apple's iOS. There are, however, several other platforms, which add to the market fragmentation. Among those, Windows Phone is considered to be the most promising. Besides, mobile application development has also become an important area of scientific studies.

However, developers face several challenges when developing mobile applications. First, the range of mobile platforms requires the knowledge and experience with the different programming languages as well as software development kits (hereinafter referred to as SDK). This consequentially results in a duplication of development effort and maintenance costs [Nebeling et al. 2013]. Second, developers need to consider the physical limitations of mobile devices. Third, due to the low selling price of the mobile applications, developers need to reduce the development costs.

On the other hand, if we want to increase reusability, decompose problem into several easily understood segments or enable the future modifications, object-oriented programming metrics (hereinafter referred to as OO metrics) need to be considered. They represent a dependable guidelines that may help ensure good programming practices and write a reliable code. Such metrics have been applied and validated numerous times for the development of traditional software. In regard to the development of mobile applications, there is a lack of studies that would investigate the measurement of mobile application's software code. However, several attempts were already made in applying existing OO metrics to mobile applications, but were not empirically validated. Thus, based on the previous research we conducted the following hypothesis:

H_{01}: *Using the object-oriented metrics for mobile application does not differ from using object-oriented metrics for desktop or web applications.*

H_{A01}: *Using the object-oriented metrics for mobile application differs from using object-oriented metrics for desktop or web applications.*

Author's address: G. Jošt, UM-FERI, Smetanova ulica 17, 2000 Maribor; email: gregor.jost@uni-mb.si; J. Huber, UM-FERI, Smetanova ulica 17, 2000 Maribor; email: jernej.huber@uni-mb.si; M. Heričko, UM-FERI, Smetanova ulica 17, 2000 Maribor; email: marjan.hericko@uni-mb.si

In: Z. Budimac (ed.): Proceedings of the 2nd Workshop of Software Quality Analysis, Monitoring, Improvement, and Applications (SQAMIA), Novi Sad, Serbia, 15.-17.9.2013, published at http://ceur-ws.org

However, there are also other mobile specific aspects that need to be considered. The number of users of mobile applications is usually dependent on both the quality of the application itself and the targeted platforms, for which the application was developed. Ideally, it would be best to develop each application for all platforms simultaneously without any adjustment of the source code. To this end, many tools for multi-platform development have been introduced, which support the "build once, deploy everywhere" ideology, but are not yet perfect. Mobile applications, developed in native languages are still preferred among the end users. But, as already stated, developing a mobile application for several platforms is a time and cost consuming process. The development of mobile applications for multiple platforms requires different environments and knowledge of specific programming languages, depending on the platform. Each platform has its own set of functionalities, rules and requirements, thereby limiting the interoperability of applications and their simultaneous development. Thus, we investigated if and how the results of metric analysis across different platforms differ in the scope of functionally equivalent application. In this light, the following hypothesis was formed:

H_{02}: *The results of metric analysis of functionally equivalent applications across different platforms do not differ.*

H_{A02}: *The results of metric analysis of functionally equivalent applications across different platforms differ.*

In order to reject or fail to reject both null hypothesis, with the help of colleagues, we implemented an equivalent applications for each of the chosen mobile platforms. The research included: 1) the development of said applications, 2) a comparison of the development between the mobile platforms and 3) metric analysis of the mobile application. We organized our research, and consequentially this article, as follows.

The second section introduces the research foundation. The third section reviews previous work regarding the challenges of developing applications for multiple platforms as well as metrics for measuring the mobile applications' source code, with the focus on the latter. The forth section presents the details of the application development on different platforms. We defined the basic functionalities, abstract design of the user interface and summarized the business logic. The fifth section presents the results of the metric analysis of the applications, whereas the last section interprets the results, reviews the limitations and implications, as well as defines future planned activities within the article's topics.

2. RESEARCH FOUNDATION

In general, software product metrics can be divided into two categories: 1) internal, which are seen usually only to development team (e.g., size, complexity and style metrics) and 2) external metrics (product non-reliability, functionality, performance and usability metrics), which assess properties, visible to the users of a product [North Carolina State University 2005]. In this light, our research is focused on OO metrics, which are one of the internal product metrics that can be used to evaluate and predict the quality of the software.

There is a vast body of empirical results, which support the theoretical validity of such metrics. The validation of these metrics requires a demonstration that the metrics measure the right concept (e.g. a coupling metric measures actual coupling). It is also important that the metrics are associated with an important external metrics, such as reliability or fault-proneness.

OO metrics are generally accepted as software quality indicator for desktop software. They provide automated and objective way to get concrete information regarding the source code [Franke and Weise 2011]. Such analysis with the purpose of quality evaluation continues to increase in popularity in wide variety of application domains.

Recently, mobile devices experienced a significant gain in performance and the development of mobile applications is getting more similar to desktop ones in light of the programming concepts and techniques.

Regarding the metrics for mobile applications, several suggestions were already made. OO metrics like McCabe Cyclomatic Complexity, Weighted Methods per Class or Lack of Cohesion in Methods have already been used to analyze the source code of software for mobile devices [Franke and Weise 2011].

On the other hand, external metrics, such as flexibility, which is a quality requirement for software, are also very important in the mobile domain. Environment of mobile devices can change often and suddenly (loss of mobile signal, different light conditions, etc.). Another important quality attribute is adaptability, which represents the ability of software to adapt itself to new underlying systems (e.g. usage of Galileo instead of GPS). Data persistence is also a quality requirement factor, since mobile devices can run out of memory, in which case the applications should be able to properly store their states. Because quality requirements for mobile software differ from their counterparts in desktop domain, the derivation of these qualities to source code metrics must also be done in a different way. So, metrics have to be evaluated in the mobile world and weighted correspondingly before their usage.

3. RELATED WORK

3.1 Challenges and comparison of the development of mobile applications for multiple platforms

When developing mobile applications in general, we are facing distinctive challenges. [Dehlinger and Dixon 2011] have identified the following:

1. Different sizes and resolutions of displays should be considered
2. Before the development of applications it is necessary to consider a variety of platforms, different hardware manufacturers, different development approaches (native, web or hybrid) and devices (phones, tablets).
3. Mobile applications are context-dependent, and they do not present a static context (in contrast to traditional platforms and applications).
4. Because of the dynamic environment and context it may not be possible to fully satisfy functional requirements. In this case, it is desirable that application is self-adaptive and thus can provide at least incomplete functionality rather than none.

Similar conclusions were also made during the development of the official IEEE application for iOS and Android platforms. Developers have identified the following specific challenges which are independent of the chosen platform [Tracy 2012]:

1. Because of the mobile device, the speed of the network can vary, since it depends on device's position.
2. Unavailability of the network; mobile applications need to be custom made for execution in offline mode or be able to save the state before exiting.
3. Operating system, dependent on the hardware. For example, Android can behave differently on a variety of different hardware specifications. Developers need to take this into the account.
4. Different screen sizes and different sensor support.
5. Due to the large number of different mobile devices, it is difficult to comprehensively test the application.

One of the major challenges of mobile application development is the development of user interface, as was address by [Larysz et al. 2011], where a comparison and testing of user interfaces has been done for iOS, Android and Windows Phone. Another issue is also choosing a target platform for application development [Ohrt and Turau 2012]. Unlike desktop applications, where 90% of users use Windows OS, mobile application developers must consider a number of current mobile platforms to reach the same percentage of users.

3.2 Software metrics

In this section, the related work of mobile metrics is presented. We have analyzed the current state of measuring the mobile applications' source code. Based on the related work we can conclude, that metrics for the analysis of mobile applications can be divided into two groups, namely, metrics that address the

popularity of applications (number of downloads, installations) and software product metrics that measure both internal and external aspects of application development. As part of the related work, we focused on the internal software product metrics.

In the article [Kumar Maji et al. 2010], authors analyzed the number of bugs and errors on platforms such as Android and Symbian. Besides analyzing the bugs in aforementioned platforms, authors also measured their complexity, using the metrics of McCabe's cyclomatic complexity and the lines of code.

As part of the development of the middleware software, which is located above the Android platform and is responsible for implementation and maintenance of context-aware applications, the authors applied metrics to compare the conventional and their own approach for implementing mobile applications [David et al. 2011]. In addition to the software abstraction their proposed solution also offers composition, reuse and dynamic installation. Their approach was also qualitatively analyzed in the scope of a prototype implementation of the context-aware instant messaging application. The application checks user's movement and if it detects increase in speed, it warns the user. The said application was developed in two ways: simple native Android implementation and implementation using authors' proposed approach. The most important components of the latter are Context Manager Service (CMS), which collects and processes context data and Situation Evaluation Engine (SEE), which is basically a Java library that enables the developer to define their own context situation. The analysis of both applications was conducted with the use of the following metrics:

- *Number of external classes*, which represents the number of system classes that had to be included in the implementation. Due to the architecture of CMS and SEE approach, this number was always 2, as opposed to native Android development, where this value was 8.
- *The number of new methods* represents all the methods that must be implemented in the context of API interfaces. In case of CMS and SEE approach, this number was always 1, whereas for native Android development, this number was 8.
- *External constants* represents the number of constants for error code. Since SEE only returns error when information is not available, this number was always 1. Android had 5 external constants.
- *The number of concepts to learn* is limited to the platform-specific concepts such as Looper, Listener, Android Services, GPS activation etc. In the case of SEE, only Situation and SituationRule are needed, so this number is always 2. Again, Android had a higher number of concepts to learn (7).
- *The number of new lines of program code* was significantly higher for Android (167 vs. 59).
- *The time required for the development of application* (48 hours for Android and 18 hours for CMS and SEE approach).
- *The time required to test the application* (12 hours for Android and 2 hours for CMS and SEE approach).

The review of the related work shows that there are only a few articles that analyze the use of metrics that measure the programming code of mobile applications. The reason for this may be due to the fact that we can use conventional software metrics for the development of mobile applications, which we will examine in details in the following chapters.

4. IMPLEMENTATION OF PROTOTYPE MOBILE APPLICATIONS FOR MULTIPLE PLATFORMS

In this chapter we will introduce the 1) prototype mobile application, 2) supported platforms and 3) process of multiple platform development.

4.1 Prototype mobile application

Our prototype mobile application represents a mobile version of a video store. It enables users to rent and search for movies. The administrators can read the barcodes of the movies for the sake of renting and returning them. Application has been designed in a way that includes the advanced and most commonly

used concepts of mobile platforms, among which are location services, usage of maps, web services and sensors. The goal of the prototype mobile application was to develop an application on different platforms, which will ensure the native look and feel of each platform while providing the same functionalities and similar appearance.

4.2 Selected platforms

The research was limited to the following platforms: iOS, Android and Windows Phone. The reasons for choosing these platforms are as follows: Android and iOS had a more than 50% combined market share in 2011 [Gartner 2011], which is also reflected in the number of applications that are available for said mobile platforms. Half a million applications for iOS [Apple Inc. 2013] and close to 400 thousand for Android [AppTornado GmbH 2012] are a clear indication that these two platforms are popular among the users and developers. Windows Phone is the youngest of the modern mobile platforms, which is speculated to have at least 20% market share in 2015 [Gartner 2011]. The market for Windows Phone applications is also rapidly growing and has reached 50 thousand applications in a little more than a year [Blandford 2011].

Furthermore the following programming languages were used. The Windows Phone application was developed using C# programming language combined with XAML markup language. Visual Studio 2010 IDE was used. The Android application was built in Eclipse IDE, using Java programming language, combined with XML. Finally, iOS application was developed using Objective-C programming language in Xcode IDE.

4.3 Development of the prototype application

When developing applications for multiple platforms, a goal is not only to achieve quick and easy to use solutions, but also to preserve the unique user experience (look and feel), depending on the platform [Hartmann et al. 2011]. In the following subchapters we defined two aspects, which have to be unified when developing an application for different platforms, namely building a common user interface and business logic.

4.3.1 Building a common user interface

When building a user interface for cross-platform application, we have to take into the consideration advantages and disadvantages of specific platforms. When trying to achieve the same user experience (regardless of the mobile platform) we have to be aware of these restrictions or the application can result in an unnatural user experience. The visualization of the graphical elements can also impact the user experience. Although Android and iOS are somewhat visually similar, this does not apply to Windows Phone platform. The latter includes the "Windows 8 style UI", which significantly differs from others.

After we analyzed graphical elements of the aforementioned platforms, we found out that out of 26 identified, there are 19 elements on all three platforms, which have the same functionality. By using these elements we are guaranteed the same functionality of the user interface, as well as visual consistency with design guidelines of platforms. Visual coherence defines not only the correct visual form of graphical elements, but also compliance with the color schemes that are imposed by the platform. While iOS has no specifications regarding the color schemes, with the arrival of Android 4.0, Google indicated the visual guidelines for developers. Windows Phone on the other hand promotes a dynamic color changing in the applications, which depends on the color scheme the user has set for the entire operating system [Microsoft 2010].

Figure I represents one of the final screens of our applications on Windows Phone, Android and iOS platform, respectively.

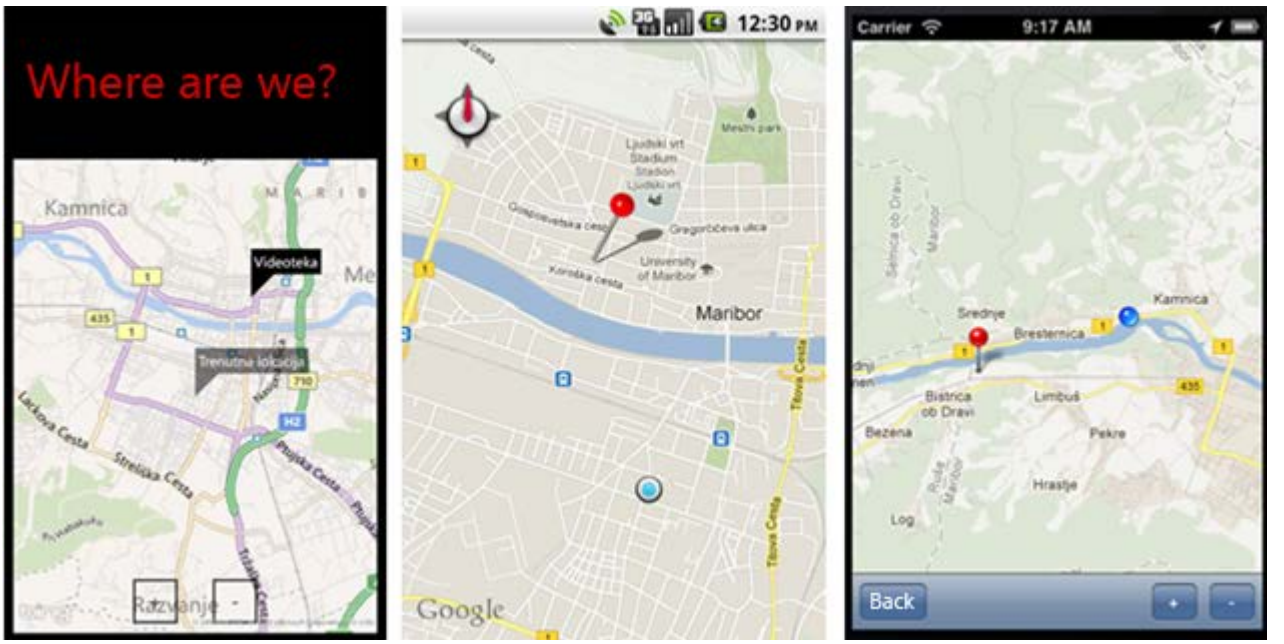

Figure I: Sample screen of the mobile application across different platforms (Windows Phone, Android and iOS, respectively)

4.3.2 Business logic

Developing applications for multiple platforms can be the cause of code duplication. If we want to maximize the reuse of code across platforms, we have to move it to the server side. In determining which part we want to move to the server, it is important to consider several factors. So, it is advisable to move the business logic to the server in the following cases:

- A particular operation consumes a lot of computing resources.
- The consistency of data is required.
- Business logic changes often, in which case we do not need to change client applications and maintain multiple versions of the business logic for older clients.
- We operate with a larger amount of data. In this case, it might not be practical to keep it entirely on the client.

However, there are situations where business logic should be on the client side. Such scenarios are as follows:

- When client has to transfer a large amount of data (for example, resizing the images).
- There is a need for off-line mode.
- There is a need for faster computing.

Business logic on the server side is usually exposed in the form of web services. This mode thus brings a new architecture: a web service-client. There are currently two popular methods of exposing web services: REST and SOAP. If we are developing a mobile applications for all three platforms, we have to use REST web services, since SOAP services are not (officially) supported on iOS and Android. REST is not a protocol but rather an architecture, which uses HTTP protocol to transfer data. So, only four methods (GET, POST, PUT, and DELETE) out of the nine are available. Regarding the format of the data, we have two options: XML or JSON [Rodriguez 2008].

Table I represents the distribution of files across the selected platforms, based on their purpose in the application structure.

Table I: Distribution of files by purpose across different platforms

Type of the files	Android	iOS	Windows Phone
Business logic	10	8	8
User interface	29	23	12
Utility files	21	5	3
Other files	3	6	5
Together	63	42	28

As is evident from the table above, Windows Phone platform uses the most files for graphical user interface (12 file out of 28 files). Second most of other files are used for business logic (8 files). In case of iOS, the majority of files is again used for graphical user interface (23 files out of 42 files). They are followed by files for business logic (8 files). Similarly, in case of Android platform, majority of files are used for the GUI purposes (29 files out of 63 files), followed by utility files (21) and business logic (10).

However, it should be noted that the server side logic was not part of the analysis.

5. METRIC ANALYSIS OF PROTOTYPE MOBILE APPLICATIONS

In this chapter, selected metrics are introduced. Furthermore, the selected plugins and tools for static code analysis are defined and the results of the metric analysis are presented.

5.1 List of selected metrics

For the sake of evaluating the code quality, besides other traditional software metrics, we primarily used the Chidamber and Kemerer (CK) metrics. The selected metrics along with the brief descriptions are provided below [North Carolina State University 2005], [Rosenberg 1998], [Chidamber and Kemerer 1994].

The first metric, depth of inheritance tree (hereinafter referred to as DIT), is defined as the maximum length from the node to the root of the tree. DIT is predicted on the following assumptions:

- The deeper a class in the hierarchy, the greater the number of methods it will most likely inherit.
- Deeper trees involve greater design complexity (due to the fact that more classes and methods are present).
- Deeper classes in the tree have a greater potential for reusing inherited methods, which is good for potential code reuse.

Number of children (NOC1) is the number of direct subclasses for each class. Classes that have a large number of children are considered difficult to modify and are also considered more complex and fault-prone.

Lack of Cohesion in methods (LCOM) represents the number of disjoint or non-intersection sets of local methods. A higher LCOM value represents a good class subdivision, whilst a low cohesion increase complexity and subsequently increasing the possibility of errors during the development of software.

Weighted Methods per Class (WMC) represents the McCabe's cyclomatic complexity. WMC is dependent on number of Methods (NOM), which indicates complexity of an individual class. Both higher WMC and NOM of a class suggest, that such a class is more complex than its peers and more application specific, limiting the potential reuse of code.

Number of classes (NOC2) is significant when comparing projects with identical functionality. Projects with more classes are perceived as being better abstracted.

Similarly to NOC2, lines of code (LOC) metric is also significant when comparing similar projects. LOC indicates the approximate number of lines in the software code. A higher LOC value might indicate that a type or method is doing too much work. Subsequently, it might also indicate that the type or method could be hard to maintain.

Cyclomatic complexity (CC) indicates how hard software code may be to test or maintain and how likely the code will produce errors. Generally, we determine the value of cyclomatic complexity by counting the number of decisions in the source code.

The described metrics are summarized in Table II in light of preferred values (either high or low) for better code quality.

Table II: Summary of selected metrics

Metric	**Desired value**
Depth of the inheritance tree	Low
Number of Children	Low
Lack of Cohesion in methods	Lower
Weighted methods per class	Low
Number of methods	Low
Number of Classes	Higher
Lines of Code	Lower
Cyclomatic complexity	Low

5.2 Results of the metric analysis

The metrics for analyzing prototype mobile applications were gathered by using the following software or plugins, depending on the platform:

- In case of Windows Phone, we used "NDepend", a tool for static code analysis for .NET applications [Smacchia 2013].
- For analyzing Android application source code, we used "Metrics2" plugin, which is available for Eclipse integrated development environment (IDE) [Sauer 2002].
- Xcode, which was used for developing the iOS application, does not enable a metric analysis of applications, it only notifies the developer in case of errors in the code. Thus, we used "Understand Metrics", a tool for static code analysis, which, among others, enables the analysis of Objective-C [Scientific Toolworks, Inc 2013].

Table III represents the results of the analysis of the source code. Cells with the gray background represent the least desirable values among platforms.

Table III: Results of metric analysis of prototype mobile applications

Metrics	**Android**	**Windows Phone**	**iOS**
DIT (average)	**2,237**	0,5	0,033
NOC1 (average)	**0,475**	0,055	0,033
LCOM (average)	0,186	0,452	**0,764**
WMC (average)	7,242	**15,174**	4,494
NOM	**249**	143	191
NOC2	**59**	18	49
LOC	2707	554	**3482**
CC (average)	1,72	**1,91**	1,15

As is evident from the Table III, Android has the highest DIT. However, this is mostly due to the fact that Android platform requires the inheritance of certain system classes, which already have a high DIT by default. The maximum value of DIT in case of Android was six. An example of such a class represents the screen for displaying the map. This particular class inherits the following chain of classes: *MapActivity* ← *Activity* ← *ContextThemeWrapper* ← *ContextWrapper* ← *Context* ← *Object*. In case of Windows Phone and iOS, we used an external tool for static code analysis. These tools do not take into the account the inheritance of system classes within platform libraries.

Differences between platforms can be once again seen in case of NOC1, which was the highest in the case of Android platform. However, "Metrics" plugin takes into the consideration not just classes, but also the interfaces. Fundamentally, the default Windows Phone and iOS structure does not include as many interfaces as Android.

LCOM is the lowest in the case of Android platform, mainly due to the GET, SET and toString methods. This is also reflected when viewing the metrics for packages and individual classes.

WMC was calculated using the following formula:

$$WMC = \left(\frac{NOM}{NOC2}\right) * CC$$

WMC has the highest value in case of Windows Phone platform, which means that there are more methods per class than in case of other platforms. It should be noted that the complementary metric NOM indicates that application, developed for Android platform has the most methods. However, in case of Android platform, these methods are divided into several classes. The increase in WMC in case of Windows Phone is mainly due to the frequency of asynchronous service calls and rewriting of the methods to navigate between pages of application.

When interpreting the LOC it is important to consider the programming language, SDK and related set of functionalities (e.g. libraries). Thus, in case of Windows Phone, the .NET framework includes advanced libraries which is also reflected in LOC metric. Parsing XML with LINQ and inclusion of attributes in entities significantly decreases the LOC metric value. Conversely, in case of Android platform, we have GET and SET methods, which additionally increase the LOC metric value. CC metric focuses on the number of branches and loops. CC metric had the lowest value in case of the iOS platform. Both Android and Windows Phone 7.5 had the higher CC metric value, mostly due to the usage of dynamic screens, whereas in case of iOS the navigation between screens is simplified using the iOS Storyboard. Since we were able to apply the metrics to mobile application's source code and gathered sound results, we failed to reject the null hypothesis H_{01}. However, since the results were not similar among the selected platforms for the functionally equivalent applications, the second null hypothesis H_{02} was rejected in favor of H_{A02} (see Table III).

6. CONCLUSION

The objective of our goal was to examine whether the traditional software metrics are appropriate for measuring the mobile applications' source code and to this end, a small-scale application was developed across different platforms. The code was then evaluated by using the traditional software metrics. After analyzing the results, we obtained sound results, regardless of the platform. In this section, we will highlight the limitations of our research, which will be followed by implications and future work. Finally, a summary of this article will be presented.

6.1 Validity evaluation

When interpreting our results, the following limitations should be taken into the account. Firstly, we have built a small-scale applications for different platforms. Larger-scale application might produce different results. Secondly, only three of the existing mobile platforms have been used. Object-oriented metrics might not be appropriate for other mobile platforms. Thirdly, out of the selected platforms, the following versions were used: iOS 5.0, Windows Phone 7.5 and Android 2.1. More recent versions of these platforms might produce different results when analyzing the source code. Fourthly, the scope of the selected metrics was limited by capabilities of existing plugins and other third-party tools for static code analysis. Other metrics might not be appropriate for mobile applications. Fifthly, we did not conduct any advanced statistical analysis, which would confirm or reject statistically significant differences of metrics' values between platforms. Sixthly, prototype mobile applications were developed by three different programmers, one for each of the selected platforms. Had the same programmer develop mobile applications for the said platforms, the results of the metric analysis might differ. And finally, the metrics

were not validated within the scope of mobile environment, but were merely applied to our prototype mobile applications.

6.2 Implications and future work

Although the existing literature suggested the usage of OO metrics in the scope of mobile application development, none had any empirical data. Also, none of the existing literature compared results of applying metrics between functionally equivalent mobile applications on different platforms. In this light, our results present the starting point for detailed analysis of applying OO metrics to the domain of mobile application development. The most important finding therefore was the fact that we successfully applied OO metrics to mobile domain. Also, our results suggest that the difference between mobile platforms in the sense of applying the metrics to the source code of mobile application is present. Thus, an application with the similar functionalities, graphical user interface and used sensors might produce different metrics' values when considering the differences between the platforms. So, when applying OO metrics to mobile domain, those differences should be considered in the future research.

In the future we plan to conduct an empirically validated and statistically detailed analysis of OO metrics in the scope of a large-scaled mobile application development. We will include more existing mobile platforms (e.g. Blackberry) as well as multi-platform frameworks.

6.3 Summary

The objective of our article was to examine whether the traditional software metrics are appropriate for measuring the mobile applications' source code. A small-scale application was then developed across three different platforms (Android, iOS and Windows Phone). We evaluated the code by using the traditional software metrics. After applying these metrics and analyzing the code, we obtained sound results on all the platforms. If we aggregate the results, we can argue that traditional software metrics can be used for mobile applications' source code as well, which was the notion of the first null hypothesis. However, since the results of metrics' analysis of functionally equivalent application were slightly different between platforms, the second null hypothesis was rejected. Further detailed statistical analysis is required, in the scope of a more complex, large-scale mobile applications.

ACKNOWLEDGEMENTS

The authors wish to thank Aleš Černezel, Mitja Krajnc and Boris Ovčjak for their assistance in preparing the prototype mobile applications as well as for helping with writing of the article.

REFERENCES

Apple Inc., 2013. The AppStore. Apple.

AppTornado GmbH, 2012. Number of available Android applications. AppBrain.

Blandford, R., 2011. Windows Phone Marketplace passes 50,000 apps. All About Windows Phone.

Chidamber, S.R. and Kemerer, C.F., 1994. A metrics suite for object oriented design. IEEE Transactions on Software Engineering, 20(6), pp.476–493.

David, L., Endler, M., Barbosa, S.D.J. and Filho, J.V., 2011. Middleware Support for Context-Aware Mobile Applications with Adaptive Multimodal User Interfaces. In 2011 4th International Conference on Ubi-Media Computing (U-Media). 2011 4th International Conference on Ubi-Media Computing (U-Media). pp. 106–111.

Dehlinger, J. and Dixon, J., 2011. Mobile Application Software Engineering: Challenges and Research Directions. In Santa Monica, CA, USA.

Franke, D. and Weise, C., 2011. Providing a Software Quality Framework for Testing of Mobile Applications. In 2011 IEEE Fourth International Conference on Software Testing, Verification and Validation (ICST). 2011 IEEE Fourth International Conference on Software Testing, Verification and Validation (ICST). pp. 431–434.

Gartner, 2011. Gartner Says Android to Command Nearly Half of Worldwide Smartphone Operating System Market by Year-End 2012. Gartner Newsroom.

Gordon, A.J., 2013. Concepts for mobile programming. In Proceedings of the 18th ACM conference on Innovation and technology in computer science education. ITiCSE '13. New York, NY, USA: ACM, pp. 58–63.

Hartmann, G., Stead, G. and DeGani, A., 2011. Cross-platform mobile development.

Kumar Maji, A., Hao, K., Sultana, S. and Bagchi, S., 2010. Characterizing Failures in Mobile OSes: A Case Study with Android and

Symbian. In 2010 IEEE 21st International Symposium on Software Reliability Engineering (ISSRE). 2010 IEEE 21st International Symposium on Software Reliability Engineering (ISSRE). pp. 249–258.

Larysz, J., Němec, M. and Fasuga, R., 2011. User Interfaces and Usability Issues Form Mobile Applications. In V. Snasel, J. Platos, & E. El-Qawasmeh, eds. Digital Information Processing and Communications. Communications in Computer and Information Science. Springer Berlin Heidelberg, pp. 29–43.

Microsoft, 2010. UI Design and Interaction Guide for Windows Phone 7, version 2.0.

Nebeling, M., Zimmerli, C. and Norrie, M., 2013. Informing the design of new mobile development methods and tools. In CHI '13 Extended Abstracts on Human Factors in Computing Systems. CHI EA '13. New York, NY, USA: ACM, pp. 283–288.

North Carolina State University, 2005. An Introduction to Object-Oriented Metrics.

Ohrt, J. and Turau, V., 2012. Cross-Platform Development Tools for Smartphone Applications. Computer, 45(9), pp.72–79.

Rodriguez, A., 2008. RESTful Web services: The basics. IBM.

Rosenberg, L., 1998. Applying and Interpreting Object Oriented Metrics. In Software Technology Conference. NASA Software Assurance Technology Center (SACT).

Sauer, F., 2002. Eclipse Metrics plugin.

Scientific Toolworks, Inc, 2013. Understand Metrics.

Smacchia, P., 2013. NDepend.

Tracy, K.W., 2012. Mobile Application Development Experiences on Apple #x2019;s iOS and Android OS. IEEE Potentials, 31(4), pp.30–34.