EPiC Series in Computing

Volume 64, 2019, Pages 119–128

Proceedings of 28th International Conference
on Software Engineering and Data Engineering

Adaptation of Orthogonal Defect Classification for Mobile Applications

Osama Barack[1] and LiGuo Huang[2]

[1] Southern Methodist University, Dallas, Texas, USA
obarack@smu.edu
[2] Southern Methodist University, Dallas, Texas, USA
lghuang@smu.edu

Abstract

As mobile applications have become popular among end-users, developers have introduced a wide range of features that increase the complexity of application code. Orthogonal Defect Classification (ODC) is a model that enables developers to classify defects and track the process of inspection and testing. However, ODC was introduced to classify defects of traditional software. Mobile applications differ from traditional applications in many ways; they are susceptible to external factors, such as screen and network changes, notifications, and phone interruptions, which affect the applications' functioning. Therefore, in this paper, the ODC model will be adapted to accommodate defects of mobile applications. This allows us to address newly introduced application defects found in the mobile domain, such as energy, notification, and Graphical User Interface (GUI). In addition, based on the new model, we classify found defects of two well-known mobile applications. Moreover, we discuss one-way and two-way analyses. This work provides developers with a suitable defect analysis technique for mobile applications.

1 Introduction

Due to the importance and popularity of services that mobile applications provide, mobile applications require a short and accurate cycle of defect classification that is adapted to the nature of mobile environments. Ram Chillarege [5] introduced the concept of Orthogonal Defect Classification (ODC) for traditional software and defined it as "a concept that enables in-process feedback to developers by extracting signatures on the development process from defects". ODC was originally introduced to classify defects of traditional software, whereas the nature of mobile applications differs somewhat. In mobile environments, new functionalities and features were developed, which introduced additional factors such as energy, network, incompatibility, Graphical User Interface (GUI), interruption, and notification. Defects originating from these factors need to be classified for better defect resolution. Therefore, to benefit from the ODC concept, ODC needs to be adapted to attend to these new factors.

In this paper, we adapt ODC to mobile applications by considering the characteristics of the mobile environment. This includes adding the new factors to the ODC framework to classify defects in order to improve the reliability of mobile applications after release. In addition, we

F. Harris, S. Dascalu, S. Sharma and R. Wu (eds.), SEDE 2019 (EPiC Series in Computing, vol. 64),
pp. 119–128

provide one-way and two-way analyses to accommodate the results of the classification process. The provided classification process and analyses are based on defect reports of two mobile applications, Tomdroid and Telegram. Tomdroid [2] is a note-taking application that has a unique wiki-style display in the Android platform, and Telegram [7] is a cloud-based instant messaging and voice over IP service.

The remainder of this paper is structured as follows: Section 2 provides related work. Section 3 explains our proposed method to adapt ODC to mobile environments. Section 4 presents a case study. Section 5 provides an analysis of the results and a discussion. Section 6 presents the conclusions and future work that can be achieved in this field.

2 Related Work

With the widespread presence of traditional software, Chillarege et al. [5] [4] introduced the ODC concept to classify defects and find their root causes. This allows for reaching a short cycle of defect analysis during the software development process.

With the internet and web applications have become worldwide providers of online services, Ma and Tian [13] presented an adaptation of ODC for web errors based on defects found in web logs, and they provided analyses of their results to improve the reliability of web applications.

Cloud computing is the new revolution of web services by providing Software as a Service (SaaS). Alannsary and Tian [1] proposed a defect analysis framework by adapting ODC to SaaS. The new framework considered new characteristics introduced in SaaS, such as multi-tenancy and isolation.

With the lack of defect classifications during black-box testing, Li et al .[11] presented a new defect classification framework called Orthogonal Defect Classification for Black-box Defect (ODC-BD). The proposed framework aims to help black-box defect analyzers and black-box testers, while enhancing the efficiency of the analysis and testing processes.

Wasserman [14] provided an overview of research issues in software engineering for mobile software development. The overview included development processes, tools, user interface design, application portability, quality, and security.

Different approaches inspired by the new types of faults and failures that arise in mobile applications during and after the development process have been discussed. Holl and Elberzhager [9] classified mobile application failures and defined the relationships between their failures and various aspects of their faults. Lelli et al. [10] proposed a model that identifies and classifies GUIs' faults. In their work, they presented an empirical analysis and assessment for their model. However, a classification framework that does not cover all types of mobile application defects hinders in-process feedback from being fast and accurate.

Existing studies provided helpful frameworks to classify traditional software defects in general. However, due to the popularity and high demand of mobile applications, classifying mobile application defects is crucial to software development. Therefore, we propose adapting the ODC concept to mobile environments to classify the new types of defects, improve in-process feedback, and present the results and analyses of applying our framework to bug reports of mobile applications.

3 New Methodology

When mobile software development started, mobile applications were small with only a few thousand lines of code compared to traditional software. Since then, mobile software devel-

Type	Description	Process Associations
Function	Capability, product interface, interface with hardware or global data structures (requires a formal design change)	Design
Interface	Errors from interactions with other components, modules or device drivers via macros, call statements, control blocks, or parameter lists.	LLD
Checking	Program logic that failed to validate data and values before they are used	LLD or Code
Assignment	A few lines of code errors, such as initialization of control blocks or data structure	Code
Timing/Serialization	Real-time resources	LLD
Build/Package/Merge	Library systems, management of changes, or version control	Library Tools
Documentation	Publication and maintenance	Publications
Algorithm	Errors include efficiency or correctness problems of tasks or data structures	LLD
Energy	**Exhausting battery energy through excessive usage of mobile device components such as CPU, memory, sensors, etc.**	**LLD**
Network	**Improper handling of network connections**	**Code or LLD**
Incompatibility	**Mobile application is not customized for the mobile device, operating system, interactions with other applications, or supporting mobile features**	**LLD**
GUI	**Errors generated from end-user interface or screen changes**	**Design, LLD, or Code**
Interruption	**Improper handling of receiving calls or messages, activating screen saver state, or switching between applications**	**Components or Code**
Notification	**Errors from application alerts, including sound, vibration, visual, and text**	**LLD or Code**

Table 1: Adapted ODC for Mobile Environments

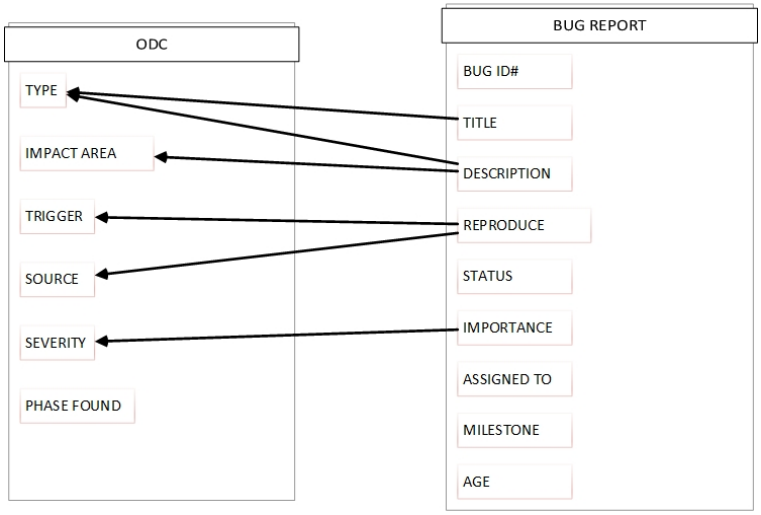

Figure 1: Mapping of Mobile Application Defect Report components to ODC components

opment has grown exponentially and has become more complex. Software engineering and software quality were applied to mobile software development to ensure its accuracy. This introduced new requirements that need to be considered during the development process, such as power consumption, interacting with other mobile applications and environments, screen and sensor handling, and limitations of mobile devices.

The characteristics of mobile application environments were examined and studied to identify the differences between defects of traditional software and those of mobile applications. In the original ODC, defect types were characterized as *function, interface, assignment, checking, timing/serialization, build/package/merge,* and *algorithm.* However, there are new defects that cannot be classified using the original ODC. Therefore, the proposed framework caters for mobile application defects. As shown in bold in Table 1, we added the following defect types to the original ODC. A brief description for each defect type is provided below:

1. ***Energy***: Mobile devices consume energy from batteries to run, whereas PCs use power. Due to excessive runs of application code, defects start arising by consuming more energy.

2. ***Network***: In mobile environments, networks change rapidly. Due to improper handling of these changes, mobile applications produce defects and cause crashing or freezing.

3. ***Incompatibility***: Due to the variety of mobile devices and their operating systems, mobile environments do not support some mobile applications or their features; therefore, defects are produced from lacking to consider mobile environment limitations.

4. ***GUI***: There is a strong relationship between user interface and mobile screen properties such as size, touch, landscape, color, brightness, etc. Since the user interface is essential when working with mobile applications, in our proposed framework, we extracted the user-interface subcategory from the *function* defect type and created a new defect type called *GUI.*

5. ***Interruption***: Mobile devices get interrupted by calls, messages, alerts, etc. Improper handling of these interruption may cause defects that make mobile applications stop

Severity	Tomdroid # of Defects	Telegram # of Defects
Critical	5	5
High	52	21
Medium	36	33
Low	38	9

Table 2: One-Way Analysis Based On Defect Severity

responding.

6. ***Notification***: Many different types of mobile notifications may produce defects by giving false notifications (time, place, or content).

To properly classify defects using bug reports, components of these reports (title, description, reproduce, and importance) are mapped to the components of the ODC model (type, impact area, trigger, source, and severity) as illustrated in Figure 1. This enables performing one-way, two-way, and multi-way analyses.

4 Case Study

Canonical has developed a website and a web application called Launchpad [12] to allow open-source software to be developed and maintained by end-user developers. Launchpad has 42,947 projects and 1,779,680 bug reports. We chose two popular mobile applications from Launchpad. First, the Tomdroid [2] is a note-taking application that has a unique wiki-style display in the Android platform. In addition, Tomdroid has a format that enables syncing notes with the Tomboy application [6]. Second, the Telegram [7] is a cloud-based instant messaging and voice-over IP service. The reporting bug site categorizes bugs by severity (critical, high, medium, low, wishlist, and undecided). Tomdroid had 220 bug reports [3], and Telegram had 246 bug reports [8]. In our study, we only considered bugs that had severity levels from *critical* to *low*. Wishlist bugs were discarded since they were not considered defects. Undecided bugs were ignored due to they were not categorized. Therefore, 131 defects for Tomdroid and 68 defects for Telegram were applicable for our case study. Furthermore, each bug was manually studied and classified based on one of the defect types available in the proposed ODC framework. This classification process was preformed by three individuals with computer science backgrounds specifically in software engineering.

5 Analyses of the Results and a Discussion

Classifying defects of mobile applications provides one-way, two-way, and multi-way analyses. We analyzed the descriptions and importance of the bug reports for the selected mobile applications to identify the *severity* level and *type* for each defect. In addition, we performed one-way analysis by examining one of the proposed ODC attributes, such as *type* and/or *severity*, in order to focus on areas where defects were highly dense. Moreover, we performed two-way analysis by examining the intersection between *type* and *severity* to explore areas that were not covered in the one-way analysis.

Type	Tomdroid # of Defects	Telegram # of Defects
Function	6	14
Interface	3	1
Checking	-	-
Assignment	47	17
Timing/Serialization	11	-
Build/Package/Release	1	1
Documentation	-	-
Algorithm	3	2
Energy	-	2
Network	-	5
Incompatibility	29	5
GUI	26	10
Interruption	2	1
Notification	3	10

Table 3: One-Way Analysis Based On Defect Type

Severity	Type	# of Defects
Critical	Assignment	1
	Timing/Serialization	1
	Incompatibility	3
High	Assignment	20
	Timing/Serialization	7
	Incompatibility	17
	GUI	6
	Notification	2
Medium	Function	2
	Assignment	15
	Timing/Serialization	2
	Build/Package/Release	1
	Algorithm	2
	Incompatibility	5
	GUI	7
	Interruption	2
Low	Function	4
	Interface	3
	Assignment	11
	Timing/Serialization	1
	Algorithm	1
	Incompatibility	4
	GUI	13
	Notification	1

Table 4: Tomdroid: Two-Way Analysis Based On Defect Severity and Type

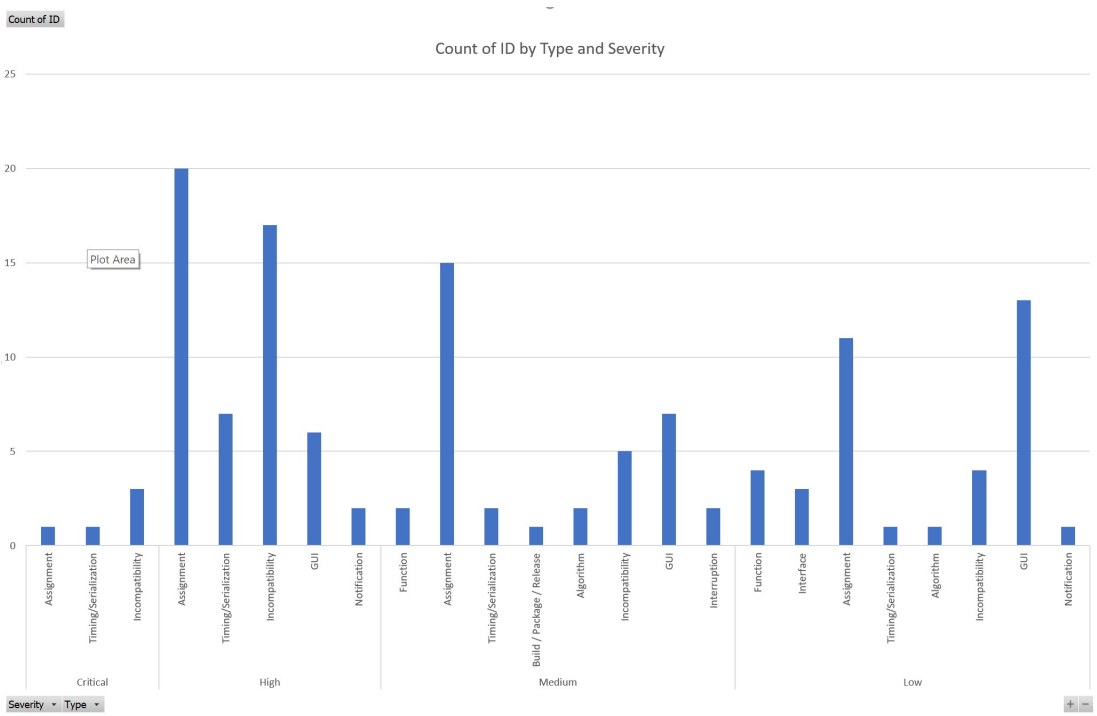

Figure 2: Tomdroid: Two-Way Analysis Based On Defect Severity and Type

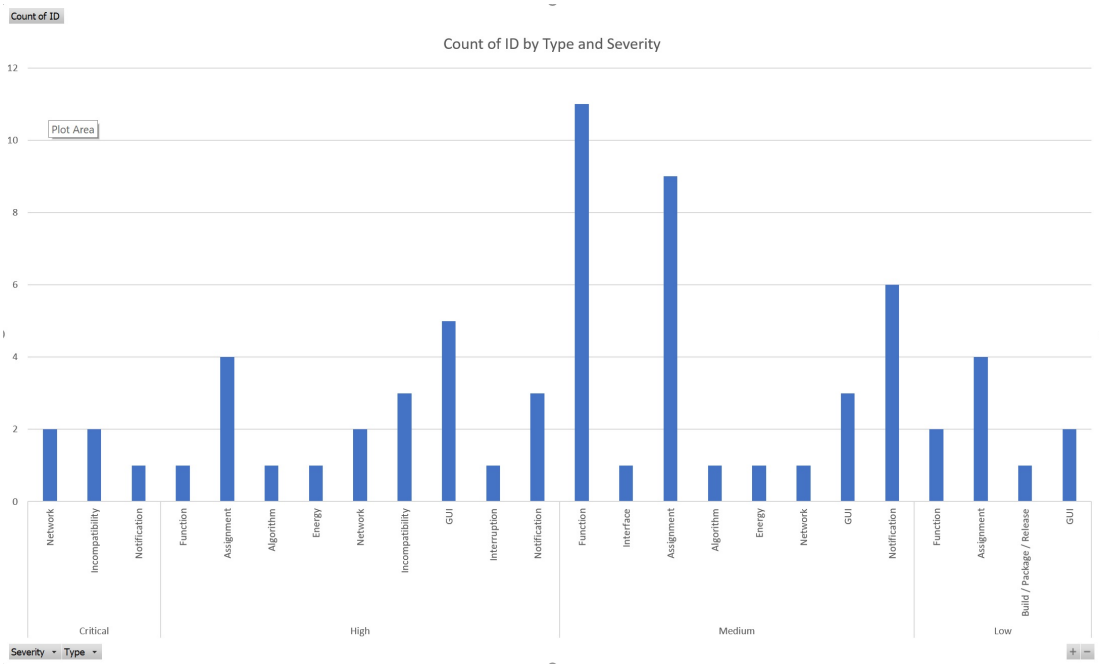

Figure 3: Telegram: Two-Way Analysis Based On Defect Severity and Type

Severity	Type	# of Defects
Critical	Network	2
	Incompatibility	2
	Notification	1
High	Function	1
	Assignment	4
	Algorithm	1
	Energy	1
	Network	2
	Incompatibility	3
	GUI	5
	Interruption	1
	Notification	3
Medium	Function	11
	Interface	1
	Assignment	9
	Algorithm	1
	Energy	1
	Network	1
	GUI	3
	Notification	6
Low	Function	2
	Assignment	4
	Build/Package/Release	1
	GUI	2

Table 5: Telegram: Two-Way Analysis Based On Defect Severity and Type

5.1 One-Way Analysis

One-way analysis can be done by analyzing one attribute of the ODC framework at a time. The *severity* and *type* defect attributes are used to find those areas with high defect rates that need attention to reduce the number of defects and eliminate their root causes.

Table 2 illustrates the classification of defects by severity level for both mobile applications. Most defects in the Tomdroid application were produced with *high severity*, indicating that developers should focus on these defects and find their root causes. However, no red flags were raised for the Telegram application since most defects were classified as *medium severity*.

Table 3 illustrates the classification of defects by the *type* attribute. For both applications, most defects were produced for the *assignment type*, raising a red flag about the level of the programmers' skills and experience. For Tomdroid, the second most found defects were produced for the *incompatibility* and *GUI* defect *types*, indicating that developers need to be aware of the application's interactions with different systems, and improve the handling of the user interface and device screen. For Telegram, The *function* defect *type* showed that the software development stages-analysis and design-were producing the second highest number of defects in the application. In addition, the *GUI* and *notification* defect *types* also had high numbers of defects, indicating that the handling of the application notifications need to be improved, and the user interface and screen states need to be properly designed and adjusted to the nature of the mobile environment.

5.2 Two-Way Analysis

Two-way analysis can be done by analyzing the intersection between two of the ODC attributes, allowing the information that is related to the defects to be displayed, identify their root causes, and find the best solutions. In this case study, we analyzed the intersection between the *type* and *severity* defect attributes.

Table 4 and Figure 2 illustrate that the high rate of defects was concentrated in the intersection between the attribute severity: *high* and the types: *assignment* and *incompatibility*. This confirms that programmers are injecting the code with faults as shown in the one-way analysis, which may indicate their lack of programming experience or skills. This might also suggest improving the code testing stage before the mobile application release.

Table 5 and Figure 3 illustrate that the high rate of defects was concentrated in the intersection between the attribute severity: *medium* and the types: *function*, *assignment*, and *notification*. Since the severity was *medium*, this does not raise much concern. However, the *types function* and *notification* indicate that the analysis and design stages need to be performed with more consideration of the mobile environment's characteristics and nature.

5.3 Discussion

This paper demonstrated adapting the ODC concept to mobile environments to confirm its feasibility. Defects generated from mobile environments and related to new types, such as *energy* and *notification*, can be classified by the original ODC. However, it will provide misleading information related to the defect's origin and may lead to not exposing the root cause, which can prevent finding the best solution. Exposing mobile application defects by implementing the proposed ODC framework will benefit developers in acquiring short and accurate in-process feedback. In addition, classifying the two mobile application defects proved that the proposed ODC adaptation is significantly effective for defect classification and resolution, specifically when implementing one-way and two-way analyses.

6 Conclusions and Future Work

In this paper, we proposed adapting the ODC model to mobile environments to classify defects during software development and after release. New characteristics and factors of mobile environments and applications were studied and considered, which led to adding new defect types to the original ODC framework. In addition, a case study was presented where defects from bug reports were extracted and classified based on the proposed framework. Moreover, we provided one-way and two-way analyses and discussed their results. Our work will make in-process feedback more accurate during mobile software development and improve software reliability. In future work, we will classify defects of mobile applications from different mobile environments in order to generalize our findings.

References

[1] M Alannsary and J Tian. Cloud-odc: Defect classification and analysis for the cloud. In *Proceedings of the International Conference on Software Engineering Research and Practice (SERP)*, page 71. The Steering Committee of The World Congress in Computer Science, 2015.

[2] Olivier Bilodeau. Tomdroid application. https://launchpad.net/tomdroid.

[3] Olivier Bilodeau. Tomdroid bug report. https://github.com/tomboy-notes/tomdroid/issues.

[4] Ram Chillarege. Odc-a 10x for root cause analysis. 2006.

[5] Ram Chillarege, Inderpal S Bhandari, Jarir K Chaar, Michael J Halliday, Diane S Moebus, Bonnie K Ray, and M-Y Wong. Orthogonal defect classification-a concept for in-process measurements. *IEEE Transactions on software Engineering*, 18(11):943–956, 1992.

[6] Alex Graveley. Tomdroid application. https://github.com/tomboy-notes/tomboy.

[7] Tiago Salem Herrmann. Telegram application. https://launchpad.net/telegram-app.

[8] Tiago Salem Herrmann. Telegram bug report. https://bugs.launchpad.net/telegram-app.

[9] Konstantin Holl and Frank Elberzhager. A mobile-specific failure classification and its usage to focus quality assurance. In *2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 385–388. IEEE, 2014.

[10] Valéria Lelli, Arnaud Blouin, and Benoit Baudry. Classifying and qualifying gui defects. In *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*, pages 1–10. IEEE, 2015.

[11] Ning Li, Zhanhuai Li, and Xiling Sun. Classification of software defect detected by black-box testing: An empirical study. In *2010 Second World Congress on Software Engineering*, volume 2, pages 234–240. IEEE, 2010.

[12] Canonical Ltd. Launchpad. https://launchpad.net.

[13] Li Ma and Jeff Tian. Web error classification and analysis for reliability improvement. *Journal of Systems and Software*, 80(6):795–804, 2007.

[14] Tony Wasserman. Software engineering issues for mobile application development. *FoSER 2010*, 2010.