Improving Quality on Native and Cross-Platform Mobile Applications

Rudy Bisiaux, Mikael Desertot, Sylvain Lecomte, Joachim Perchat, Dorian Petit
FRANCE
name.firstname@univ-valenciennes.fr

***Abstract*—This paper discusses quality markers for a mobile application, both during conception and development, to propose the most suitable way to validate them automatically. We focus on native and cross-platform applications, as well as component based development. To achieve this, we rely both on research papers of the domain, and on our partnership with the Keyneosoft company. An industrial expertise is useful to identify real problems encountered with quality processing for mobile applications.**

Keywords*–*Mobile, Quality, Product Line.

I. INTRODUCTION

With more than 4 billion mobile devices around the world [5] and more than 5 million applications in the different stores [2], the mobile is omnipresent. But developing mobile applications means complying with several constraints and that comes at a cost [22]. In this paper, we highlight the differences between the implementation of classic apps (for Desktop, Web or server) and mobile apps. Afterwards, we define the quality of a mobile application [24] and determine some quality checkpoints. Finally, we describe a solution for validating these checkpoints, to be able to evaluate the overall quality of a mobile application. The objective is to reduce the cost of creating and maintaining these applications by addressing quality control in these two phases. A lot of time is wasted to rollback, hot-fix or replace parts of the application during conception, development or tests, if a minimum quality threshold is not reached. To do this, we introduce the mobile application development concept in Section II. We define the needs for mobile application development, software quality and software engineering in Section III. We detail our approach in Section IV. To finish we conclude in Section V.

II. MOBILE DEVELOPMENT

Developing mobile applications is different from developing classic software even if some similarities exist (like conception, development, test or continuous integration). Two main divergent points are explained based both on the Keyneosoft experience and a survey about mobile applications development challenges [15].

A. Market constraints

A mobile application is produced and released only in a few weeks. This implies the creation of the application quickly and correctly from the beginning. Once deployed, there is no time left to correct mistakes. Moreover, the first release of a mobile application will contain only a few primary features. Afterwards, more and more features are added. The quality of the initial application and all its additional features have to be certified. An application with poor development quality will be more difficult to manage, and adding features will cause regressions.

The heterogeneity of mobile Operating Systems, even in the same family, is also a major problem. The behavior of an application can be different between two OS versions. An example is given by Android version 6.0, where an application needs runtime permissions to work whereas these permissions were not mandatory on previous versions of this OS [11]. To reach all the market stores (Android, iOS) with an application, we have to multiply the supported OS (different languages imply a higher cost of production). Some answers have been proposed, offering cross-platform solutions. Multiple cross-platforms frameworks exist, which are based on web development like Ionic [14] or cross compiling like COMMON [17] and Xamarin [6].

Another heterogeneity issue is due to the manufacturers who apply overlays on OS. Once again, the behaviour of the application can be disturbed by these overlays and the developer has to check for all of them. The last heterogeneity drawback concerns the device screen size. The user interface has to be clear and consistent regardless of the screen size. But with Android or iOS devices, managing the screen size has also a cost, even when using cross-platform solutions.

All these constraints are imposed by the market; they can not be changed but need to be considered when developing mobile applications.

B. Development constraints

To deal with the complexity of mobile applications, different kinds of designs are available. The most popular is the MVC (Model-View-Controler) pattern [19], but some technologies like Xamarin replace this design pattern with MVVM (Model-View-ViewModel) [20], where a view-controller replaces the current controller to notify the view. With these two kinds of designs, the application does not embed a lot of data. The data is usually extracted from a database or a Web service call. For remote sources, a mobile application needs to be connected to collect them. Then, the quality of these services can not be certified because they are externals. This identifies one of the most important challenges, namely, how

Copyright (c) IARIA, 2017. ISBN: 978-1-61208-568-5

to maintain the distributed quality over the business logic of mobile applications.

Another challenge is the integration of third party libraries, like when using social networks for connection/identification. These libraries contain uncontrollable code, which has a high risk for quality criteria. They are often interdependent from the main application, and do not follow the releases of the platform's update. When changing an obsolete library, the developer has to verify the impacted code in the application, especially since it has a strong dependency to it.

These constraints produce a lot of bugs, like code quality bugs or integration bugs. In addition, there is currently a context difference between the development of the application and its actual use after release. Indeed, the context like the number of users or the stability of the remote services can change the final quality.

In this paper, we present a suitable solution to take these constrains into account.

III. State of Art

In this section, we discuss the different aspects needed to qualify the mobile software process.

A. Software product line

To understand how mobile software is made, let us have a look at the Software Product Line (SPL) defined by the Software Engineering Institute (SEI) to manage and organize a software product processing [12]. This SPL describes the different steps of the process. At the beginning, the entry points are the client needs (functionalities), used at the conception phase to determine the different implementations technologies. But it also helps to define the way functionalities will be isolated in different modules, relying on the components standards. These technical and functional requirements are done by experts. The components designed are then produced during the implementation phase. Finally, the test phase intends to validate the different components and functionalities created during the implementation phase. Afterwards, the release phase is triggered to distribute the final product. Continuous integration is the usual way to automate these phases. In Figure 1, we detail the common use of continuous integration. With some tools, we can automate some parts of the software's process. An orchestrator can play defined jobs to control source repositories or source codes with different versions, compile the code, run tests and delivery the final product. A feedback of all these operations can be provided to developers and to managers. These processes are associated with management methods from the way to develop a software, to the product team management. The team management can impact the quality of the process so we have to consider it. The size of the team and the development' speed time leads us to Scrum management methods [18]. Indeed, this method is specially well suited to these features as said in [22].

B. Software quality

Our research mainly focuses on the quality in mobile applications. Quality is an important characteristic of software.

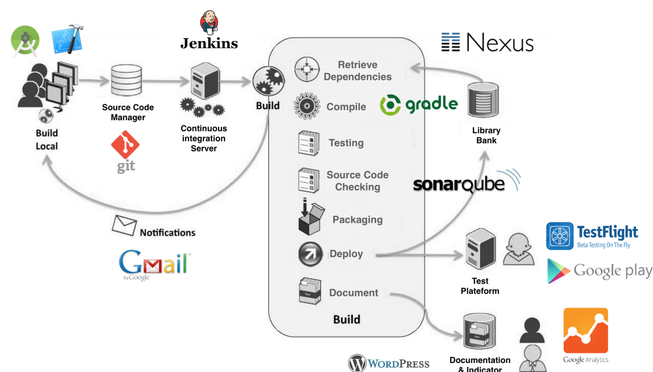

Figure 1: Continuous integration example

When many solutions exist for classic applications, the previous constraints are not embedded in these solutions. The need for a specific mobile quality model is real, as described in [22] or [24]. Software quality is a widespread subject already validated by different kinds of certifications.

Firstly, we have certifications based on the software company structure, like CMMI (Capability Maturity Model Integration)[21]. This approach evaluates the maturity of the company in order to determine its capability to produce quality software. This approach is not suitable for mobile application development because this kind of application is deployed quickly [15] and also because they are organization specific and our approach considers the source code level of the process. Secondly, we have production certifications, where the

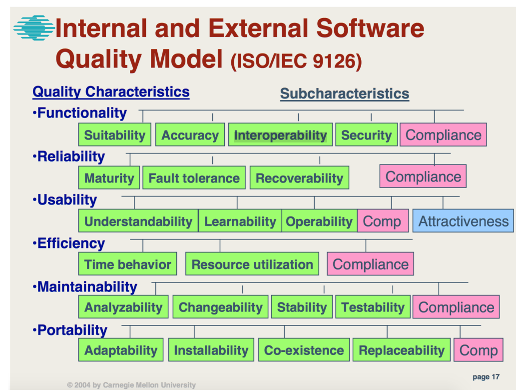

Figure 2: Square Software Quality criteria

most popular is Square from the ISO9126 certification [25]. The quality of the application is defined by rating different criteria, as described in Figure 2. In our approach, we are looking for validating these criteria by using mobile context checkpoints. When Square certification gives a criterion, we need to find a way to check it during mobile development. Some approaches define a mobile quality model. For example, Zahra [24] proposes to add data integrity notions to validate data consistency when the application is paused or stopped. But

Copyright (c) IARIA, 2017. ISBN: 978-1-61208-568-5

it is not enough. Because business logic distribution is strongly used in mobile software, the data integrity must be constant when the application interacts with other systems like servers. So, we have to generate checkpoints that match the quality criteria of the Square certification for every component in a mobile application. Finally, we have to generate checkpoints that match the quality of these components' implementation.

Based on our definition of the mobile development constraints II-B, we notice that every constraint can not match these criteria. The logic distribution as well as the high dependency to third party libraries are too specific and can not be associated with any criteria. For mobile software processes, we use two new criteria to define the quality of remote services, and the usability, quality and quality of integration, of a third party library.

All of these checkpoints need to be controlled so we need to monitor them and decide when we should process them. The next section will tackle these problems.

IV. Proposition

The first step to validate the checkpoints identified above, is defining the stages of a basic software product line [12] :

- Conception, to generate the applications architecture.
- Implementation, to produce the sources.
- And finally, testing, which is done by multiple actors.

But these stages are validated by testing the final product. Our approach is based on defining key points that have to be checked at each level of the software product line.

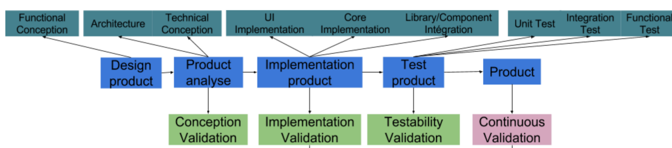

Figure 3: mobile software product line

Figure 3 shows a basic mobile software product line. We added, in green and purple, some steps for checkpoints validations. Thanks to them, the goal is to have, at the end of the process, a qualitative and sustainable mobile application to match the market constraints. These steps are going to be done simultaneously with the software product line ones, and automatically powered by continuous integration tools [7].

A. Conception validation

At first, we generate control points (based on the software engineering standards) for the conception phase of the application. This will allow to validate the usage of design patterns [23], libraries' dependencies and isolation of components [3]. This step will be done by experts. The organizational part of quality in a company is a key to permit these validations. Due to delay constraints imposed by the market, this validation could be time consuming. But agile project managements like Scrum provide time management solutions. An exemple of standard used in mobile development, is the component based development. A component [3] is a reusable self-working element. A mobile application is an aggregation of these components. Each component has its own properties and interfaces which allow other components to interact with him. The quality of this component has to be validated for all the constraints we saw above. As a component needs to work for every OS and version, it has to work against any environment and it can be modified without affecting the other components using it. When all the components are validated individually, and the integration of these components is validated too, we can validate the whole application.

B. Implementation validation

Then, we are looking for implementation validation, and this will be done in two ways. To guarantee that the implementation is matching the conception, we use different methods.

- Generate class diagram using UML (Unified Modeling Language) tools and compare them with the conception phase.
- Use pair programming to validate the implementation by different developers.
- Perform code revision, using version merge requests.

These approaches match with the organizational concept of quality. Afterwards, some differences persist between conception granularity and implementation caused by the technology environment. Because Android and Apple display guidelines to establish implementation standards, they also provide tools to check these rules. But these guidelines are not enough. For example, there is no guideline description to explain the best way to integrate a third-party library. The use of component based programming imposes some rules too. These new rules are suitable for mobile development and can easily be integrated in static analyse tools (like Android lint). They can also be easily integrated to a continuous integration platform. Furthermore, a component or library developer can embed these rules in their components or libraries. This analysis should be done on every code modification and automatically, based on different checkpoints we are going to formalize.

C. Test validation

Then, we check the validity of both unit and integration tests. As shown in our software product line with Figure 3, the tests can be split in three different domains, Unit, Integration or Functional. The objective here is to define proper tests, check implementations and finally run them over different devices, under different OS versions and different context of use (with or without network etc...). This allows us to validate that the software does what it intend to in each context. We have to define a severity threshold to reach for an application to be released. These tests have to be written manually, but can also be automatically recorded and played on several devices with different tools. Once again, component based engineering offers the possibility to embed tests with the component. The continuous integration platform should integrate these tools to automatically run tests.

Copyright (c) IARIA, 2017. ISBN: 978-1-61208-568-5

D. Continuous validation

These different validation steps can certify the quality of the application. We need to ensure that the conception and the implementation are sustainable and that the tests are continuously validating. For example, if just one month after the release, Android releases a new version of its OS, some of our checkpoints can be in a wrong state and our application does not validate any more. We need a way for measuring this impact without releasing a new version of the application. To achieve this, trackers will be added to the application source code to verify the different checkpoints sustainability like in [16]. Once these track events are set, they retrieve information about any potential anomaly and could alert the developer. These notions are already used for crash reporting and runtime healing like [16] but are not used yet to do the same treatment we exposed. This method is used to keep an eye on implementation, when the final user is interacting with the application. Thanks to data callback, we can determine the sustainability of a suitable implementation for evolutions.

The validation system has to be flexible, as some checkpoints are more important than others and priorities may vary during all the process time (configurable severity level).

E. Literature overview

Some papers report challenges like user acceptance [13], highlighting quality criteria from user experience, like Application interface design, Application performance, battery efficiency, phone features and connectivity cost. In [4], Dehlinger and Dixon point out the differences between classic applications and mobile applications, affecting the engineering process. Criteria like mobile screen size heterogeneity, platform heterogeneity etc. are a challenge for developers. Some papers propose a new definition of quality criteria like [10] for a specific branch of mobile app, the M-commerce, by questioning the user. Franke et al. propose a framework automating some existing quality check [9] and to extract a quality model ISO9126 [8]. The tradeoff between speed development and quality is discussed by Hansen [1]. It shows that Agile development is the best suited for mobile development and that quality automatic tools can be used to reduce cost/tile/risk in mobile applications. But the time spent to set up the automation and to maintain it costs more than quality control by different ways for small projects.

F. Realization

To illustrate our approach, we use the case of the Network Http request. As said above, mobile software relies on network to retrieve data from servers and to do this they use http request. We need an exhaustive list of our checkpoints in this use case. These checkpoints are defined by using the specifications of an http request like body, header and error code, pairing with the use case of http request and the development skills. These checkpoints can be embedded in third party libraries. When these checkpoints pass, we can be sure that the application can use an http request call or a library without errors or degrading the quality. In the conception part we have to ensure that the Http request is made by only one component (singleton). For the implementation part, we have to check if every checkpoint is validated by parsing the source code. For instance, we have to be sure that when a POST request is sent, the body part is filled. To finish, we add trackers to monitor the quantity and the reason of rejected requests. We have defined every parameter that compose an HTTP request to generate checkpoints and validate all the parameters for an Http request.

V. Conclusion

Our goal is to increase the quality of mobile applications. To achieve this, we identify a mostly exhaustive list of quality checkpoints extended from the quality model ISO9126 to verify. This quality level will be improved and simplified by relying on component models, from common mobile application functionalities to the core components. These checkpoints have to be validated during all the software process (conception, implementation, test and release) to guarantee its sustainability. A checkpoint, the way to validate it and the actors are led by the phase involved. Moreover, a tracking system is added to monitor checkpoint validation, even after the application's release. For now, we are working on implementing and validating checkpoints over a simple distributed application in a software product line.

References

[1] Hansen Aaron. A mobile software quality framework. Lionbridge Technologies, 2007.

[2] AppFigure. Mobile application quantity on appstore by statrista. http://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/, 2015.

[3] Xia Cai, Michael R Lyu, Kam-Fai Wong, and Roy Ko. Component-based software engineering: technologies, development frameworks, and quality assurance schemes. In *Software Engineering Conference, 2000. APSEC 2000. Proceedings. Seventh Asia-Pacific*, pages 372–379. IEEE, 2000.

[4] Josh Dehlinger and Jeremy Dixon. Mobile application software engineering: Challenges and research directions. In *Workshop on Mobile Software Engineering*, volume 2, pages 29–32, 2011.

[5] DeviceFigure. Mobile device quantity in 2015 by statrista. http://www.statista.com/statistics/274774/forecast-of-mobile-phone-users-worldwide/, 2015.

[6] Jared Dickson. Xamarin mobile development. 2013.

[7] Paul M Duvall. *Continuous integration*. Pearson Education India, 2007.

[8] Dominik Franke, Stefan Kowalewski, and Carsten Weise. A mobile software quality model. In *Quality Software (QSIC), 2012 12th International Conference on*, pages 154–157. IEEE, 2012.

[9] Dominik Franke and Carsten Weise. Providing a software quality framework for testing of mobile applications. In *Software Testing, Verification and Validation (ICST), 2011 IEEE Fourth International Conference on*, pages 431–434. IEEE, 2011.

[10] John D Garofalakis, Antonia Stefani, Vasilios Stefanis, and Michalis Nik Xenos. Quality attributes of consumer-based m-commerce systems. In *ICE-B*, pages 130–136, 2007.

[11] Google. Android 6.0 change. https://developer.android.com, 2015.

[12] Svein Hallsteinsen, Mike Hinchey, Sooyong Park, and Klaus Schmid. Dynamic software product lines. *Computer*, 41(4), 2008.

[13] Selim Ickin, Katarzyna Wac, Markus Fiedler, Lucjan Janowski, Jin-Hyuk Hong, and Anind K Dey. Factors influencing quality of experience of commonly used mobile applications. *IEEE Communications Magazine*, 50(4), 2012.

Copyright (c) IARIA, 2017. ISBN: 978-1-61208-568-5

[14] ionic. Ionic home page. http://ionicframework.com/, 2016.

[15] Mona Erfani Joorabchi, Ali Mesbah, and Philippe Kruchten. Real challenges in mobile app development. In *2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 15–24. IEEE, 2013.

[16] Renaud Pawlak, Carlos Noguera, and Nicolas Petitprez. *Spoon: Program analysis and transformation in java*. PhD thesis, Inria, 2006.

[17] Joachim Perchat, Mikael Desertot, and Sylvain Lecomte. Common framework: A hybrid approach to integrate cross-platform components in mobile application. *Journal of Computer Science*, 10(11):2165, 2014.

[18] Ken Schwaber and Jeff Sutherland. The scrum guide (2013). *http://www. scrum. org/Scrum-Guides¿. Acessado em*, 16:18, 2013.

[19] Kishori Sharan. Model-view-controller pattern. In *Learn JavaFX 8*, pages 419–434. Springer, 2015.

[20] Artem Syromiatnikov and Danny Weyns. A journey through the land of model-view-design patterns. In *Software Architecture (WICSA), 2014 IEEE/IFIP Conference on*, pages 21–30. IEEE, 2014.

[21] CMMI Product Team. Cmmi for development, version 1.2. 2006.

[22] Anthony I Wasserman. Software engineering issues for mobile application development. In *Proceedings of the FSE/SDP workshop on Future of software engineering research*, pages 397–400. ACM, 2010.

[23] Pree Wolfgang. *Design patterns for object-oriented software development*. Reading, Mass.: Addison-Wesley, 1994.

[24] Sobia Zahra, Asra Khalid, and Ali Javed. An efficient and effective new generation objective quality model for mobile applications. *International Journal of Modern Education and Computer Science*, 5(4):36, 2013.

[25] Dave Zubrow. Software quality requirements and evaluation, the iso 25000 series. *Software Engineering Institute, Carnegie Mellon*, 2004.

Copyright (c) IARIA, 2017. ISBN: 978-1-61208-568-5